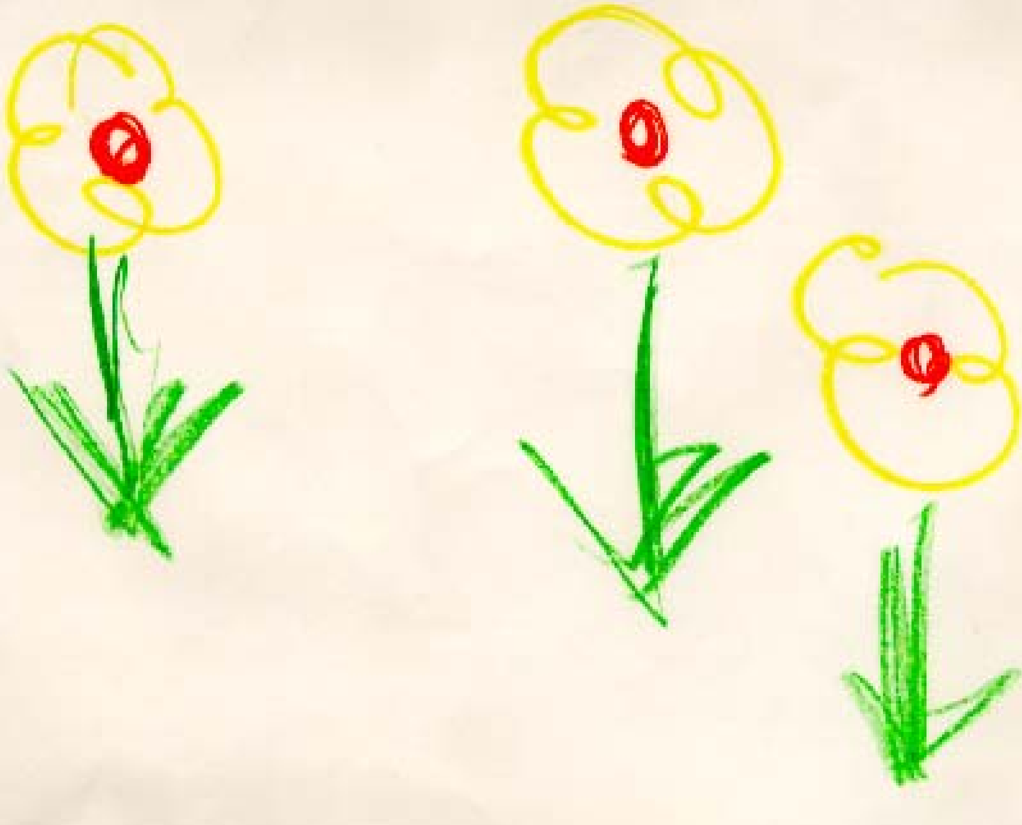
CS 373

Name: Johnny
alias: pikachu

HW1



1

Trees have no cycles but must be connected.
Tourn'ments are cliques with their edges directed.
Hamilton circuits, Eulerian paths,
These are a few of my favorite graphs.

Matchings and bicliques and blossoms and bases,
Kempe chains, hypercubes, forests, and faces,
AKS networks that split into halves,
These are a few of my favorite graphs.

Short paths of co-authors leading to Erdos,
Large neural networks that translate from Kurdish,
Finite projective planes - they make me laugh.
These are a few of my favorite graphs.

Chorus: Propositions, corollaries,
Problems that are starred,
I simply remember my favorite graphs
And then they don't seem so hard.

If there's no K_5 or K_{3,3} minor,
Old Kuratowski says it'll be 'plinor'.
Four's enough colors if there are no gaffes!
These are a few of my favorite graphs.

Quadrangles, thrackles, and triangulations,
Minor-closed families and sparsifications,
Voronoi diagrams found on giraffes,
These are a few of my favorite graphs.

Chorus: Propositions, corollaries,
When they're just too deep,
I simply remember my favorite graphs
And then I go right to sleep.

Look! I minimized the slop of my favourite graph song!

2.

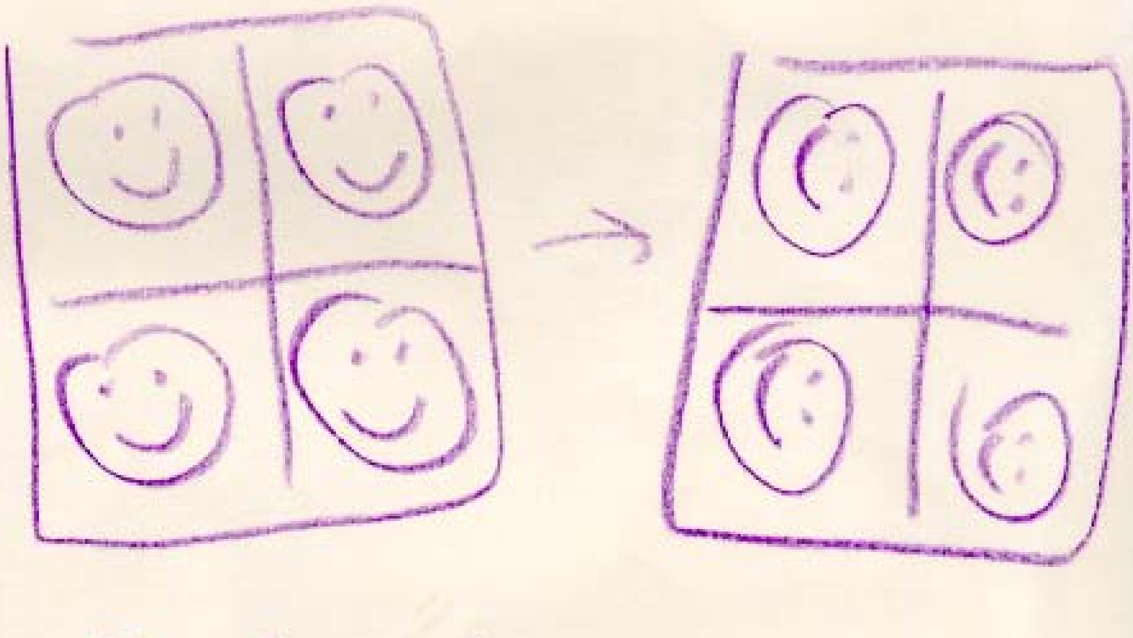
Mamma always said:
Stupid is as stupid does.

My name
is Clifford.
I'm a
red
big
dog.

3. This problem is the work of Satan. It is very scary. I was too afraid to do it. Sorry.

4. I betcha that:

$\therefore$



QED.

5. I am running this algorithm with MGD. Will report results...

much beer drinking

more analysis needed

.....

nlgn is like a cheese log

~ I'm hungry, call if I bring back

~ this is funny

~ beer.

CS 373: Combinatorial Algorithms, Spring 1999

http://www-courses.cs.uiuc.edu/ cs373

Homework 0 (due January 26, 1999 by the beginning of class)

Name:	
Net ID:	Alias:

Neatly print your name (first name first, with no comma), your network ID, and a short alias into the boxes above. Do not *sign* your name. Do not write your Social Security number. Staple this sheet of paper to the top of your homework. Grades will be listed on the course web site by alias, so your alias should not resemble your name (or your Net ID). If you do not give yourself an alias, you will be stuck with one we give you, no matter how much you hate it.

Everyone must do the problems marked ▶. Problems marked ▷ are for 1-unit grad students and others who want extra credit. (There's no such thing as "partial extra credit"!) Unmarked problems are extra practice problems for your benefit, which will not be graded. Think of them as potential exam questions.

Hard problems are marked with a star; the bigger the star, the harder the problem.

This homework tests your familiarity with the prerequisite material from CS 225 and CS 273 (and *their* prerequisites)—many of these problems appeared on homeworks and/or exams in those classes—primarily to help you identify gaps in your knowledge. **You are responsible for filling those gaps on your own.**

Undergrad/.75U Grad/1U Grad Problems

▶1. **[173/273]**

(a) Prove that any positive integer can be written as the sum of distinct powers of 2. (For example: $42 = 2^5 + 2^3 + 2^1$, $25 = 2^4 + 2^3 + 2^0$, $17 = 2^4 + 2^0$.)

(b) Prove that any positive integer can be written as the sum of distinct *nonconsecutive* Fibonacci numbers—if F_n appears in the sum, then neither F_{n+1} nor F_{n-1} will. (For example: $42 = F_9 + F_6$, $25 = F_8 + F_4 + F_2$, $17 = F_7 + F_4 + F_2$.)

(c) Prove that *any* integer can be written in the form $\sum_i \pm 3^i$, where the exponents i are distinct non-negative integers. (For example: $42 = 3^4 - 3^3 - 3^2 - 3^1$, $25 = 3^3 - 3^1 + 3^0$, $17 = 3^3 - 3^2 - 3^0$.)

▶2. **[225/273]** Sort the following functions from asymptotically smallest to largest, indicating ties if there are any: n, $\lg n$, $\lg\lg^* n$, $\lg^*\lg n$, $\lg^* n$, $n\lg n$, $\lg(n\lg n)$, $n^{n/\lg n}$, $n^{\lg n}$, $(\lg n)^n$, $(\lg n)^{\lg n}$, $2^{\sqrt{\lg n \lg\lg n}}$, 2^n, $n^{\lg\lg n}$, $\sqrt[1000]{n}$, $(1+\frac{1}{1000})^n$, $(1-\frac{1}{1000})^n$, $\lg^{1000} n$, $\lg^{(1000)} n$, $\log_{1000} n$, $\lg^n 1000$, 1.

[To simplify notation, write $f(n) \ll g(n)$ to mean $f(n) = o(g(n))$ and $f(n) \equiv g(n)$ to mean $f(n) = \Theta(g(n))$. For example, the functions n^2, n, $\binom{n}{2}$, n^3 could be sorted as follows: $n \ll n^2 \equiv \binom{n}{2} \ll n^3$.]

3. **[273/225]** Solve the following recurrences. State tight asymptotic bounds for each function in the form $\Theta(f(n))$ for some recognizable function $f(n)$. You do not need to turn in proofs (in fact, please *don't* turn in proofs), but you should do them anyway just for practice. Assume reasonable (nontrivial) base cases. Extra credit will be given for more exact solutions.

   ►(a) $A(n) = A(n/2) + n$

   (b) $B(n) = 2B(n/2) + n$

   ►(c) $C(n) = 3C(n/2) + n$

   (d) $D(n) = \max\limits_{n/3<k<2n/3} \big(D(k) + D(n-k) + n\big)$

   ►(e) $E(n) = \min\limits_{0<k<n} \big(E(k) + E(n-k) + 1\big)$

   (f) $F(n) = 4F(\lfloor n/2 \rfloor + 5) + n$

   ►(g) $G(n) = G(n-1) + 1/n$

   ⋆(h) $H(n) = H(n/2) + H(n/4) + H(n/6) + H(n/12) + n$ [Hint: $\frac{1}{2} + \frac{1}{4} + \frac{1}{6} + \frac{1}{12} = 1$.]

   ►⋆(i) $I(n) = 2I(n/2) + n/\lg n$

   ⋆(j) $J(n) = \dfrac{J(n-1)}{J(n-2)}$

►4. **[273]** Alice and Bob each have a fair n-sided die. Alice rolls her die once. Bob then repeatedly throws his die until the number he rolls is at least as big as the number Alice rolled. Each time Bob rolls, he pays Alice $\$1$. (For example, if Alice rolls a 5, and Bob rolls a 4, then a 3, then a 1, then a 5, the game ends and Alice gets $\$4$. If Alice rolls a 1, then no matter what Bob rolls, the game will end immediately, and Alice will get $\$1$.)

   Exactly how much money does Alice expect to win at this game? Prove that your answer is correct. (If you have to appeal to "intuition" or "common sense", your answer is probably wrong.)

►5. **[225]** George has a 26-node binary tree, with each node labeled by a unique letter of the alphabet. The preorder and postorder sequences of nodes are as follows:

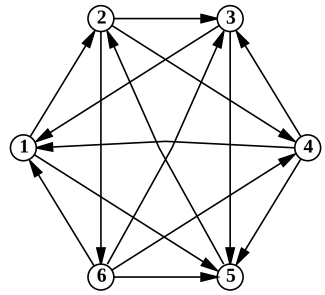
   preorder: `M N H C R S K W T G D X I Y A J P O E Z V B U L Q F`
   postorder: `C W T K S G R H D N A O E P J Y Z I B Q L F U V X M`

   Draw George's binary tree.

Only 1U Grad Problems

▷⋆1. **[225/273]** A *tournament* is a directed graph with exactly one edge between every pair of vertices. (Think of the nodes as players in a round-robin tournament, where each edge points from the winner to the loser.) A *Hamiltonian path* is a sequence of directed edges, joined end to end, that visits every vertex exactly once.

   Prove that every tournament contains at least one Hamiltonian path.



A six-vertex tournament containing the Hamiltonian path $6 \to 4 \to 5 \to 2 \to 3 \to 1$.

Practice Problems

1. **[173/273]** Recall the standard recursive definition of the Fibonacci numbers: $F_0 = 0$, $F_1 = 1$, and $F_n = F_{n-1} + F_{n-2}$ for all $n \geq 2$. Prove the following identities for all positive integers n and m.

   (a) F_n is even if and only if n is divisible by 3.

   (b) $\sum_{i=0}^{n} F_i = F_{n+2} - 1$

   (c) $F_n^2 - F_{n+1}F_{n-1} = (-1)^{n+1}$

   ★(d) If n is an integer multiple of m, then F_n is an integer multiple of F_m.

2. **[225/273]**

   (a) Prove that $2^{\lceil \lg n \rceil + \lfloor \lg n \rfloor}/n = \Theta(n)$.

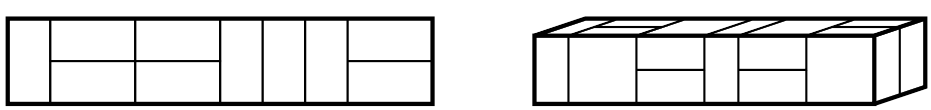
   (b) Is $2^{\lfloor \lg n \rfloor} = \Theta\big(2^{\lceil \lg n \rceil}\big)$? Justify your answer.

   (c) Is $2^{2^{\lfloor \lg \lg n \rfloor}} = \Theta\big(2^{2^{\lceil \lg \lg n \rceil}}\big)$? Justify your answer.

3. **[273]**

   (a) A *domino* is a 2×1 or 1×2 rectangle. How many different ways are there to completely fill a $2 \times n$ rectangle with n dominos?

   (b) A *slab* is a three-dimensional box with dimensions $1 \times 2 \times 2$, $2 \times 1 \times 2$, or $2 \times 2 \times 1$. How many different ways are there to fill a $2 \times 2 \times n$ box with n slabs? Set up a recurrence relation and give an *exact* closed-form solution.



A 2×10 rectangle filled with ten dominos, and a $2 \times 2 \times 10$ box filled with ten slabs.

4. **[273]** Penn and Teller have a special deck of fifty-two cards, with no face cards and nothing but clubs—the ace, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, ..., 52 of clubs. (They're big cards.) Penn shuffles the deck until each each of the 52! possible orderings of the cards is equally likely. He then takes cards one at a time from the top of the deck and gives them to Teller, stopping as soon as he gives Teller the three of clubs.

(a) On average, how many cards does Penn give Teller?

(b) On average, what is the smallest-numbered card that Penn gives Teller?

*(c) On average, what is the largest-numbered card that Penn gives Teller?

Prove that your answers are correct. (If you have to appeal to "intuition" or "common sense", your answers are probably wrong.) [Hint: Solve for an n-card deck, and then set n to 52.]

5. **[273/225]** Prove that for any nonnegative parameters a and b, the following algorithms terminate and produce identical output.

```
SlowEuclid(a, b) :
    if b > a
        return SlowEuclid(b, a)
    else if b == 0
        return a
    else
        return SlowEuclid(a, b − a)
```

```
FastEuclid(a, b) :
    if b == 0
        return a
    else
        return FastEuclid(b, a mod b)
```

CS 373: Combinatorial Algorithms, Spring 1999

http://www-courses.cs.uiuc.edu/~cs373

Homework 1 (due February 9, 1999 by noon)

Name:	
Net ID:	Alias:

Everyone must do the problems marked ▶. Problems marked ▷ are for 1-unit grad students and others who want extra credit. (There's no such thing as "partial extra credit"!) Unmarked problems are extra practice problems for your benefit, which will not be graded. Think of them as potential exam questions.

Hard problems are marked with a star; the bigger the star, the harder the problem.

Note: When a question asks you to "give/describe/present an algorithm", you need to do four things to receive full credit:

1. Design the most efficient algorithm possible. Significant partial credit will be given for less efficient algorithms, as long as they are still correct, well-presented, and correctly analyzed.
2. Describe your algorithm succinctly, using structured English/pseudocode. We don't want full-fledged compilable source code, but plain English exposition is usually not enough. Follow the examples given in the textbooks, lectures, homeworks, and handouts.
3. Justify the correctness of your algorithm, including termination if that is not obvious.
4. Analyze the time and space complexity of your algorithm.

Undergrad/.75U Grad/1U Grad Problems

▶1. Consider the following sorting algorithm:

```
StupidSort(A[0 .. n − 1]) :
  if n = 2 and A[0] > A[1]
      swap A[0] ↔ A[1]
  else if n > 2
      m = ⌈2n/3⌉
      StupidSort(A[0 .. m − 1])
      StupidSort(A[n − m .. n − 1])
      StupidSort(A[0 .. m − 1])
```

(a) Prove that StupidSort actually sorts its input.

(b) Would the algorithm still sort correctly if we replaced $m = \lceil 2n/3 \rceil$ with $m = \lfloor 2n/3 \rfloor$? Justify your answer.

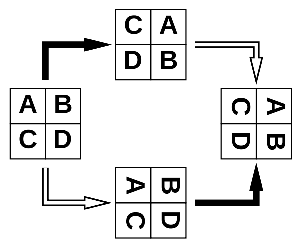
(c) State a recurrence (including the base case(s)) for the number of comparisons executed by StupidSort.

(d) Solve the recurrence, and prove that your solution is correct. [Hint: Ignore the ceiling.] Does the algorithm deserve its name?

*(e) Show that the number of *swaps* executed by StupidSort is at most $\binom{n}{2}$.

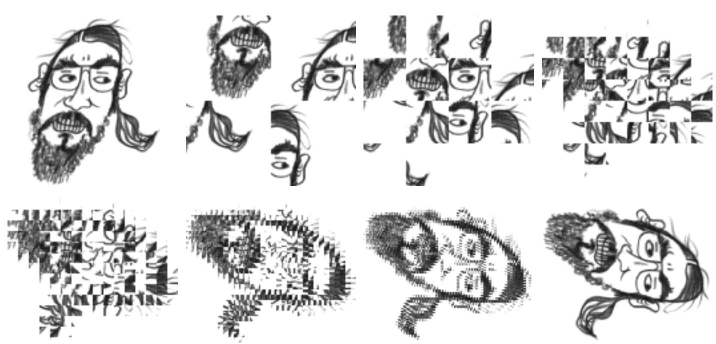
►2. Some graphics hardware includes support for an operation called *blit*, or **bl**ock **t**ransfer, which quickly copies a rectangular chunk of a pixelmap (a two-dimensional array of pixel values) from one location to another. This is a two-dimensional version of the standard C library function `memcpy()`.

Suppose we want to rotate an $n \times n$ pixelmap $90°$ clockwise. One way to do this is to split the pixelmap into four $n/2 \times n/2$ blocks, move each block to its proper position using a sequence of five blits, and then recursively rotate each block. Alternately, we can first recursively rotate the blocks and blit them into place afterwards.



Two algorithms for rotating a pixelmap.
Black arrows indicate blitting the blocks into place.
White arrows indicate recursively rotating the blocks.

The following sequence of pictures shows the first algorithm (blit then recurse) in action.



In the following questions, assume n is a power of two.

(a) Prove that both versions of the algorithm are correct.

(b) *Exactly* how many blits does the algorithm perform?

(c) What is the algorithm's running time if a $k \times k$ blit takes $O(k^2)$ time?

(d) What if a $k \times k$ blit takes only $O(k)$ time?

►3. Dynamic Programming: The Company Party
A company is planning a party for its employees. The organizers of the party want it to be a fun party, and so have assigned a 'fun' rating to every employee. The employees are organized into a strict hierarchy, i.e. a tree rooted at the president. There is one restriction, though, on the guest list to the party: both an employee and their immediate supervisor (parent in the tree) cannot both attend the party (because that would be no fun at all). Give an algorithm that makes a guest list for the party that maximizes the sum of the 'fun' ratings of the guests.

►4. Dynamic Programming: Longest Increasing Subsequence (LIS)
Give an $O(n^2)$ algorithm to find the longest increasing subsequence of a sequence of numbers. Note: the elements of the subsequence need not be adjacent in the sequence. For example, the sequence $(1, 5, 3, 2, 4)$ has an LIS $(1, 3, 4)$.

►5. Nut/Bolt Median
You are given a set of n nuts and n bolts of different sizes. Each nut matches exactly one bolt (and vice versa, of course). The sizes of the nuts and bolts are so similar that you cannot compare two nuts or two bolts to see which is larger. You can, however, check whether a nut is too small, too large, or just right for a bolt (and vice versa, of course).

In this problem, your goal is to find the median bolt (*i.e.*, the $\lfloor n/2 \rfloor$th largest) as quickly as possible.

(a) Describe an efficient deterministic algorithm that finds the median bolt. How many nut-bolt comparisons does your algorithm perform in the worst case?

(b) Describe an efficient *randomized* algorithm that finds the median bolt.

  i. State a recurrence for the expected number of nut/bolt comparisons your algorithm performs.

  ii. What is the probability that your algorithm compares the ith largest bolt with the jth largest nut?

  iii. What is the expected number of nut-bolt comparisons made by your algorithm? [Hint: Use your answer to either of the previous two questions.]

Only 1U Grad Problems

▷1. You are at a political convention with n delegates. Each delegate is a member of exactly one political party. It is impossible to tell which political party a delegate belongs to. However, you can check whether any two delegates are in the *same* party or not by introducing them to each other. (Members of the same party always greet each other with smiles and friendly handshakes; members of different parties always greet each other with angry stares and insults.)

(a) Suppose a majority (more than half) of the delegates are from the same political party. Give an efficient algorithm that identifies a member of the majority party.

*(b) Suppose exactly k political parties are represented at the convention and one party has a *plurality*: more delegates belong to that party than to any other. Give an efficient algorithm that identifies a member of the plurality party.

*(c) Suppose you don't know how many parties there are, but you do know that one party has a plurality, and at least p people in the plurality party are present. Present a practical procedure to pick a person from the plurality party as parsimoniously as possible. (Please.)

$^{\bigstar}$(d) Finally, suppose you don't know how many parties are represented at the convention, and you don't know how big the plurality is. Give an efficient algorithm to identify a member of the plurality party. How is the running time of your algorithm affected by the number of parties (k)? By the size of the plurality (p)?

Practice Problems

1. Second Smallest
   Give an algorithm that finds the *second* smallest of n elements in at most $n + \lceil \lg n \rceil - 2$ comparisons. Hint: divide and conquer to find the smallest; where is the second smallest?

2. Linear in-place 0-1 sorting
   Suppose that you have an array of records whose keys to be sorted consist only of 0's and 1's. Give a simple, linear-time $O(n)$ algorithm to sort the array in place (using storage of no more than constant size in addition to that of the array).

3. Dynamic Programming: Coin Changing
   Consider the problem of making change for n cents using the least number of coins.

   (a) Describe a greedy algorithm to make change consisting of quarters, dimes, nickels, and pennies. Prove that your algorithm yields an optimal solution.

   (b) Suppose that the available coins have the values $c^0, c^1, \ldots, c^k$ for some integers $c > 1$ and $k \geq 1$. Show that the greedy algorithm always yields an optimal solution.

   (c) Give a set of 4 coin values for which the greedy algorithm does not yield an optimal solution, show why.

   (d) Give a dynamic programming algorithm that yields an optimal solution for an arbitrary set of coin values.

   (e) Prove that, with only two coins a, b whose gcd is 1, the smallest value n for which change *can* be given for all values greater than or equal to n is $(a-1)(b-1)$.

   $^{\bigstar}$(f) For only three coins a, b, c whose gcd is 1, give an algorithm to determine the smallest value n for which change *can* be given for all values greater than n. (note: this problem is currently unsolved for $n > 4$.

4. Dynamic Programming: Paragraph Justification
   Consider the problem of printing a paragraph neatly on a printer (with fixed width font). The input text is a sequence of n words of lengths $l_1, l_2, \ldots, l_n$. The line length is M (the maximum # of characters per line). We expect that the paragraph is left justified, that all first words on a line start at the leftmost position and that there is exactly one space between any two words on the same line. We want the uneven right ends of all the lines to be together as 'neat' as possible. Our criterion of neatness is that we wish to minimize the sum, over all lines except the last, of the cubes of the numbers of extra space characters at the ends of the lines. Note: if a printed line contains words i through j, then the number of spaces at the end of the line is $M - j + i - \sum_{k=i}^{j} l_k$.

   (a) Give a dynamic programming algorithm to do this.

   (b) Prove that if the neatness function is linear, a linear time greedy algorithm will give an optimum 'neatness'.

5. Comparison of Amortized Analysis Methods
   A sequence of n operations is performed on a data structure. The ith operation costs i if i is an exact power of 2, and 1 otherwise. That is operation i costs $f(i)$, where:

$$f(i) = \begin{cases} i, & i = 2^k, \\ 1, & \text{otherwise} \end{cases}$$

   Determine the amortized cost per operation using the following methods of analysis:

   (a) Aggregate method

   (b) Accounting method

   *(c) Potential method

CS 373: Combinatorial Algorithms, Spring 1999

`http://www-courses.cs.uiuc.edu/~cs373`

Homework 2 (due Thu. Feb. 18, 1999 by noon)

Name:	
Net ID:	Alias:

Everyone must do the problems marked ►. Problems marked ▷ are for 1-unit grad students and others who want extra credit. (There's no such thing as "partial extra credit"!) Unmarked problems are extra practice problems for your benefit, which will not be graded. Think of them as potential exam questions.

Hard problems are marked with a star; the bigger the star, the harder the problem.

Note: When a question asks you to "give/describe/present an algorithm", you need to do four things to receive full credit:

1. Design the most efficient algorithm possible. Significant partial credit will be given for less efficient algorithms, as long as they are still correct, well-presented, and correctly analyzed.
2. Describe your algorithm succinctly, using structured English/pseudocode. We don't want full-fledged compilable source code, but plain English exposition is usually not enough. Follow the examples given in the textbooks, lectures, homeworks, and handouts.
3. Justify the correctness of your algorithm, including termination if that is not obvious.
4. Analyze the time and space complexity of your algorithm.

Undergrad/.75U Grad/1U Grad Problems

►1. Faster Longest Increasing Subsequence (LIS)
Give an $O(n \log n)$ algorithm to find the longest increasing subsequence of a sequence of numbers. Hint: In the dynamic programming solution, you don't really have to look back at all previous items.

►2. SELECT(A, k)
Say that a binary search tree is *augmented* if every node v also stores $|v|$, the size of its subtree.

(a) Show that a rotation in an augmented binary tree can be performed in constant time.

(b) Describe an algorithm SCAPEGOATSELECT(k) that selects the kth smallest item in an augmented scapegoat tree in $O(\log n)$ *worst-case* time. (The scapegoat trees presented in class were already augmented.)

(c) Describe an algorithm SPLAYSELECT(k) that selects the kth smallest item in an augmented splay tree in $O(\log n)$ *amortized* time.

(d) Describe an algorithm TREAPSELECT(k) that selects the kth smallest item in an augmented treap in $O(\log n)$ *expected* time.

►3. Scapegoat trees

(a) Prove that only one tree gets rebalanced at any insertion.

(b) Prove that $I(v) = 0$ in every node of a perfectly balanced tree ($I(v) = max(0, |\hat{v}| - |\check{v}|)$, where $\hat{v}$ is the child of greater height and $\check{v}$ the child of lesser height, $|v|$ is the number of nodes in subtree v, and perfectly balanced means each subtree has as close to half the leaves as possible and is perfectly balanced itself.

*(c) Show that you can rebuild a fully balanced binary tree in $O(n)$ time using only $O(1)$ additional memory.

►4. Memory Management

Suppose we can insert or delete an element into a hash table in constant time. In order to ensure that our hash table is always big enough, without wasting a lot of memory, we will use the following global rebuilding rules:

- After an insertion, if the table is more than 3/4 full, we allocate a new table twice as big as our current table, insert everything into the new table, and then free the old table.
- After a deletion, if the table is less than 1/4 full, we we allocate a new table half as big as our current table, insert everything into the new table, and then free the old table.

Show that for any sequence of insertions and deletions, the amortized time per operation is still a constant. Do not use the potential method (it makes it much more difficult).

Only 1U Grad Problems

▷1. Detecting overlap

(a) You are given a list of ranges represented by min and max (e.g. [1,3], [4,5], [4,9], [6,8], [7,10]). Give an $O(n \log n)$-time algorithm that decides whether or not a set of ranges contains a pair that overlaps. You need not report all intersections. If a range completely covers another, they are overlapping, even if the boundaries do not intersect.

(b) You are given a list of rectangles represented by min and max x- and y- coordinates. Give an $O(n \log n)$-time algorithm that decides whether or not a set of rectangles contains a pair that overlaps (with the same qualifications as above). Hint: sweep a vertical line from left to right, performing some processing whenever an end-point is encountered. Use a balanced search tree to maintain any extra info you might need.

Practice Problems

1. Amortization

   (a) Modify the binary double-counter (see class notes Feb. 2) to support a new operation Sign, which determines whether the number being stored is positive, negative, or zero, in constant time. The amortized time to increment or decrement the counter should still be a constant.

   [Hint: Suppose p is the number of significant bits in P, and n is the number of significant bits in N. For example, if $P = 17 = 10001_2$ and $N = 0$, then $p = 5$ and $n = 0$. Then $p - n$ always has the same sign as $P - N$. Assume you can update p and n in $O(1)$ time.]

   *(b) Do the same but now you can't assume that p and n can be updated in $O(1)$ time.

*2. Amortization
Suppose instead of powers of two, we represent integers as the sum of Fibonacci numbers. In other words, instead of an array of bits, we keep an array of "fits", where the ith least significant fit indicates whether the sum includes the ith Fibonacci number F_i. For example, the fit string 101110 represents the number $F_6 + F_4 + F_3 + F_2 = 8 + 3 + 2 + 1 = 14$. Describe algorithms to increment and decrement a fit string in constant amortized time. [Hint: Most numbers can be represented by more than one fit string. This is *not* the same representation as on Homework 0.]

3. Rotations

   (a) Show that it is possible to transform any n-node binary search tree into any other n-node binary search tree using at most $2n - 2$ rotations.

   *(b) Use fewer than $2n - 2$ rotations. Nobody knows how few rotations are required in the worst case. There is an algorithm that can transform any tree to any other in at most $2n - 6$ rotations, and there are pairs of trees that are $2n - 10$ rotations apart. These are the best bounds known.

4. Fibonacci Heaps: SECONDMIN
We want to find the second smallest of a set efficiently.

   (a) Implement SECONDMIN by using a Fibonacci heap as a black box. Remember to justify its correctness and running time.

   *(b) Modify the Fibonacci Heap data structure to implement SECONDMIN in constant time.

5. Give an efficient implementation of the operation **Fib-Heap-Change-Key(**H, x, k**)**, which changes the key of a node x in a Fibonacci heap H to the value k. The changes you make to Fibonacci heap data structure to support your implementation should not affect the amortized running time of any other Fibonacci heap operations. Analyze the amortized running time of your implementation for cases in which k is greater than, less than, or equal to $key[x]$.

CS 373: Combinatorial Algorithms, Spring 1999

http://www-courses.cs.uiuc.edu/~cs373

Homework 2 (due Thu. Feb. 18, 1999 by noon)

Name:	
Net ID:	Alias:

Everyone must do the problems marked ▶. Problems marked ▷ are for 1-unit grad students and others who want extra credit. (There's no such thing as "partial extra credit"!) Unmarked problems are extra practice problems for your benefit, which will not be graded. Think of them as potential exam questions.

Hard problems are marked with a star; the bigger the star, the harder the problem.

Note: When a question asks you to "give/describe/present an algorithm", you need to do four things to receive full credit:

1. Design the most efficient algorithm possible. Significant partial credit will be given for less efficient algorithms, as long as they are still correct, well-presented, and correctly analyzed.
2. Describe your algorithm succinctly, using structured English/pseudocode. We don't want full-fledged compilable source code, but plain English exposition is usually not enough. Follow the examples given in the textbooks, lectures, homeworks, and handouts.
3. Justify the correctness of your algorithm, including termination if that is not obvious.
4. Analyze the time and space complexity of your algorithm.

Undergrad/.75U Grad/1U Grad Problems

▶1. Faster Longest Increasing Subsequence (LIS)
Give an $O(n \log n)$ algorithm to find the longest increasing subsequence of a sequence of numbers. Hint: In the dynamic programming solution, you don't really have to look back at all previous items.

▶2. SELECT(A, k)
Say that a binary search tree is *augmented* if every node v also stores $|v|$, the size of its subtree.

(a) Show that a rotation in an augmented binary tree can be performed in constant time.

(b) Describe an algorithm SCAPEGOATSELECT(k) that selects the kth smallest item in an augmented scapegoat tree in $O(\log n)$ *worst-case* time. (The scapegoat trees presented in class were already augmented.)

(c) Describe an algorithm SPLAYSELECT(k) that selects the kth smallest item in an augmented splay tree in $O(\log n)$ *amortized* time.

(d) Describe an algorithm TREAPSELECT(k) that selects the kth smallest item in an augmented treap in $O(\log n)$ *expected* time.

►3. Scapegoat trees

(a) Prove that only one tree gets rebalanced at any insertion.

(b) Prove that $I(v) = 0$ in every node of a perfectly balanced tree ($I(v) = max(0, |\hat{v}| - |\check{v}|)$, where $\hat{v}$ is the child of greater height and $\check{v}$ the child of lesser height, $|v|$ is the number of nodes in subtree v, and perfectly balanced means each subtree has as close to half the leaves as possible and is perfectly balanced itself.

*(c) Show that you can rebuild a fully balanced binary tree in $O(n)$ time using only $O(1)$ additional memory.

►4. Memory Management

Suppose we can insert or delete an element into a hash table in constant time. In order to ensure that our hash table is always big enough, without wasting a lot of memory, we will use the following global rebuilding rules:

- After an insertion, if the table is more than 3/4 full, we allocate a new table twice as big as our current table, insert everything into the new table, and then free the old table.
- After a deletion, if the table is less than 1/4 full, we we allocate a new table half as big as our current table, insert everything into the new table, and then free the old table.

Show that for any sequence of insertions and deletions, the amortized time per operation is still a constant. Do not use the potential method (it makes it much more difficult).

Only 1U Grad Problems

▷1. Detecting overlap

(a) You are given a list of ranges represented by min and max (e.g. [1,3], [4,5], [4,9], [6,8], [7,10]). Give an $O(n \log n)$-time algorithm that decides whether or not a set of ranges contains a pair that overlaps. You need not report all intersections. If a range completely covers another, they are overlapping, even if the boundaries do not intersect.

(b) You are given a list of rectangles represented by min and max x- and y- coordinates. Give an $O(n \log n)$-time algorithm that decides whether or not a set of rectangles contains a pair that overlaps (with the same qualifications as above). Hint: sweep a vertical line from left to right, performing some processing whenever an end-point is encountered. Use a balanced search tree to maintain any extra info you might need.

Practice Problems

1. Amortization

   (a) Modify the binary double-counter (see class notes Feb. 2) to support a new operation Sign, which determines whether the number being stored is positive, negative, or zero, in constant time. The amortized time to increment or decrement the counter should still be a constant.

   [Hint: Suppose p is the number of significant bits in P, and n is the number of significant bits in N. For example, if $P = 17 = 10001_2$ and $N = 0$, then $p = 5$ and $n = 0$. Then $p - n$ always has the same sign as $P - N$. Assume you can update p and n in $O(1)$ time.]

   *(b) Do the same but now you can't assume that p and n can be updated in $O(1)$ time.

*2. Amortization
Suppose instead of powers of two, we represent integers as the sum of Fibonacci numbers. In other words, instead of an array of bits, we keep an array of ”fits”, where the ith least significant fit indicates whether the sum includes the ith Fibonacci number F_i. For example, the fit string 101110 represents the number $F_6 + F_4 + F_3 + F_2 = 8 + 3 + 2 + 1 = 14$. Describe algorithms to increment and decrement a fit string in constant amortized time. [Hint: Most numbers can be represented by more than one fit string. This is *not* the same representation as on Homework 0.]

3. Rotations

   (a) Show that it is possible to transform any n-node binary search tree into any other n-node binary search tree using at most $2n - 2$ rotations.

   *(b) Use fewer than $2n - 2$ rotations. Nobody knows how few rotations are required in the worst case. There is an algorithm that can transform any tree to any other in at most $2n - 6$ rotations, and there are pairs of trees that are $2n - 10$ rotations apart. These are the best bounds known.

4. Fibonacci Heaps: SECONDMIN
We want to find the second smallest of a set efficiently.

   (a) Implement SECONDMIN by using a Fibonacci heap as a black box. Remember to justify its correctness and running time.

   *(b) Modify the Fibonacci Heap data structure to implement SECONDMIN in constant time.

5. Give an efficient implementation of the operation **Fib-Heap-Change-Key(**H, x, k**)**, which changes the key of a node x in a Fibonacci heap H to the value k. The changes you make to Fibonacci heap data structure to support your implementation should not affect the amortized running time of any other Fibonacci heap operations. Analyze the amortized running time of your implementation for cases in which k is greater than, less than, or equal to $key[x]$.

CS 373: Combinatorial Algorithms, Spring 1999

`http://www-courses.cs.uiuc.edu/~cs373`

Homework 3 (due Thu. Mar. 11, 1999 by noon)

Name:	
Net ID:	Alias:

Everyone must do the problems marked ▶. Problems marked ▷ are for 1-unit grad students and others who want extra credit. (There's no such thing as "partial extra credit"!) Unmarked problems are extra practice problems for your benefit, which will not be graded. Think of them as potential exam questions.

Hard problems are marked with a star; the bigger the star, the harder the problem.

Note: When a question asks you to "give/describe/present an algorithm", you need to do four things to receive full credit:

1. (New!) If not already done, model the problem appropriately. Often the problem is stated in real world terms; give a more rigorous description of the problem. This will help you figure out what is assumed (what you know and what is arbitrary, what operations are and are not allowed), and find the tools needed to solve the problem.
2. Design the most efficient algorithm possible. Significant partial credit will be given for less efficient algorithms, as long as they are still correct, well-presented, and correctly analyzed.
3. Describe your algorithm succinctly, using structured English/pseudocode. We don't want full-fledged compilable source code, but plain English exposition is usually not enough. Follow the examples given in the textbooks, lectures, homeworks, and handouts.
4. Justify the correctness of your algorithm, including termination if that is not obvious.
5. Analyze the time and space complexity of your algorithm.

Undergrad/.75U Grad/1U Grad Problems

▶1. Hashing

(a) (2 pts) Consider an open-address hash table with uniform hashing and a load factor $\alpha = 1/2$. What is the expected number of probes in an unsuccessful search? Successful search?

(b) (3 pts) Let the hash function for a table of size m be

$$h(x) = \lfloor Amx \rfloor \bmod m$$

where $A = \hat{\phi} = \frac{\sqrt{5}-1}{2}$. Show that this gives the best possible spread, i.e. if the x are hashed in order, $x + 1$ will be hashed in the largest remaining contiguous interval.

►2. (5 pts) Euler Tour:
Given an **undirected** graph $G = (V, E)$, give an algorithm that finds a cycle in the graph that visits every edge *exactly* once, or says that it can't be done.

►3. (5 pts) Makefiles:
In order to facilitate recompiling programs from multiple source files when only a small number of files have been updated, there is a UNIX utility called 'make' that only recompiles those files that were changed, *and* any intermediate files in the compiliation that depend on those changed. Design an algorithm to recompile only those necessary.

►4. (5 pts) Shortest Airplane Trip:
A person wants to fly from city A to city B in the shortest possible time. S/he turns to the traveling agent who knows all the departure and arrival times of all the flights on the planet. Give an algorithm that will allow the agent to choose an optimal route. Hint: rather than modify Dijkstra's algorithm, modify the data. The time is from departure to arrival at the destination, so it will include layover time (time waiting for a connecting flight).

►5. (9 pts, 3 each) Minimum Spanning Tree changes Suppose you have a graph G and an MST of that graph (i.e. the MST has already been constructed).

(a) Give an algorithm to update the MST when an edge is added to G.

(b) Give an algorithm to update the MST when an edge is deleted from G.

(c) Give an algorithm to update the MST when a vertex (and possibly edges to it) is added to G.

Only 1U Grad Problems

▷1. Nesting Envelopes
You are given an unlimited number of each of n different types of envelopes. The dimensions of envelope type i are $x_i \times y_i$. In nesting envelopes inside one another, you can place envelope A inside envelope B if and only if the dimensions A are *strictly smaller* than the dimensions of B. Design and analyze an algorithm to determine the largest number of envelopes that can be nested inside one another.

Practice Problems

1. The incidence matrix of an undirected graph $G = (V, E)$ is a $|V| \times |E|$ matrix $B = (b_{ij})$ such that
$$b_{ij} = \begin{cases} 1 & (i,j) \in E, \\ 0 & (i,j) \notin E. \end{cases}$$

   (a) Describe what all the entries of the matrix product BB^T represent (B^T is the matrix transpose). Justify.

   (b) Describe what all the entries of the matrix product $B^T B$ represent. Justify.

   ★(c) Let $C = BB^T - 2A$. Let C' be C with the first row and column removed. Show that $\det C'$ is the number of spanning trees. (A is the adjacency matrix of G, with zeroes on the diagonal).

2. $o(V^2)$ Adjacency Matrix Algorithms

   (a) Give an $O(V)$ algorithm to decide whether a directed graph contains a *sink* in an adjacency matrix representation. A sink is a vertex with in-degree $V - 1$.

   (b) An undirected graph is a scorpion if it has a vertex of degree 1 (the sting) connected to a vertex of degree two (the tail) connected to a vertex of degree $V - 2$ (the body) connected to the other $V - 3$ vertices (the feet). Some of the feet may be connected to other feet.

   Design an algorithm that decides whether a given adjacency matrix represents a scorpion by examining only $O(V)$ of the entries.

   (c) Show that it is impossible to decide whether G has at least one edge in $O(V)$ time.

3. Shortest Cycle:
Given an **undirected** graph $G = (V, E)$, and a weight function $f : E \to \mathbf{R}$ on the **edges**, give an algorithm that finds (in time polynomial in V and E) a cycle of smallest weight in G.

4. Longest Simple Path:
Let graph $G = (V, E)$, $|V| = n$. A *simple path* of G, is a path that does not contain the same vertex twice. Use dynamic programming to design an algorithm (not polynomial time) to find a simple path of maximum length in G. Hint: It can be done in $O(n^c 2^n)$ time, for some constant c.

5. Minimum Spanning Tree:
Suppose all edge weights in a graph G are equal. Give an algorithm to compute an MST.

6. Transitive reduction:
Give an algorithm to construct a *transitive reduction* of a directed graph G, i.e. a graph G^{TR} with the fewest edges (but with the same vertices) such that there is a path from a to b in G iff there is also such a path in G^{TR}.

7. (a) What is $5^{2^{29^{5^0}}+23^{4^1}+17^{3^2}+11^{2^3}+5^{1^4}} \bmod 6$?

(b) What is the capital of Nebraska? Hint: It is not Omaha. It is named after a famous president of the United States that was not George Washington. The distance from the Earth to the Moon averages roughly 384,000 km.

CS 373: Combinatorial Algorithms, Spring 1999

`http://www-courses.cs.uiuc.edu/~cs373`

Homework 4 (due Thu. Apr. 1, 1999 by noon)

Name:	
Net ID:	Alias:

Everyone must do the problems marked ►. Problems marked ▷ are for 1-unit grad students and others who want extra credit. (There's no such thing as "partial extra credit"!) Unmarked problems are extra practice problems for your benefit, which will not be graded. Think of them as potential exam questions.

Hard problems are marked with a star; the bigger the star, the harder the problem.

Note: When a question asks you to "give/describe/present an algorithm", you need to do four things to receive full credit:

1. (New!) If not already done, model the problem appropriately. Often the problem is stated in real world terms; give a more rigorous description of the problem. This will help you figure out what is assumed (what you know and what is arbitrary, what operations are and are not allowed), and find the tools needed to solve the problem.
2. Design the most efficient algorithm possible. Significant partial credit will be given for less efficient algorithms, as long as they are still correct, well-presented, and correctly analyzed.
3. Describe your algorithm succinctly, using structured English/pseudocode. We don't want full-fledged compilable source code, but plain English exposition is usually not enough. Follow the examples given in the textbooks, lectures, homeworks, and handouts.
4. Justify the correctness of your algorithm, including termination if that is not obvious.
5. Analyze the time and space complexity of your algorithm.

Undergrad/.75U Grad/1U Grad Problems

►1. (5 pts total) Collinearity
Give an $O(n^2 \log n)$ algorithm to determine whether any three points of a set of n points are collinear. Assume two dimensions and *exact* arithmetic.

►2. (4 pts, 2 each) Convex Hull Recurrence
Consider the following generic recurrence for convex hull algorithms that divide and conquer:

$$T(n,h) = T(n_1,h_1) + T(n_2,h_2) + O(n)$$

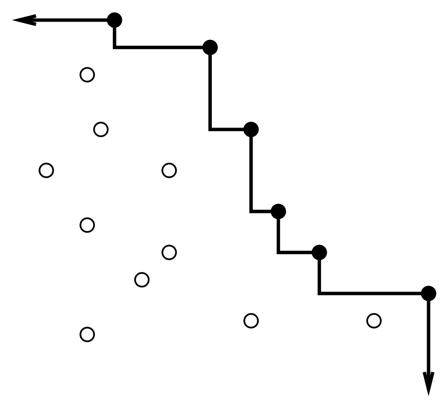
where $n \geq n_1 + n_2, h = h_1 + h_2$ and $n \geq h$. This means that the time to compute the convex hull is a function of both n, the number of input points, and h, the number of convex hull vertices. The splitting and merging parts of the divide-and-conquer algorithm take $O(n)$ time. When n is a constant, $T(n,h)$ is $O(1)$, but when h is a constant, $T(n,h)$ is $O(n)$. Prove that for both of the following restrictions, the solution to the recurrence is $O(n \log h)$:

(a) $h_1, h_2 < \frac{3}{4}h$

(b) $n_1, n_2 < \frac{3}{4}n$

►3. (5 pts) Circle Intersection
Give an $O(n \log n)$ algorithm to test whether any two circles in a set of size n intersect.

►4. (5 pts total) Staircases
You are given a set of points in the first quadrant. A *left-up* point of this set is defined to be a point that has no points both greater than it in both coordinates. The left-up subset of a set of points then forms a *staircase* (see figure).



(a) (3 pts) Give an $O(n \log n)$ algorithm to find the staircase of a set of points.

(b) (2 pts) Assume that points are chosen uniformly at random within a rectangle. What is the average number of points in a staircase? Justify. Hint: you will be able to give an exact answer rather than just asymptotics. You have seen the same analysis before.

Only 1U Grad Problems

▷1. (6 pts, 2 each) Ghostbusters and Ghosts
A group of n ghostbusters is battling n ghosts. Each ghostbuster can shoot a single energy beam at a ghost, eradicating it. A stream goes in a straight line and terminates when it hits a ghost. The ghostbusters must all fire at the same time and no two energy beams may cross. The positions of the ghosts and ghostbusters is fixed in the plane (assume that no three points are collinear).

(a) Prove that for any configuration ghosts and ghostbusters there exists such a non-crossing matching.

(b) Show that there exists a line passing through one ghostbuster and one ghost such that the number of ghostbusters on one side of the line equals the number of ghosts on the same side. Give an efficient algorithm to find such a line.

(c) Give an efficient divide and conquer algorithm to pair ghostbusters and ghosts so that no two streams cross.

Practice Problems

1. Basic Computation (assume two dimensions and *exact* arithmetic)

   (a) Intersection: Extend the basic algorithm to determine if two line segments intersect by taking care of *all* degenerate cases.

   (b) Simplicity: Give an $O(n \log n)$ algorithm to determine whether an n-vertex polygon is simple.

   (c) Area: Give an algorithm to compute the area of a simple n-polygon (not necessarily convex) in $O(n)$ time.

   (d) Inside: Give an algorithm to determine whether a point is within a simple n-polygon (not necessarily convex) in $O(n)$ time.

2. External Diagonals and Mouths

   (a) A pair of polygon vertices defines an *external diagonal* if the line segment between them is completely outside the polygon. Show that every nonconvex polygon has at least one external diagonal.

   (b) Three consective polygon vertices p, q, r form a *mouth* if p and r define an external diagonal. Show that every nonconvex polygon has at least one mouth.

3. On-Line Convex Hull
   We are given the set of points one point at a time. After receiving each point, we must compute the convex hull of all those points so far. Give an algorithm to solve this problem in $O(n^2)$ (We could obviously use Graham's scan n times for an $O(n^2 \log n)$ algorithm). Hint: How do you maintain the convex hull?

4. Another On-Line Convex Hull Algorithm

   (a) Given an n-polygon and a point outside the polygon, give an algorithm to find a tangent.

   *(b) Suppose you have found both tangents. Give an algorithm to remove the points from the polygon that are within the angle formed by the tangents (as segments!) and the opposite side of the polygon.

   (c) Use the above to give an algorithm to compute the convex hull on-line in $O(n \log n)$

★5. Order of the size of the convex hull
   The convex hull on $n \geq 3$ points can have anywhere from 3 to n points. The average case depends on the distribution.

   (a) Prove that if a set of points is chosen randomly within a given rectangle. then the average size of the convex hull is $O(\log n)$.

   (b) Prove that if a set of points is chosen randomly within a given circle. then the average size of the convex hull is $O(\sqrt{n})$.

CS 373: Combinatorial Algorithms, Spring 1999

http://www-courses.cs.uiuc.edu/~cs373

Homework 5 (due Thu. Apr. 22, 1999 by noon)

Name:	
Net ID:	Alias:

Everyone must do the problems marked ▶. Problems marked ▷ are for 1-unit grad students and others who want extra credit. (There's no such thing as "partial extra credit"!) Unmarked problems are extra practice problems for your benefit, which will not be graded. Think of them as potential exam questions.

Hard problems are marked with a star; the bigger the star, the harder the problem.

Note: You will be held accountable for the appropriate responses for answers (e.g. give models, proofs, analyses, etc)

Undergrad/.75U Grad/1U Grad Problems

▶1. (5 pts) Show how to find the occurrences of pattern P in text T by computing the prefix function of the string PT (the concatenation of P and T).

▶2. (10 pts total) Fibonacci strings and KMP matching
Fibonacci strings are defined as follows:

$$F_1 = \text{"b"}, \qquad F_2 = \text{"a"}, \qquad \text{and } F_n = F_{n-1}F_{n-2}, (n > 2)$$

where the recursive rule uses concatenation of strings, so F_2 is "ab", F_3 is "aba". Note that the length of F_n is the nth Fibonacci number.

(a) (2 pts) Prove that in any Fibonacci string there are no two b's adjacent and no three a's.

(b) (2 pts) Give the unoptimized and optimized 'prefix' (fail) function for F_7.

(c) (3 pts) Prove that, in searching for a Fibonacci string of length m using unoptimized KMP, it may shift up to $\lceil \log_\phi m \rceil$ times, where $\phi = (1 + \sqrt{5})/2$, is the golden ratio. (Hint: Another way of saying the above is that we are given the string F_n and we may have to shift n times. Find an example text T that gives this number of shifts).

(d) (3 pts) What happens here when you use the optimized prefix function? Explain.

▶3. (5 pts) Prove that finding the second smallest of n elements takes $n + \lceil \lg n \rceil - 2$ comparisons in the worst case. Prove for both upper and lower bounds. Hint: find the (first) smallest using an elimination tournament.

▶4. (4 pts, 2 each) Lower Bounds on Adjacency Matrix Representations of Graphs

(a) Prove that the time to determine if an undirected graph has a cycle is $\Omega(V^2)$.

(b) Prove that the time to determine if there is a path between two nodes in an undirected graph is $\Omega(V^2)$.

Only 1U Grad Problems

▷1. (5 pts) Prove that $\lceil 3n/2 \rceil - 2$ comparisons are necessary in the worst case to find both the minimum and maximum of n numbers. Hint: Consider how many are potentially either the min or max.

Practice Problems

1. String matching with wild-cards
   Suppose you have an alphabet for patterns that includes a 'gap' or wild-card character; any length string of any characters can match this additional character. For example if '$*$' is the wild-card, then the pattern 'foo*bar*nad' can be found in 'foofoowangbarnad'. Modify the computation of the prefix function to correctly match strings using KMP.

2. Prove that there is no comparison sort whose running time is linear for at least 1/2 of the $n!$ inputs of length n. What about at least $1/n$? What about at least $1/2^n$?.

3. Prove that $2n - 1$ comparisons are necessary in the worst case to merge two sorted lists containing n elements each.

4. Find asymptotic upper and lower bounds to $\lg(n!)$ without Stirling's approximation (Hint: use integration).

5. Given a sequence of n elements of n/k blocks (k elements per block) all elements in a block are less than those to the right in sequence, show that you cannot have the whole sequence sorted in better than $\Omega(n \lg k)$. Note that the entire sequence would be sorted if each of the n/k blocks were individually sorted in place. Also note that combining the lower bounds for each block is not adequate (that only gives an upper bound).

6. Some elementary reductions

   (a) Prove that if you can decide whether a graph G has a clique of size k (or less) then you can decide whether a graph G' has an independent set of size k (or more).

   (b) Prove that if you can decide whether one graph G_1 is a subgraph of another graph G_2 then you can decide whether a graph G has a clique of size k (or less).

7. There is no Proof but We are pretty Sure
   Justify (prove) the following logical rules of inference:

   (a) Classical - If $a \rightarrow b$ and a hold, then b holds.

   (b) Fuzzy - Prove: If $a \rightarrow b$ holds, and a holds with probability p, then b holds with probability less than p. Assume all probabilities are independent.

   (c) Give formulas for computing the probabilities of the fuzzy logical operators 'and', 'or', 'not', and 'implies', and fill out truth tables with the values T (true, $p = 1$), L (likely, $p = 0.9$), M (maybe, $p = 0.5$), N (not likely, $p = 0.1$), and F (false, $p = 0$).

(d) If you have a poly time (algorithmic) reduction from problem B to problem A (i.e. you can solve B using A with a poly time conversion), and it is very unlikely that A has better than lower bound $\Omega(2^n)$ algorithm, what can you say about problem A. Hint: a solution to A implies a solution to B.

CS 373: Combinatorial Algorithms, Spring 1999

Midterm 1 (February 23, 1999)

Name:	
Net ID:	Alias:

This is a closed-book, closed-notes exam!

If you brought anything with you besides writing instruments and your $8\frac{1}{2}'' \times 11''$ cheat sheet, please leave it at the front of the classroom.

- **Don't panic!**
- Print your name, netid, and alias in the boxes above, and print your name at the top of every page.
- **Answer four of the five questions on the exam.** Each question is worth 10 points. If you answer more than four questions, the one with the lowest score will be ignored. **1-unit graduate students must answer question #5.**
- Please write your answers on the front of the exam pages. Use the backs of the pages as scratch paper. Let us know if you need more paper.
- Read the entire exam before writing anything. Make sure you understand what the questions are asking. If you give a beautiful answer to the wrong question, you'll get no credit. If any question is unclear, please ask one of us for clarification.
- Don't spend too much time on any single problem. If you get stuck, move on to something else and come back later.
- Write *something* down for every problem. Don't panic and erase large chunks of work. Even if you think it's nonsense, it might be worth partial credit.
- **Don't panic!**

#	Score	Grader
1		
2		
3		
4		
5		

1. **Multiple Choice**

   Every question below has one of the following answers.

   (a) $\Theta(1)$ (b) $\Theta(\log n)$ (c) $\Theta(n)$ (d) $\Theta(n \log n)$ (e) $\Theta(n^2)$

   For each question, write the letter that corresponds to your answer. You do not need to justify your answer. Each correct answer earns you 1 point, but each incorrect answer *costs* you $\frac{1}{2}$ point. (You cannot score below zero.)

☐ What is $\sum_{i=1}^{n} i$?

☐ What is $\sum_{i=1}^{n} \frac{1}{i}$?

☐ What is the solution of the recurrence $T(n) = T(\sqrt{n}) + n$?

☐ What is the solution of the recurrence $T(n) = T(n-1) + \lg n$?

☐ What is the solution of the recurrence $T(n) = 2T\left(\left\lceil \frac{n+27}{2} \right\rceil\right) + 5n - 7\sqrt{\lg n} + \frac{1999}{n}$?

☐ The amortized time for inserting one item into an n-node splay tree is $O(\log n)$. What is the worst-case time for a sequence of n insertions into an initially empty splay tree?

☐ The expected time for inserting one item into an n-node randomized treap is $O(\log n)$. What is the worst-case time for a sequence of n insertions into an initially empty treap?

☐ What is the worst-case running time of randomized quicksort?

☐ How many bits are there in the binary representation of the nth Fibonacci number?

☐ What is the worst-case cost of merging two arbitrary splay trees with n items total into a single splay tree with n items.

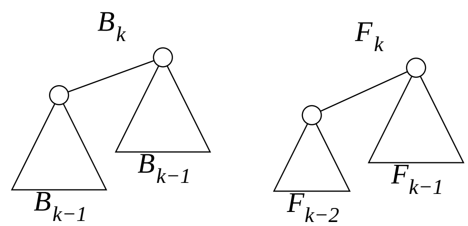
☐ Suppose you correctly identify three of the answers to this question as obviously wrong. If you pick one of the two remaining answers at random, what is your expected score for this problem?

2. (a) **[5 pt]** Recall that a binomial tree of order k, denoted B_k, is defined recursively as follows. B_0 is a single node. For any $k > 0$, B_k consists of two copies of B_{k-1} linked together.

   Prove that the degree of any node in a binomial tree is equal to its height.

   (b) **[5 pt]** Recall that a Fibonacci tree of order k, denoted F_k, is defined recursively as follows. F_1 and F_2 are both single nodes. For any $k > 2$, F_k consists of an F_{k-2} linked to an F_{k-1}.

   Prove that for any node v in a Fibonacci tree, $\mathrm{height}(v) = \lceil \mathrm{degree}(v)/2 \rceil$.



Recursive definitions of binomial trees and Fibonacci trees.

3. Consider the following randomized algorithm for computing the smallest element in an array.

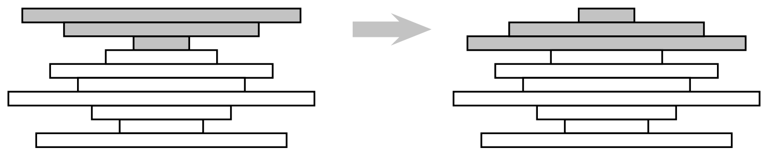
```
RandomMin(A[1..n]):
  min ← ∞
  for i ← 1 to n in random order
      if A[i] < min
          min ← A[i]   (⋆)
      return min
```

(a) **[1 pt]** In the worst case, how many times does RandomMin execute line $(\star)$?

(b) **[3 pt]** What is the probability that line $(\star)$ is executed during the nth iteration of the for loop?

(c) **[6 pt]** What is the exact expected number of executions of line $(\star)$? (A correct $\Theta()$ bound is worth 4 points.)

4. Suppose we have a stack of n pancakes of different sizes. We want to sort the pancakes so that smaller pancakes are on top of larger pancakes. The only operation we can perform is a *flip* — insert a spatula under the top k pancakes, for some k between 1 and n, and flip them all over.



Flipping the top three pancakes

   (a) **[3 pt]** Describe an algorithm to sort an arbitrary stack of n pancakes.

   (b) **[3 pt]** Prove that your algorithm is correct.

   (c) **[2 pt]** Exactly how many flips does your algorithm perform in the worst case? (A correct $\Theta()$ bound is worth one point.)

   (d) **[2 pt]** Suppose one side of each pancake is burned. Exactly how many flips do you need to sort the pancakes *and* have the burned side of every pancake on the bottom? (A correct $\Theta()$ bound is worth one point.)

5. You are given an array $A[1\,..\,n]$ of integers. Describe and analyze an algorithm that finds the largest sum of of elements in a contiguous subarray $A[i\,..\,j]$.

   For example, if the array contains the numbers $(-6, 12, -7, 0, 14, -7, 5)$, then the largest sum is $19 = 12 - 7 + 0 + 14$.

   $$\boxed{-6}\underbrace{\boxed{12}\boxed{-7}\boxed{0}\boxed{14}}_{19}\boxed{-7}\boxed{5}$$

   To get full credit, your algorithm must run in $\Theta(n)$ time — there are at least three different ways to do this. An algorithm that runs in $\Theta(n^2)$ time is worth 7 points.

CS 373: Combinatorial Algorithms, Spring 1999

Midterm 2 (April 6, 1999)

Name:	
Net ID:	Alias:

This is a closed-book, closed-notes exam!

If you brought anything with you besides writing instruments and your $8\frac{1}{2}'' \times 11''$ cheat sheet, please leave it at the front of the classroom.

- **Don't panic!**
- Print your name, netid, and alias in the boxes above, and print your name at the top of every page.
- **Answer four of the five questions on the exam.** Each question is worth 10 points. If you answer more than four questions, the one with the lowest score will be ignored. **1-unit graduate students must answer question #5.**
- Please write your answers on the front of the exam pages. You can use the backs of the pages as scratch paper. Let us know if you need more paper.
- Read the entire exam before writing anything. Make sure you understand what the questions are asking. If you give a beautiful answer to the wrong question, you'll get no credit. If any question is unclear, please ask one of us for clarification.
- Don't spend too much time on any single problem. If you get stuck, move on to something else and come back later.
- Write *something* down for every problem. Don't panic and erase large chunks of work. Even if you think it's nonsense, it might be worth partial credit.
- **Don't panic!**

#	Score	Grader
1		
2		
3		
4		
5		

1. **Bipartite Graphs**

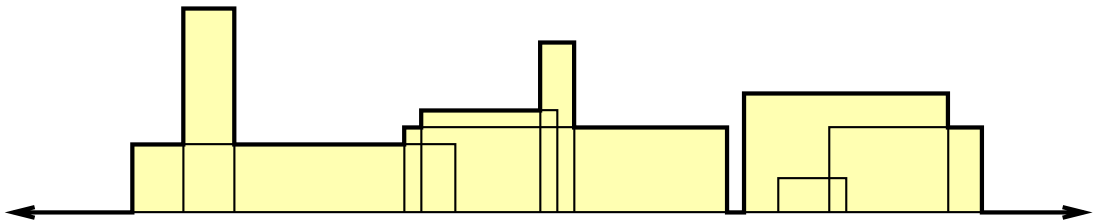
   A graph (V, E) is *bipartite* if the vertices V can be partitioned into two subsets L and R, such that every edge has one vertex in L and the other in R.

   (a) Prove that every tree is a bipartite graph.

   (b) Describe and analyze an efficient algorithm that determines whether a given connected, undirected graph is bipartite.

2. **Manhattan Skyline**

   The purpose of the following problem is to compute the outline of a projection of rectangular buildings. You are given the height, width, and left x-coordinate of n rectangles. The bottom of each rectangle is on the x-axis. Describe and analyze an efficient algorithm to compute the vertices of the "skyline".



A set of rectangles and its skyline.

3. **Least Cost Vertex Weighted Path**

   Suppose you want to drive from Champaign to Los Angeles via a network of roads connecting cities. You don't care how long it takes, how many cities you visit, or how much gas you use. All you care about is how much money you spend on food. Each city has a possibly different, but fixed, value for food.

   More formally, you are given a directed graph $G = (V, E)$ with nonnegative weights on the *vertices* $w\colon V \to \mathbb{R}^+$, a source vertex $s \in V$, and a target vertex $t \in V$. Describe and analyze an efficient algorithm to find a minimum-weight path from s to t. [Hint: Modify the graph.]

4. **Union-Find with Alternate Rule**

   In the UNION-FIND data structure described in CLR and in class, each set is represented by a rooted tree. The UNION algorithm, given two sets, decides which set is to be the parent of the other by comparing their ranks, where the rank of a set is an upper bound on the height of its tree.

   Instead of rank, we propose using the *weight* of the set, which is just the number of nodes in the set. Here's the modified UNION algorithm:

```
UNION(A, B):
  if weight(A) > weight(B)
      parent(B) ← A
      weight(A) ← weight(A) + weight(B)
  else
      parent(A) ← B
      weight(B) ← weight(A) + weight(B)
```

   Prove that if we use this method, then after any sequence of n MAKESETs, UNIONs, and FINDs (with path compression), the *height* of the tree representing any set is $O(\log n)$.

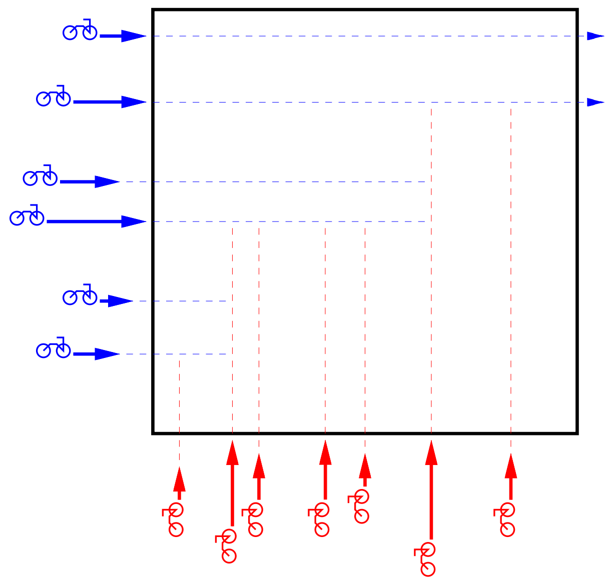
   [Hint: First prove it without path compression, and then argue that path compression doesn't matter (for this problem).]

5. **Motorcycle Collision**

   One gang, Hell's Ordinates, start west of the arena facing directly east; the other, The Vicious Abscissas of Death, start south of the arena facing due north. All the motorcycles start moving simultaneously at a prearranged signal. Each motorcycle moves at a fixed speed—no speeding up, slowing down, or turning is allowed. Each motorcycle leaves an oil trail behind it. If another motorcycle crosses that trail, it falls over and stops leaving a trail.

   More formally, we are given two sets H and V, each containing n motorcycles. Each motorcycle is represented by three numbers (s, x, y): its speed and the x- and y-coordinates of its initial location. Bikes in H move horizontally; bikes in V move vertically.

   Assume that the bikes are infinitely small points, that the bike trails are infinitely thin lie segments, that a bike crashes stops *exactly* when it hits a oil trail, and that no two bikes collide with each other.



Two sets of motorcycles and the oil trails they leave behind.

   (a) Solve the case $n = 1$. Given only two motorcycles moving perpendicular to each other, determine which one of them falls over and where in $O(1)$ time.

   (b) Describe an efficient algorithm to find the set of all points where motorcycles fall over.

5. **Motorcycle Collision (continued)**

   Incidentally, the movie *Tron* is being shown during Roger Ebert's Forgotten Film Festival at the Virginia Theater in Champaign on April 25. Get your tickets now!

CS 373: Combinatorial Algorithms, Spring 1999

Final Exam (May 7, 1999)

Name:	
Net ID:	Alias:

This is a closed-book, closed-notes exam!

If you brought anything with you besides writing instruments and your two $8\frac{1}{2}'' \times 11''$ cheat sheets, please leave it at the front of the classroom.

- Print your name, netid, and alias in the boxes above, and print your name at the top of every page.
- **Answer six of the seven questions on the exam.** Each question is worth 10 points. If you answer every question, the one with the lowest score will be ignored. **1-unit graduate students must answer question #7.**
- Please write your answers on the front of the exam pages. Use the backs of the pages as scratch paper. Let us know if you need more paper.
- Read the entire exam before writing anything. Make sure you understand what the questions are asking. If you give a beautiful answer to the wrong question, you'll get no credit. If any question is unclear, please ask one of us for clarification.
- Don't spend too much time on any single problem. If you get stuck, move on to something else and come back later.
- Write *something* down for every problem. Don't panic and erase large chunks of work. Even if you think it's nonsense, it might be worth partial credit.

#	Score	Grader
1		
2		
3		
4		
5		
6		
7		

1. **Short Answer**

sorting	induction	Master theorem	divide and conquer
randomized algorithm	amortization	brute force	hashing
binary search	depth-first search	splay tree	Fibonacci heap
convex hull	sweep line	minimum spanning tree	shortest paths
shortest path	adversary argument	NP-hard	reduction
string matching	evasive graph property	dynamic programming	H_n

Choose from the list above the best method for solving each of the following problems. We do *not* want complete solutions, just a short description of the proper solution technique! Each item is worth 1 point.

(a) Given a Champaign phone book, find your own phone number.

(b) Given a collection of n rectangles in the plane, determine whether any two intersect in $O(n \log n)$ time.

(c) Given an undirected graph G and an integer k, determine if G has a complete subgraph with k edges.

(d) Given an undirected graph G, determine if G has a triangle — a complete subgraph with three vertices.

(e) Prove that any n-vertex graph with minimum degree at least $n/2$ has a Hamiltonian cycle.

(f) Given a graph G and three distinguished vertices u, v, and w, determine whether G contains a path from u to v that passes through w.

(g) Given a graph G and two distinguished vertices u and v, determine whether G contains a path from u to v that passes through at most 17 edges.

(h) Solve the recurrence $T(n) = 5T(n/17) + O(n^{4/3})$.

(i) Solve the recurrence $T(n) = 1/n + T(n-1)$, where $T(0) = 0$.

(j) Given an array of n integers, find the integer that appears most frequently in the array.

(a) ______________________ (f) ______________________

(b) ______________________ (g) ______________________

(c) ______________________ (h) ______________________

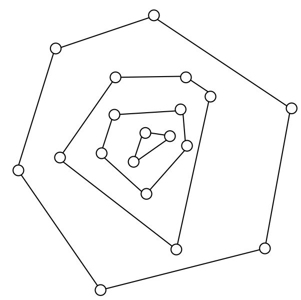
(d) ______________________ (i) ______________________

(e) ______________________ (j) ______________________

2. **Convex Layers**

   Given a set Q of points in the plane, define the *convex layers* of Q inductively as follows: The first convex layer of Q is just the convex hull of Q. For all $i > 1$, the ith convex layer is the convex hull of Q after the vertices of the first $i - 1$ layers have been removed.

   Give an $O(n^2)$-time algorithm to find all convex layers of a given set of n points. [Partial credit for a correct slower algorithm; extra credit for a correct faster algorithm.]



A set of points with four convex layers.

3. Suppose you are given an array of n numbers, sorted in increasing order.

   (a) **[3 pts]** Describe an $O(n)$-time algorithm for the following problem:

   Find two numbers from the list that add up to zero, or report that there is no such pair. In other words, find two numbers a and b such that $a + b = 0$.

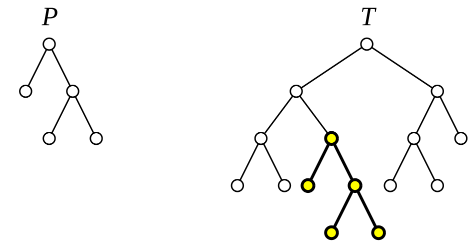
   (b) **[7 pts]** Describe an $O(n^2)$-time algorithm for the following problem:

   Find *three* numbers from the list that add up to zero, or report that there is no such triple. In other words, find three numbers a, b, and c, such that $a + b + c = 0$. [Hint: Use something similar to part (a) as a subroutine.]

4. **Pattern Matching**

   (a) **[4 pts]** A *cyclic rotation* of a string is obtained by chopping off a prefix and gluing it at the end of the string. For example, ALGORITHM is a cyclic shift of RITHMALGO. Describe and analyze an algorithm that determines whether one string $P[1..m]$ is a cyclic rotation of another string $T[1..n]$.

   (b) **[6 pts]** Describe and analyze an algorithm that decides, given any two binary trees P and T, whether P equals a subtree of T. [Hint: First transform both trees into strings.]



P occurs exactly once as a subtree of T.

5. **Two-stage Sorting**

   (a) **[1 pt]** Suppose we are given an array $A[1..n]$ of distinct integers. Describe an algorithm that splits A into n/k subarrays, each with k elements, such that the elements of each subarray $A[(i-1)k+1..ik]$ are sorted. Your algorithm should run in $O(n \log k)$ time.

   (b) **[2 pts]** Given an array $A[1..n]$ that is already split into n/k sorted subarrays as in part (a), describe an algorithm that sorts the entire array in $O(n \log(n/k))$ time.

   (c) **[3 pts]** Prove that your algorithm from part (a) is optimal.

   (d) **[4 pts]** Prove that your algorithm from part (b) is optimal.

4	14	7	3	1	20	11	9	5	13	12	19	10	16	17	2	8	6	18	15

$$\Downarrow (a)$$

1	4	3	7	14 ‖	5	9	11	13	20 ‖	10	12	16	17	19 ‖	2	6	8	15	18

$$\Downarrow (b)$$

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20

6. **SAT Reduction**

   Suppose you are have a black box that magically solves SAT (the formula satisfiability problem) in constant time. That is, given a boolean formula of variables and logical operators ($\wedge, \vee, \neg$), the black box tells you, in constant time, whether or not the formula can be satisfied. Using this black box, design and analyze a **polynomial-time** algorithm that computes an assignment to the variables that satisfies the formula.

7. **Knapsack**

   You're hiking through the woods when you come upon a treasure chest filled with objects. Each object has a different size, and each object has a price tag on it, giving its value. There is no correlation between an object's size and its value. You want to take back as valuable a subset of the objects as possible (in one trip), but also making sure that you will be able to carry it in your knapsack which has a limited size.

   In other words, you have an integer capacity K and a target value V, and you want to decide whether there is a subset of the objects whose total size is *at most* K and whose total value is *at least* V.

   (a) **[5 pts]** Show that this problem is NP-hard. [Hint: Restate the problem more formally, then reduce from the NP-hard problem PARTITION: Given a set S of nonnegative integers, is there a partition of S into disjoint subsets A and B (where $A \cup B = S$) whose sums are equal, *i.e.*, $\sum_{a \in A} a = \sum_{b \in B} b$.]

   (b) **[5 pts]** Describe and analyze a dynamic programming algorithm to solve the knapsack problem in $O(nK)$ time. Prove your algorithm is correct.

CS 373: Combinatorial Algorithms, Fall 2000

Homework 0, due August 31, 2000 at the beginning of class

Name:	
Net ID:	Alias:

Neatly print your name (first name first, with no comma), your network ID, and a short alias into the boxes above. **Do not *sign* your name. Do *not* write your Social Security number.** Staple this sheet of paper to the top of your homework.

Grades will be listed on the course web site by alias give us, so your alias should not resemble your name or your Net ID. If you don't give yourself an alias, we'll give you one that you won't like.

Before you do anything else, read the Homework Instructions and FAQ on the CS 373 course web page (http://www-courses.cs.uiuc.edu/~cs373/hw/faq.html), and then check the box below. This web page gives instructions on how to write and submit homeworks—staple your solutions together in order, write your name and netID on every page, don't turn in source code, analyze everything, use good English and good logic, and so forth.

☐ I have read the CS 373 Homework Instructions and FAQ.

This homework tests your familiarity with the prerequisite material from CS 173, CS 225, and CS 273—many of these problems have appeared on homeworks or exams in those classes—primarily to help you identify gaps in your knowledge. **You are responsible for filling those gaps on your own.** Parberry and Chapters 1–6 of CLR should be sufficient review, but you may want to consult other texts as well.

Required Problems

1. Sort the following 25 functions from asymptotically smallest to asymptotically largest, indicating ties if there are any:

$$\begin{array}{ccccc}
1 & n & n^2 & \lg n & \lg(n \lg n) \\
\lg^* n & \lg^* 2^n & 2^{\lg^* n} & \lg \lg^* n & \lg^* \lg n \\
n^{\lg n} & (\lg n)^n & (\lg n)^{\lg n} & n^{1/\lg n} & n^{\lg \lg n} \\
\log_{1000} n & \lg^{1000} n & \lg^{(1000)} n & \left(1 + \frac{1}{n}\right)^n & n^{1/1000}
\end{array}$$

To simplify notation, write $f(n) \ll g(n)$ to mean $f(n) = o(g(n))$ and $f(n) \equiv g(n)$ to mean $f(n) = \Theta(g(n))$. For example, the functions n^2, n, $\binom{n}{2}$, n^3 could be sorted either as $n \ll n^2 \equiv \binom{n}{2} \ll n^3$ or as $n \ll \binom{n}{2} \equiv n^2 \ll n^3$.

2. (a) Prove that any positive integer can be written as the sum of distinct powers of 2. For example: $42 = 2^5 + 2^3 + 2^1$, $25 = 2^4 + 2^3 + 2^0$, $17 = 2^4 + 2^0$. [Hint: "Write the number in binary" is *not* a proof; it just restates the problem.]

   (b) Prove that any positive integer can be written as the sum of distinct *nonconsecutive* Fibonacci numbers—if F_n appears in the sum, then neither F_{n+1} nor F_{n-1} will. For example: $42 = F_9 + F_6$, $25 = F_8 + F_4 + F_2$, $17 = F_7 + F_4 + F_2$.

   (c) Prove that *any* integer (positive, negative, or zero) can be written in the form $\sum_i \pm 3^i$, where the exponents i are distinct non-negative integers. For example: $42 = 3^4 - 3^3 - 3^2 - 3^1$, $25 = 3^3 - 3^1 + 3^0$, $17 = 3^3 - 3^2 - 3^0$.

3. Solve the following recurrences. State tight asymptotic bounds for each function in the form $\Theta(f(n))$ for some recognizable function $f(n)$. You do not need to turn in proofs (in fact, please *don't* turn in proofs), but you should do them anyway just for practice. If no base cases are given, assume something reasonable but nontrivial. Extra credit will be given for more exact solutions.

   (a) $A(n) = 3A(n/2) + n$

   (b) $B(n) = \max\limits_{n/3<k<2n/3} \big(B(k) + B(n-k) + n\big)$

   (c) $C(n) = 4C(\lfloor n/2 \rfloor + 5) + n^2$

   *(d) $D(n) = 2D(n/2) + n/\lg n$

   *(e) $E(n) = \dfrac{E(n-1)}{E(n-2)}$, where $E(1) = 1$ and $E(2) = 2$.

4. Penn and Teller have a special deck of fifty-two cards, with no face cards and nothing but clubs—the ace, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, $\ldots$, 52 of clubs. (They're big cards.) Penn shuffles the deck until each each of the $52!$ possible orderings of the cards is equally likely. He then takes cards one at a time from the top of the deck and gives them to Teller, stopping as soon as he gives Teller the three of clubs.

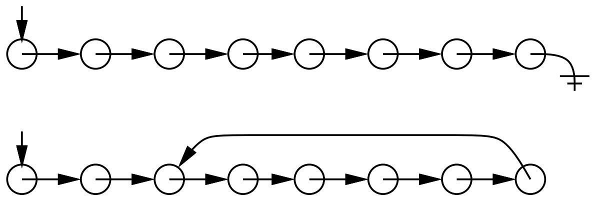
   (a) On average, how many cards does Penn give Teller?

   (b) On average, what is the smallest-numbered card that Penn gives Teller?

   *(c) On average, what is the largest-numbered card that Penn gives Teller?

   [Hint: Solve for an n-card deck, and then set $n = 52$.] Prove that your answers are correct. If you have to appeal to "intuition" or "common sense", your answers are probably wrong!

5. Suppose you have a pointer to the head of singly linked list. Normally, each node in the list only has a pointer to the next element, and the last node's pointer is NULL. Unfortunately, your list might have been corrupted by a bug in somebody else's code[1], so that the last node has a pointer back to some other node in the list instead.

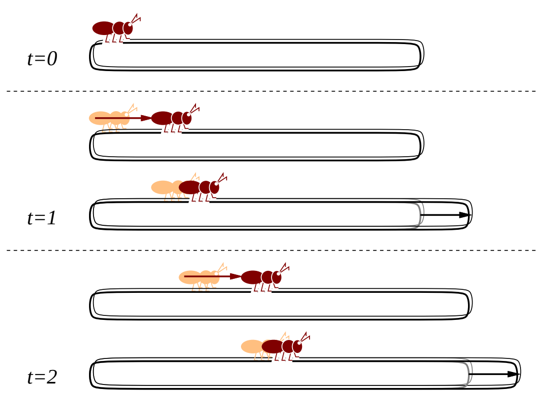


Top: A standard singly-linked list. Bottom: A corrupted singly linked list.

Describe an algorithm[2] that determines whether the linked list is corrupted or not. Your algorithm must not modify the list. For full credit, your algorithm should run in $O(n)$ time, where n is the number of nodes in the list, and use $O(1)$ extra space (not counting the list itself).

6. *[This problem is required only for graduate students taking CS 373 for a full unit; anyone else can submit a solution for extra credit.]*

An ant is walking along a rubber band, starting at the left end. Once every second, the ant walks one inch to the right, and then you make the rubber band one inch longer by pulling on the right end. The rubber band stretches uniformly, so stretching the rubber band also pulls the ant to the right. The initial length of the rubber band is n inches, so after t seconds, the rubber band is $n + t$ inches long.



Every second, the ant walks an inch, and then the rubber band is stretched an inch longer.

(a) How far has the ant moved after t seconds, as a function of n and t? Set up a recurrence and (for full credit) give an *exact* closed-form solution. [Hint: What *fraction* of the rubber band's length has the ant walked?]

*(b) How long does it take the ant to get to the right end of the rubber band? For full credit, give an answer of the form $f(n) + \Theta(1)$ for some explicit function $f(n)$.

[1] After all, *your* code is always completely 100% bug-free. Isn't that right, Mr. Gates?

[2] Since you've read the Homework Instructions, you know what the phrase "describe an algorithm" means. Right?

Practice Problems

These remaining practice problems are entirely for your benefit. Don't turn in solutions—we'll just throw them out—but feel free to ask us about these questions during office hours and review sessions. Think of these as potential exam questions (hint, hint).

1. Recall the standard recursive definition of the Fibonacci numbers: $F_0 = 0$, $F_1 = 1$, and $F_n = F_{n-1} + F_{n-2}$ for all $n \geq 2$. Prove the following identities for all positive integers n and m.

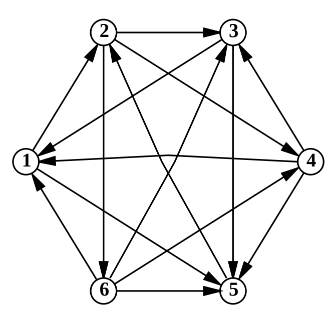
   (a) F_n is even if and only if n is divisible by 3.

   (b) $\sum_{i=0}^{n} F_i = F_{n+2} - 1$

   (c) $F_n^2 - F_{n+1}F_{n-1} = (-1)^{n+1}$

   ★(d) If n is an integer multiple of m, then F_n is an integer multiple of F_m.

2. A *tournament* is a directed graph with exactly one edge between every pair of vertices. (Think of the nodes as players in a round-robin tournament, where each edge points from the winner to the loser.) A *Hamiltonian path* is a sequence of directed edges, joined end to end, that visits every vertex exactly once. Prove that every tournament contains at least one Hamiltonian path.



A six-vertex tournament containing the Hamiltonian path $6 \to 4 \to 5 \to 2 \to 3 \to 1$.

3. (a) Prove the following identity by induction:

$$\binom{2n}{n} = \sum_{k=0}^{n} \binom{n}{k}\binom{n}{n-k}.$$

   (b) Give a non-inductive combinatorial proof of the same identity, by showing that the two sides of the equation count exactly the same thing in two different ways. There is a correct one-sentence proof.

4. (a) Prove that $2^{\lceil \lg n \rceil + \lfloor \lg n \rfloor}/n = \Theta(n)$.

   (b) Is $2^{\lfloor \lg n \rfloor} = \Theta\big(2^{\lceil \lg n \rceil}\big)$? Justify your answer.

   (c) Is $2^{2^{\lfloor \lg \lg n \rfloor}} = \Theta\big(2^{2^{\lceil \lg \lg n \rceil}}\big)$? Justify your answer.

   (d) Prove that if $f(n) = O(g(n))$, then $2^{f(n)} = O(2^{g(n)})$. Justify your answer.

   (e) Prove that $f(n) = O(g(n))$ does *not* imply that $\log(f(n)) = O(\log(g(n)))$?.

   *(f) Prove that $\log^k n = o(n^{1/k})$ for any positive integer k.

5. Solve the following recurrences. State tight asymptotic bounds for each function in the form $\Theta(f(n))$ for some recognizable function $f(n)$. You do not need to turn in proofs (in fact, please *don't* turn in proofs), but you should do them anyway just for practice. If no base cases are given, assume something reasonable (but nontrivial). Extra credit will be given for more exact solutions.

   (a) $A(n) = A(n/2) + n$

   (b) $B(n) = 2B(n/2) + n$

   (c) $C(n) = \min\limits_{0<k<n} \big(C(k) + C(n-k) + 1\big)$, where $C(1) = 1$.

   (d) $D(n) = D(n-1) + 1/n$

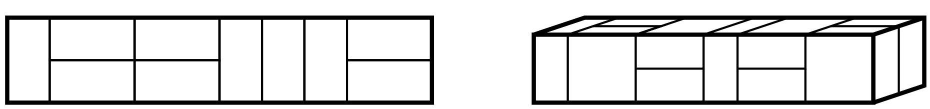
   *(e) $E(n) = 8E(n-1) - 15E(n-2) + 1$

   *(f) $F(n) = (n-1)(F(n-1) + F(n-2))$, where $F(0) = F(1) = 1$

   $^\bigstar$(g) $G(n) = G(n/2) + G(n/4) + G(n/6) + G(n/12) + n$ [Hint: $\frac{1}{2} + \frac{1}{4} + \frac{1}{6} + \frac{1}{12} = 1$.]

6. (a) A *domino* is a 2×1 or 1×2 rectangle. How many different ways are there to completely fill a $2 \times n$ rectangle with n dominos? Set up a recurrence relation and give an *exact* closed-form solution.

   (b) A *slab* is a three-dimensional box with dimensions $1 \times 2 \times 2$, $2 \times 1 \times 2$, or $2 \times 2 \times 1$. How many different ways are there to fill a $2 \times 2 \times n$ box with n slabs? Set up a recurrence relation and give an *exact* closed-form solution.



A 2×10 rectangle filled with ten dominos, and a $2 \times 2 \times 10$ box filled with ten slabs.

7. Professor George O'Jungle has a favorite 26-node binary tree, whose nodes are labeled by letters of the alphabet. The preorder and postorder sequences of nodes are as follows:

   preorder: `M N H C R S K W T G D X I Y A J P O E Z V B U L Q F`
   postorder: `C W T K S G R H D N A O E P J Y Z I B Q L F U V X M`

   Draw Professor O'Jungle's binary tree, and give the inorder sequence of nodes.

8. Alice and Bob each have a fair n-sided die. Alice rolls her die once. Bob then repeatedly throws his die until he rolls a number at least as big as the number Alice rolled. Each time Bob rolls, he pays Alice $\$1$. (For example, if Alice rolls a 5, and Bob rolls a 4, then a 3, then a 1, then a 5, the game ends and Alice gets $\$4$. If Alice rolls a 1, then no matter what Bob rolls, the game will end immediately, and Alice will get $\$1$.)

   Exactly how much money does Alice expect to win at this game? Prove that your answer is correct. If you have to appeal to "intuition" or "common sense", your answer is probably wrong!

9. Prove that for any nonnegative parameters a and b, the following algorithms terminate and produce identical output.

```
SlowEuclid(a, b):
  if b > a
      return SlowEuclid(b, a)
  else if b = 0
      return a
  else
      return SlowEuclid(b, a − b)
```

```
FastEuclid(a, b):
  if b = 0
      return a
  else
      return FastEuclid(b, a mod b)
```

CS 373: Combinatorial Algorithms, Fall 2000
Homework 1 (due September 12, 2000 at midnight)

Name:		
Net ID:	Alias:	U $^3/_4$ 1

Name:		
Net ID:	Alias:	U $^3/_4$ 1

Name:		
Net ID:	Alias:	U $^3/_4$ 1

Starting with Homework 1, homeworks may be done in teams of up to three people. Each team turns in just one solution, and every member of a team gets the same grade. Since 1-unit graduate students are required to solve problems that are worth extra credit for other students, **1-unit grad students may not be on the same team as 3/4-unit grad students or undergraduates.**

Neatly print your name(s), NetID(s), and the alias(es) you used for Homework 0 in the boxes above. Please also tell us whether you are an undergraduate, 3/4-unit grad student, or 1-unit grad student by circling U, $^3/_4$, or 1, respectively. Staple this sheet to the top of your homework.

Required Problems

1. Suppose we want to display a paragraph of text on a computer screen. The text consists of n words, where the ith word is p_i pixels wide. We want to break the paragraph into several lines, each exactly P pixels long. Depending on which words we put on each line, we will need to insert different amounts of white space between the words. The paragraph should be fully justified, meaning that the first word on each line starts at its leftmost pixel, and *except for the last line*, the last character on each line ends at its rightmost pixel. There must be at least one pixel of whitespace between any two words on the same line.

   Define the *slop* of a paragraph layout as the sum over all lines, *except the last*, of the cube of the number of extra white-space pixels in each line (not counting the one pixel required between every adjacent pair of words). Specifically, if a line contains words i through j, then the amount of extra white space on that line is $P - j + i - \sum_{k=i}^{j} p_k$. Describe a dynamic programming algorithm to print the paragraph with minimum slop.

2. Consider the following sorting algorithm:

```
STUPIDSORT(A[0 .. n − 1]) :
  if n = 2 and A[0] > A[1]
      swap A[0] ↔ A[1]
  else if n > 2
      m ← ⌈2n/3⌉
      STUPIDSORT(A[0 .. m − 1])
      STUPIDSORT(A[n − m .. n − 1])
      STUPIDSORT(A[0 .. m − 1])
```

   (a) Prove that STUPIDSORT actually sorts its input.

   (b) Would the algorithm still sort correctly if we replaced the line $m \leftarrow \lceil 2n/3 \rceil$ with $m \leftarrow \lfloor 2n/3 \rfloor$? Justify your answer.

   (c) State a recurrence (including the base case(s)) for the number of comparisons executed by STUPIDSORT.

   (d) Solve the recurrence, and prove that your solution is correct. [Hint: Ignore the ceiling.] Does the algorithm deserve its name?

   *(e) Show that the number of *swaps* executed by STUPIDSORT is at most $\binom{n}{2}$.

3. The following randomized algorithm selects the rth smallest element in an unsorted array $A[1 .. n]$. For example, to find the smallest element, you would call RANDOMSELECT$(A, 1)$; to find the median element, you would call RANDOMSELECT$(A, \lfloor n/2 \rfloor)$. Recall from lecture that PARTITION splits the array into three parts by comparing the pivot element $A[p]$ to every other element of the array, using $n - 1$ comparisons altogether, and returns the new index of the pivot element.

```
RANDOMSELECT(A[1 .. n], r) :
  p ← RANDOM(1, p)
  k ← PARTITION(A[1 .. n], p)

  if r < k
      return RANDOMSELECT(A[1 .. k − 1], r)
  else if r > k
      return RANDOMSELECT(A[k + 1 .. n], r − k)
  else
      return A[k]
```

   (a) State a recurrence for the expected running time of RANDOMSELECT, as a function of n and r.

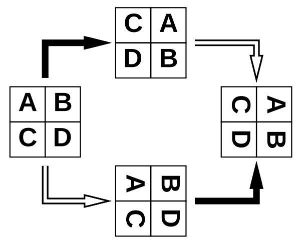
   (b) What is the *exact* probability that RANDOMSELECT compares the ith smallest and jth smallest elements in the input array? The correct answer is a simple function of i, j, and r. [Hint: Check your answer by trying a few small examples.]

   *(c) What is the expected running time of RANDOMSELECT, as a function of n and r? You can use either the recurrence from part (a) or the probabilities from part (b). For extra credit, give the *exact* expected number of comparisons.

   (d) What is the expected number of times that RANDOMSELECT calls itself recursively?

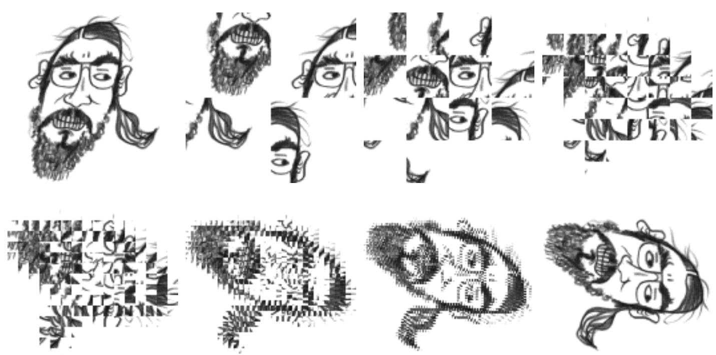
4. Some graphics hardware includes support for an operation called *blit*, or **bl**ock **t**ransfer, which quickly copies a rectangular chunk of a pixelmap (a two-dimensional array of pixel values) from one location to another. This is a two-dimensional version of the standard C library function `memcpy()`.

   Suppose we want to rotate an $n \times n$ pixelmap $90°$ clockwise. One way to do this is to split the pixelmap into four $n/2 \times n/2$ blocks, move each block to its proper position using a sequence of five blits, and then recursively rotate each block. Alternately, we can first recursively rotate the blocks and blit them into place afterwards.



Two algorithms for rotating a pixelmap.
Black arrows indicate blitting the blocks into place.
White arrows indicate recursively rotating the blocks.

The following sequence of pictures shows the first algorithm (blit then recurse) in action.



In the following questions, assume n is a power of two.

(a) Prove that both versions of the algorithm are correct. [Hint: If you exploit all the available symmetries, your proof will only be a half of a page long.]

(b) *Exactly* how many blits does the algorithm perform?

(c) What is the algorithm's running time if a $k \times k$ blit takes $O(k^2)$ time?

(d) What if a $k \times k$ blit takes only $O(k)$ time?

5. The traditional Devonian/Cornish drinking song "The Barley Mow" has the following pseudolyrics[1], where $container[i]$ is the name of a container that holds 2^i ounces of beer.[2]

```
BarleyMow(n):
  "Here's a health to the barley-mow, my brave boys,"
  "Here's a health to the barley-mow!"

  "We'll drink it out of the jolly brown bowl,"
  "Here's a health to the barley-mow!"
  "Here's a health to the barley-mow, my brave boys,"
  "Here's a health to the barley-mow!"

  for i ← 1 to n
      "We'll drink it out of the container[i], boys,"
      "Here's a health to the barley-mow!"
      for j ← i downto 1
          "The container[j],"
      "And the jolly brown bowl!"
      "Here's a health to the barley-mow!"
      "Here's a health to the barley-mow, my brave boys,"
      "Here's a health to the barley-mow!"
```

(a) Suppose each container name $container[i]$ is a single word, and you can sing four words a second. How long would it take you to sing BarleyMow(n)? (Give a tight asymptotic bound.)

(b) If you want to sing this song for $n > 20$, you'll have to make up your own container names, and to avoid repetition, these names will get progressively longer as n increases[3]. Suppose $container[n]$ has $\Theta(\log n)$ syllables, and you can sing six syllables per second. Now how long would it take you to sing BarleyMow(n)? (Give a tight asymptotic bound.)

(c) Suppose each time you mention the name of a container, you drink the corresponding amount of beer: one ounce for the jolly brown bowl, and 2^i ounces for each $container[i]$. Assuming for purposes of this problem that you are at least 21 years old, *exactly* how many ounces of beer would you drink if you sang BarleyMow(n)? (Give an *exact* answer, not just an asymptotic bound.)

[1]Pseudolyrics are to lyrics as pseudocode is to code.

[2]One version of the song uses the following containers: nipperkin, gill pot, half-pint, pint, quart, pottle, gallon, half-anker, anker, firkin, half-barrel, barrel, hogshead, pipe, well, river, and ocean. Every container in this list is twice as big as its predecessor, except that a firkin is actually 2.25 ankers, and the last three units are just silly.

[3]"We'll drink it out of the hemisemidemiyottapint, boys!"

6. *[This problem is required only for graduate students taking CS 373 for a full unit; anyone else can submit a solution for extra credit.]*

   A company is planning a party for its employees. The employees in the company are organized into a strict hierarchy, that is, a tree with the company president at the root. The organizers of the party have assigned a real number to each employee measuring how 'fun' the employee is. In order to keep things social, there is one restriction on the guest list: an employee cannot attend the party if their immediate supervisor is present. On the other hand, the president of the company *must* attend the party, even though she has a negative fun rating; it's her company, after all. Give an algorithm that makes a guest list for the party that maximizes the sum of the 'fun' ratings of the guests.

Practice Problems

1. Give an $O(n^2)$ algorithm to find the longest increasing subsequence of a sequence of numbers. The elements of the subsequence need not be adjacent in the sequence. For example, the sequence $\langle 1,5,3,2,4 \rangle$ has longest increasing subsequence $\langle 1,3,4 \rangle$.

2. You are at a political convention with n delegates. Each delegate is a member of exactly one political party. It is impossible to tell which political party a delegate belongs to. However, you can check whether any two delegates are in the *same* party or not by introducing them to each other. (Members of the same party always greet each other with smiles and friendly handshakes; members of different parties always greet each other with angry stares and insults.)

   (a) Suppose a majority (more than half) of the delegates are from the same political party. Give an efficient algorithm that identifies a member of the majority party.

   (b) Suppose exactly k political parties are represented at the convention and one party has a *plurality*: more delegates belong to that party than to any other. Present a practical procedure to pick a person from the plurality party as parsimoniously as possible. (Please.)

3. Give an algorithm that finds the *second* smallest of n elements in at most $n + \lceil \lg n \rceil - 2$ comparisons. [Hint: divide and conquer to find the smallest; where is the second smallest?]

4. Suppose that you have an array of records whose keys to be sorted consist only of 0's and 1's. Give a simple, linear-time $O(n)$ algorithm to sort the array in place (using storage of no more than constant size in addition to that of the array).

5. Consider the problem of making change for n cents using the least number of coins.

   (a) Describe a greedy algorithm to make change consisting of quarters, dimes, nickels, and pennies. Prove that your algorithm yields an optimal solution.

   (b) Suppose that the available coins have the values $c^0, c^1, \ldots, c^k$ for some integers $c > 1$ and $k \geq 1$. Show that the obvious greedy algorithm always yields an optimal solution.

   (c) Give a set of 4 coin values for which the greedy algorithm does not yield an optimal solution.

   (d) Describe a dynamic programming algorithm that yields an optimal solution for an arbitrary set of coin values.

   (e) Suppose we have only two types of coins whose values a and b are relatively prime. Prove that any value of greater than $(a-1)(b-1)$ can be made with these two coins.

   *(f) For only three coins a, b, c whose greatest common divisor is 1, give an algorithm to determine the smallest value n such that change *can* be given for all values greater than n. [Note: this problem is currently unsolved for more than four coins!]

6. Suppose you have a subroutine that can find the median of a set of n items (*i.e.*, the $\lfloor n/2 \rfloor$ smallest) in $O(n)$ time. Give an algorithm to find the kth biggest element (for arbitrary k) in $O(n)$ time.

7. You're walking along the beach and you stub your toe on something in the sand. You dig around it and find that it is a treasure chest full of gold bricks of different (integral) weight. Your knapsack can only carry up to weight n before it breaks apart. You want to put as much in it as possible without going over, but you *cannot* break the gold bricks up.

   (a) Suppose that the gold bricks have the weights $1, 2, 4, 8, \ldots, 2^k$, $k \geq 1$. Describe and prove correct a greedy algorithm that fills the knapsack as much as possible without going over.

   (b) Give a set of 3 weight values for which the greedy algorithm does *not* yield an optimal solution and show why.

   (c) Give a dynamic programming algorithm that yields an optimal solution for an arbitrary set of gold brick values.

CS 373: Combinatorial Algorithms, Fall 2000
Homework 2 (due September 28, 2000 at midnight)

Name:		
Net ID:	Alias:	U $^3\!/\!_4$ 1

Name:		
Net ID:	Alias:	U $^3\!/\!_4$ 1

Name:		
Net ID:	Alias:	U $^3\!/\!_4$ 1

Starting with Homework 1, homeworks may be done in teams of up to three people. Each team turns in just one solution, and every member of a team gets the same grade. Since 1-unit graduate students are required to solve problems that are worth extra credit for other students, **1-unit grad students may not be on the same team as 3/4-unit grad students or undergraduates.**

Neatly print your name(s), NetID(s), and the alias(es) you used for Homework 0 in the boxes above. Please also tell us whether you are an undergraduate, 3/4-unit grad student, or 1-unit grad student by circling U, $^3\!/\!_4$, or 1, respectively. Staple this sheet to the top of your homework.

Required Problems

1. Faster Longest Increasing Subsequence (15 pts)
   Give an $O(n \log n)$ algorithm to find the longest increasing subsequence of a sequence of numbers. *[Hint: In the dynamic programming solution, you don't really have to look back at all previous items. There was a practice problem on HW 1 that asked for an $O(n^2)$ algorithm for this. If you are having difficulty, look at the HW 1 solutions.]*

2. SELECT(A, k) (10 pts)
   Say that a binary search tree is *augmented* if every node v also stores $|v|$, the size of its subtree.

   (a) Show that a rotation in an augmented binary tree can be performed in constant time.

   (b) Describe an algorithm SCAPEGOATSELECT(k) that selects the kth smallest item in an augmented scapegoat tree in $O(\log n)$ *worst-case* time.

   (c) Describe an algorithm SPLAYSELECT(k) that selects the kth smallest item in an augmented splay tree in $O(\log n)$ *amortized* time.

(d) Describe an algorithm TREAPSELECT(k) that selects the kth smallest item in an augmented treap in $O(\log n)$ *expected* time.

[Hint: The answers for (b), (c), and (d) are almost exactly the same!]

3. Scapegoat trees (15 pts)

   (a) Prove that only one subtree gets rebalanced in a scapegoat tree insertion.

   (b) Prove that $I(v) = 0$ in every node of a perfectly balanced tree. (Recall that $I(v) = \max\{0, |T| - |s| - 1\}$, where T is the child of greater height and s the child of lesser height, and $|v|$ is the number of nodes in subtree v. A perfectly balanced tree has two perfectly balanced subtrees, each with as close to half the nodes as possible.)

   *(c) Show that you can rebuild a fully balanced binary tree from an unbalanced tree in $O(n)$ time using only $O(\log n)$ additional memory. For 5 extra credit points, use only $O(1)$ additional memory.

4. Memory Management (10 pts)
   Suppose we can insert or delete an element into a hash table in constant time. In order to ensure that our hash table is always big enough, without wasting a lot of memory, we will use the following global rebuilding rules:

   - After an insertion, if the table is more than 3/4 full, we allocate a new table twice as big as our current table, insert everything into the new table, and then free the old table.
   - After a deletion, if the table is less than 1/4 full, we we allocate a new table half as big as our current table, insert everything into the new table, and then free the old table.

   Show that for any sequence of insertions and deletions, the amortized time per operation is still a constant. Do *not* use the potential method—it makes the problem much too hard!

5. Fibonacci Heaps: SECONDMIN (10 pts)

   (a) Implement SECONDMIN by using a Fibonacci heap as a black box. Remember to justify its correctness and running time.

   *(b) Modify the Fibonacci Heap data structure to implement the SECONDMIN operation in constant time, without degrading the performance of any other Fibonacci heap operation.

Practice Problems

1. Amortization

   (a) Modify the binary double-counter (see class notes Sept 12) to support a new operation SIGN, which determines whether the number being stored is positive, negative, or zero, in constant time. The amortized time to increment or decrement the counter should still be a constant.

   [Hint: Suppose p is the number of significant bits in P, and n is the number of significant bits in N. For example, if $P = 17 = 10001_2$ and $N = 0$, then $p = 5$ and $n = 0$. Then $p - n$ always has the same sign as $P - N$. Assume you can update p and n in $O(1)$ time.]

   *(b) Do the same but now you can't assume that p and n can be updated in $O(1)$ time.

*2. Amortization
   Suppose instead of powers of two, we represent integers as the sum of Fibonacci numbers. In other words, instead of an array of bits, we keep an array of 'fits', where the ith least significant fit indicates whether the sum includes the ith Fibonacci number F_i. For example, the fit string 101110 represents the number $F_6 + F_4 + F_3 + F_2 = 8 + 3 + 2 + 1 = 14$. Describe algorithms to increment and decrement a fit string in constant amortized time. *[Hint: Most numbers can be represented by more than one fit string. This is not the same representation as on Homework 0!]*

3. Rotations

   (a) Show that it is possible to transform any n-node binary search tree into any other n-node binary search tree using at most $2n - 2$ rotations.

   *(b) Use fewer than $2n - 2$ rotations. Nobody knows how few rotations are required in the worst case. There is an algorithm that can transform any tree to any other in at most $2n - 6$ rotations, and there are pairs of trees that are $2n - 10$ rotations apart. These are the best bounds known.

4. Give an efficient implementation of the operation CHANGEKEY(x, k), which changes the key of a node x in a Fibonacci heap to the value k. The changes you make to Fibonacci heap data structure to support your implementation should not affect the amortized running time of any other Fibonacci heap operations. Analyze the amortized running time of your implementation for cases in which k is greater than, less than, or equal to $key[x]$.

5. Detecting overlap

   (a) You are given a list of ranges represented by min and max (*e.g.*, [1,3], [4,5], [4,9], [6,8], [7,10]). Give an $O(n \log n)$-time algorithm that decides whether or not a set of ranges contains a pair that overlaps. You need not report all intersections. If a range completely covers another, they are overlapping, even if the boundaries do not intersect.

(b) You are given a list of rectangles represented by min and max x- and y-coordinates. Give an $O(n \log n)$-time algorithm that decides whet her or not a set of rectangles contains a pair that overlaps (with the same qualifications as above). [Hint: sweep a vertical line from left to right, performing some processing whenever an end-point is encountered. Use a balanced search tree to maintain any extra info you might need.]

6. Comparison of Amortized Analysis Methods
A sequence of n operations is performed on a data structure. The ith operation costs i if i is an exact power of 2, and 1 otherwise. That is operation i costs $f(i)$, where:

$$f(i) = \begin{cases} i, & i = 2^k, \\ 1, & \text{otherwise} \end{cases}$$

Determine the amortized cost per operation using the following methods of analysis:

(a) Aggregate method

(b) Accounting method

*(c) Potential method

CS 373: Combinatorial Algorithms, Fall 2000
Homework 3 (due October 17, 2000 at midnight)

Name:		
Net ID:	Alias:	U $^3/_4$ 1

Name:		
Net ID:	Alias:	U $^3/_4$ 1

Name:		
Net ID:	Alias:	U $^3/_4$ 1

Starting with Homework 1, homeworks may be done in teams of up to three people. Each team turns in just one solution, and every member of a team gets the same grade. Since 1-unit graduate students are required to solve problems that are worth extra credit for other students, **1-unit grad students may not be on the same team as 3/4-unit grad students or undergraduates.**

Neatly print your name(s), NetID(s), and the alias(es) you used for Homework 0 in the boxes above. Please also tell us whether you are an undergraduate, 3/4-unit grad student, or 1-unit grad student by circling U, $^3/_4$, or 1, respectively. Staple this sheet to the top of your homework.

Required Problems

1. Suppose you have to design a dictionary that holds 2048 items.

   (a) How many probes are used for an unsuccessful search if the dictionary is implemented as a sorted array? Assume the use of Binary Search.

   (b) How large a hashtable do you need if your goal is to have 2 as the expected number of probes for an unsuccessful search?

   (c) How much more space is needed by the hashtable compared to the sorted array? Assume that each pointer in a linked list takes 1 word of storage.

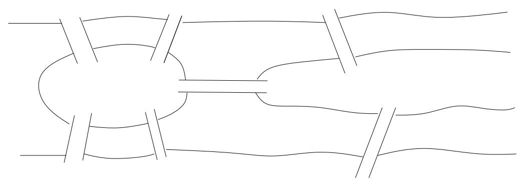
2. In order to facilitate recompiling programs from multiple source files when only a small number of files have been updated, there is a UNIX utility called 'make' that only recompiles those files that were changed after the most recent compilation, *and* any intermediate files in the compilation that depend on those that were changed. A Makefile is typically composed of a list of source files that must be compiled. Each of these source files is dependent on some of

the other files which are listed. Thus a source file must be recompiled if a file on which it depends is changed.

Assuming you have a list of which files have been recently changed, as well as a list for each source file of the files on which it depends, design an algorithm to recompile only those necessary. Don't worry about the details of parsing a Makefile.

3. A person wants to fly from city A to city B in the shortest possible time. She turns to the traveling agent who knows all the departure and arrival times of all the flights on the planet. Give an algorithm that will allow the agent to choose a route with the minimum total travel time—initial takeoff to final landing, including layovers. *[Hint: Modify the data and call a shortest-path algorithm.]*

4. During the eighteenth century the city of Königsberg in East Prussia was divided into four sections by the Pregel river. Seven bridges connected these regions, as shown below. It was said that residents spent their Sunday walks trying to find a way to walk about the city so as to cross each bridge exactly once and then return to their starting point.



   (a) Show how the residents of the city could accomplish such a walk or prove no such walk exists.
   (b) Given any undirected graph $G = (V, E)$, give an algorithm that finds a cycle in the graph that visits every edge exactly once, or says that it can't be done.

5. Suppose you have a graph G and an MST of that graph (i.e. the MST has already been constructed).
   (a) Give an algorithm to update the MST when an edge is added to G.
   (b) Give an algorithm to update the MST when an edge is deleted from G.
   (c) Give an algorithm to update the MST when a vertex (and possibly edges to it) is added to G.

6. *[This problem is required only for graduate students taking CS 373 for a full unit; anyone else can submit a solution for extra credit.]*

   You are given an unlimited number of each of n different types of envelopes. The dimensions of envelope type i are $x_i \times y_i$. In nesting envelopes inside one another, you can place envelope A inside envelope B if and only if the dimensions A are *strictly smaller* than the dimensions of B. Design and analyze an algorithm to determine the largest number of envelopes that can be nested inside one another.

Practice Problems

★1. Let the hash function for a table of size m be

$$h(x) = \lfloor Amx \rfloor \bmod m$$

where $A = \hat{\phi} = \frac{\sqrt{5}-1}{2}$. Show that this gives the best possible spread, i.e. if the x are hashed in order, $x + 1$ will be hashed in the largest remaining contiguous interval.

2. The incidence matrix of an undirected graph $G = (V, E)$ is a $|V| \times |E|$ matrix $B = (b_{ij})$ such that

$$b_{ij} = \big[(i, j) \in E\big] = \begin{cases} 1 & \text{if } (i, j) \in E, \\ 0 & \text{if } (i, j) \notin E. \end{cases}$$

   (a) Describe what all the entries of the matrix product BB^T represent (B^T is the matrix transpose).

   (b) Describe what all the entries of the matrix product $B^T B$ represent.

   ★(c) Let $C = BB^T - 2A$, where A is the adjacency matrix of G, with zeroes on the diagonal. Let C' be C with the first row and column removed. Show that $\det C'$ is the number of spanning trees.

3. (a) Give an $O(V)$ algorithm to decide whether a directed graph contains a *sink* in an adjacency matrix representation. A sink is a vertex with in-degree $V - 1$.

   (b) An undirected graph is a scorpion if it has a vertex of degree 1 (the sting) connected to a vertex of degree two (the tail) connected to a vertex of degree $V - 2$ (the body) connected to the other $V - 3$ vertices (the feet). Some of the feet may be connected to other feet.

   Design an algorithm that decides whether a given adjacency matrix represents a scorpion by examining only $O(V)$ of the entries.

   (c) Show that it is impossible to decide whether G has at least one edge in $O(V)$ time.

4. Given an *undirected* graph $G = (V, E)$, and a weight function $f : E \to \mathbb{R}$ on the *edges*, give an algorithm that finds (in time polynomial in V and E) a cycle of smallest weight in G.

5. Let $G = (V, E)$ be a graph with n vertices. A *simple path* of G, is a path that does not contain the same vertex twice. Use dynamic programming to design an algorithm (not polynomial time) to find a simple path of maximum length in G. Hint: It can be done in $O(n^c 2^n)$ time, for some constant c.

6. Suppose all edge weights in a graph G are equal. Give an algorithm to compute a minimum spanning tree of G.

7. Give an algorithm to construct a *transitive reduction* of a directed graph G, i.e. a graph G^{TR} with the fewest edges (but with the same vertices) such that there is a path from a to b in G iff there is also such a path in G^{TR}.

8. (a) What is $5^{2^{29^{5^0}+23^{4^1}+17^{3^2}+11^{2^3}+5^{1^4}}} \bmod 6$?

   (b) What is the capital of Nebraska? Hint: It is not Omaha. It is named after a famous president of the United States that was not George Washington. The distance from the Earth to the Moon averages roughly 384,000 km.

CS 373: Combinatorial Algorithms, Fall 2000
Homework 4 (due October 26, 2000 at midnight)

Name:		
Net ID:	Alias:	U $^3\!/_4$ 1

Name:		
Net ID:	Alias:	U $^3\!/_4$ 1

Name:		
Net ID:	Alias:	U $^3\!/_4$ 1

Homeworks may be done in teams of up to three people. Each team turns in just one solution, and every memeber of a team gets the same grad. Since 1-unit graduate students are required to solve problems that are worth extra credit for other students, **1-unit grad students may not be on the same team as 3/4-unit grad students or undergraduates.**

Neatly print your name(s), NetID(s), and the alias(es) you used for Homework 0 in the boxes above. Please also tell us whether you are an undergraduate, 3/4-unit grad student, or 1-unit grad student by circling U, $^3\!/_4$, or 1, respectively. Staple this sheet to the top of your homework.

RequiredProblems

1. (10 points) A certain algorithms professor once claimed that the height of an n-node Fibonacci heap is of height $O(\log n)$. Disprove his claim by showing that for a positive integer n, a sequence of Fibonacci heap operations that creates a Fibonacci heap consisting of just one tree that is a (downward) linear chain of n nodes.

2. (20 points) *Fibonacci strings* are defined as follows:

$$\begin{aligned} F_1 &= \mathsf{b} \\ F_2 &= \mathsf{a} \\ F_n &= F_{n-1}F_{n-2} \quad \text{for all } n > 2 \end{aligned}$$

   where the recursive rule uses concatenation of strings, so $F_3 = \mathsf{ab}$, $F_4 = \mathsf{aba}$, and so on. Note that the length of F_n is the nth Fibonacci number.

   (a) Prove that in any Fibonacci string there are no two b's adjacent and no three a's.

(b) Give the unoptimized and optimized failure function for F_7.

(c) Prove that, in searching for the Fibonacci string F_k, the unoptimized KMP algorithm may shift $\lceil k/2 \rceil$ times on the same text character. In other words, prove that there is a chain of failure links $j \to fail[j] \to fail[fail[j]] \to \ldots$ of length $\lceil k/2 \rceil$, and find an example text T that would cause KMP to traverse this entire chain on the same position in the text.

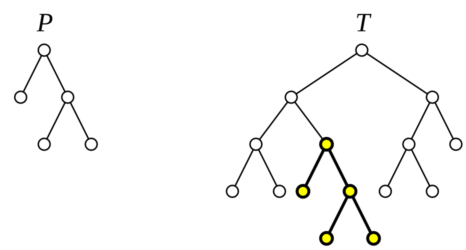
(d) What happens here when you use the optimized prefix function? Explain.

3. (10 points) Show how to extend the Rabin-Karp fingerprinting method to handle the problem of looking for a given $m \times m$ pattern in an $n \times n$ array of characters. The pattern may be shifted horizontally and vertically, but it may not be rotated.

4. (10 points)

(a) A *cyclic rotation* of a string is obtained by chopping off a prefix and gluing it at the end of the string. For example, ALGORITHM is a cyclic shift of RITHMALGO. Describe and analyze an algorithm that determines whether one string $P[1 .. m]$ is a cyclic rotation of another string $T[1 .. n]$.

(b) Describe and analyze an algorithm that decides, given any two binary trees P and T, whether P equals a subtree of T. We want an algorithm that compares the *shapes* of the trees. There is no data stored in the nodes, just pointers to the left and right children. *[Hint: First transform both trees into strings.]*



P occurs exactly once as a subtree of T.

5. (10 points) *[This problem is required only for graduate students taking CS 373 for a full unit; anyone else can submit a solution for extra credit.]*

Refer to the notes for lecture 11 for this problem. The GenericSSSP algorithm described in class can be implemented using a stack for the 'bag'. Prove that the resulting algorithm can be forced to perform in $\Omega(2^n)$ relaxation steps. To do this, you need to describe, for any positive integer n, a specific weighted directed n-vertex graph that forces this exponential behavior. The easiest way to describe such a family of graphs is using an *algorithm*!

Practice Problems

1. String matching with wild-cards
   Suppose you have an alphabet for patterns that includes a 'gap' or wild-card character; any length string of any characters can match this additional character. For example if '*' is the wild-card, then the pattern foo*bar*nad can be found in foofoowangbarnad. Modify the computation of the prefix function to correctly match strings using KMP.

2. Prove that there is no comparison sort whose running time is linear for at least 1/2 of the $n!$ inputs of length n. What about at least $1/n$? What about at least $1/2^n$?.

3. Prove that $2n - 1$ comparisons are necessary in the worst case to merge two sorted lists containing n elements each.

4. Find asymptotic upper and lower bounds to $\lg(n!)$ without Stirling's approximation (Hint: use integration).

5. Given a sequence of n elements of n/k blocks (k elements per block) all elements in a block are less than those to the right in sequence, show that you cannot have the whole sequence sorted in better than $\Omega(n \lg k)$. Note that the entire sequence would be sorted if each of the n/k blocks were individually sorted in place. Also note that combining the lower bounds for each block is not adequate (that only gives an upper bound).

6. Show how to find the occurrences of pattern P in text T by computing the prefix function of the string PT (the concatenation of P and T).

7. Lower Bounds on Adjacency Matrix Representations of Graphs
   (a) Prove that the time to determine if an undirected graph has a cycle is $\Omega(V^2)$.
   (b) Prove that the time to determine if there is a path between two nodes in an undirected graph is $\Omega(V^2)$.

CS 373: Combinatorial Algorithms, Fall 2000

Homework 1 (due November 16, 2000 at midnight)

Name:		
Net ID:	Alias:	U $^3\!/_4$ 1

Name:		
Net ID:	Alias:	U $^3\!/_4$ 1

Name:		
Net ID:	Alias:	U $^3\!/_4$ 1

Starting with Homework 1, homeworks may be done in teams of up to three people. Each team turns in just one solution, and every member of a team gets the same grade. Since 1-unit graduate students are required to solve problems that are worth extra credit for other students, **1-unit grad students may not be on the same team as 3/4-unit grad students or undergraduates.**

Neatly print your name(s), NetID(s), and the alias(es) you used for Homework 0 in the boxes above. Please also tell us whether you are an undergraduate, 3/4-unit grad student, or 1-unit grad student by circling U, $^3\!/_4$, or 1, respectively. Staple this sheet to the top of your homework.

Required Problems

1. Give an $O(n^2 \log n)$ algorithm to determine whether any three points of a set of n points are collinear. Assume two dimensions and *exact* arithmetic.

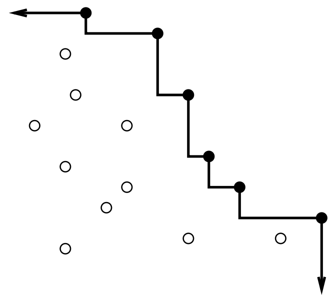
2. We are given an array of n bits, and we want to determine if it contains two consecutive 1 bits. Obviously, we can check every bit, but is this always necessary?

   (a) (4 pts) Show that when n mod 3 = 0 or 2, we must examine every bit in the array. that is, give an adversary strategy that forces any algorithm to examine every bit when $n = 2, 3, 5, 6, 8, 9, \ldots$.

   (b) (4 pts) Show that when $n = 3k + 1$, we only have to examine $n - 1$ bits. That is, describe an algorithm that finds two consecutive 1s or correctly reports that there are none after examining at most $n - 1$ bits, when $n = 1, 4, 7, 10, \ldots$.

   (c) (2 pts) How many n-bit strings are there with two consecutive ones? For which n is this number even or odd?

3. You are given a set of points in the plane. A point is *maximal* if there is no other point both both above and to the right. The subset of maximal points of points then forms a *staircase*.



The staircase of a set of points. Maximal points are black.

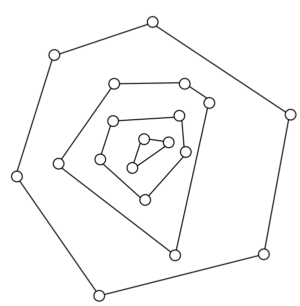
   (a) (0 pts) Prove that maximal points are *not* necessarily on the convex hull.

   (b) (6 pts) Give an $O(n \log n)$ algorithm to find the maximal points.

   (c) (4 pts) Assume that points are chosen uniformly at random within a rectangle. What is the average number of maximal points? Justify. Hint: you will be able to give an exact answer rather than just asymptotics. You have seen the same analysis before.

4. Given a set Q of points in the plane, define the *convex layers* of Q inductively as follows: The first convex layer of Q is just the convex hull of Q. For all $i > 1$, the ith convex layer is the convex hull of Q after the vertices of the first $i - 1$ layers have been removed.

   Give an $O(n^2)$-time algorithm to find all convex layers of a given set of n points.



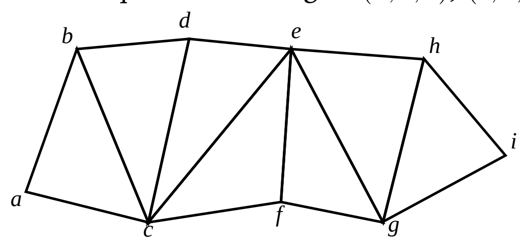
A set of points with four convex layers.

5. Prove that finding the second smallest of n elements takes $n + \lceil \lg n \rceil - 2$ comparisons in the worst case. Prove for both upper and lower bounds. Hint: find the (first) smallest using an elimination tournament.

6. *[This problem is required only for graduate students taking CS 373 for a full unit; anyone else can submit a solution for extra credit.]*

   Almost all computer graphics systems, at some level, represent objects as collections of triangles. In order to minimize storage space and rendering time, many systems allow objects to be stored as a set of *triangle strips*. A triangle strip is a sequence of vertices $\langle v_1, v_2, \ldots, v_k \rangle$, where each contiguous triple of vertices v_i, v_{i+1}, v_{i+2} represents a triangle. As the rendering system reads the sequence of vertices and draws the triangles, it keeps the two most recent vertices in a cache.

   Some systems allow triangle strips to contain *swaps*: special flags indicating that the order of the two cached vertices should be reversed. For example, the triangle strip $\langle a, b, c, d, \mathsf{swap}, e, f, \mathsf{swap}, g, h, i \rangle$ represents the sequence of triangles (a, b, c), (b, c, d), (d, c, e), (c, e, f), (f, e, g).



   Two triangle strips are *disjoint* if they share no triangles (although they may share vertices). The *length* of a triangle strip is the length of its vertex sequence, including swaps; for example, the example strip above has length 11. A *pure* triangle strip is one with no swaps. The adjacency graph of a triangle strip is a simple path. If the strip is pure, this path alternates between left and right turns.

   Suppose you are given a set S of interior-disjoint triangles whose adjacency graph is a tree. (In other words, S is a triangulation of a simple polygon.) Describe a linear-time algorithm to decompose S into a set of disjoint triangle strips of minimum total length.

Practice Problems

1. Consider the following generic recurrence for convex hull algorithms that divide and conquer:

$$T(n, h) = T(n_1, h_1) + T(n_2, h_2) + O(n)$$

   where $n \geq n_1 + n_2, h = h_1 + h_2$ and $n \geq h$. This means that the time to compute the convex hull is a function of both n, the number of input points, and h, the number of convex hull vertices. The splitting and merging parts of the divide-and-conquer algorithm take $O(n)$ time. When n is a constant, $T(n, h) = O(1)$, but when h is a constant, $T(n, h) = O(n)$. Prove that for both of the following restrictions, the solution to the recurrence is $O(n \log h)$:

   (a) $h_1, h_2 < \frac{3}{4}h$

   (b) $n_1, n_2 < \frac{3}{4}n$

2. Circle Intersection
   Give an $O(n \log n)$ algorithm to test whether any two circles in a set of size n intersect.

3. Basic polygon computations (assume *exact* arithmetic)

   (a) Intersection: Extend the basic algorithm to determine if two line segments intersect by taking care of *all* degenerate cases.

   (b) Simplicity: Give an $O(n \log n)$ algorithm to determine whether an n-vertex polygon is simple.

   (c) Area: Give an algorithm to compute the area of a simple n-polygon (not necessarily convex) in $O(n)$ time.

   (d) Inside: Give an algorithm to determine whether a point is within a simple n-polygon (not necessarily convex) in $O(n)$ time.

4. We are given the set of points one point at a time. After receiving each point, we must compute the convex hull of all those points so far. Give an algorithm to solve this problem in $O(n^2)$ total time. (We could obviously use Graham's scan n times for an $O(n^2 \log n)$-time algorithm). Hint: How do you maintain the convex hull?

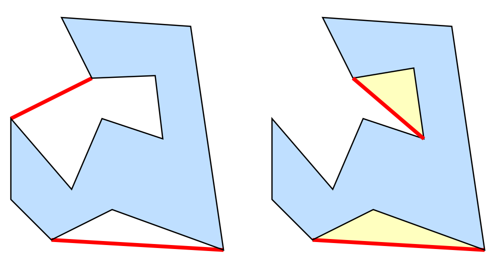
5. *(a) Given an n-polygon and a point outside the polygon, give an algorithm to find a tangent.

   (b) Suppose you have found both tangents. Give an algorithm to remove the points from the polygon that are within the angle formed by the tangents (as segments!) and the opposite side of the polygon.

   (c) Use the above to give an algorithm to compute the convex hull on-line in $O(n \log n)$

6. (a) A pair of polygon vertices defines an *external diagonal* if the line segment between them is completely outside the polygon. Show that every nonconvex polygon has at least one external diagonal.

   (b) Three consective polygon vertices p, q, r form a *mouth* if p and r define an external diagonal. Show that every nonconvex polygon has at least one mouth.



Two external diagonals Two mouths

7. A group of n ghostbusters is battling n ghosts. Each ghostbuster can shoot a single energy beam at a ghost, eradicating it. A stream goes in a straight line and terminates when it hits the ghost. The ghostbusters all fire at the same time and no two energy beams may cross. The positions of the ghosts and ghostbusters are fixed points in the plane.

   (a) Prove that for any configuration of ghosts and ghostbusters, there is such a non-crossing matching. (Assume that no three points are collinear.)

(b) Show that there is a line passing through one ghostbuster and one ghost such that the number of ghostbusters on one side of the line equals the number of ghosts on the same side. Give an efficient algorithm to find such a line.

(c) Give an efficient divide and conquer algorithm to pair ghostbusters and ghosts so that no two streams cross.

CS 373: Combinatorial Algorithms, Fall 2000

Homework 6 (due December 7, 2000 at midnight)

Name:		
Net ID:	Alias:	U $^3\!/\!_4$ 1

Name:		
Net ID:	Alias:	U $^3\!/\!_4$ 1

Name:		
Net ID:	Alias:	U $^3\!/\!_4$ 1

Starting with Homework 1, homeworks may be done in teams of up to three people. Each team turns in just one solution, and every member of a team gets the same grade. Since 1-unit graduate students are required to solve problems that are worth extra credit for other students, **1-unit grad students may not be on the same team as 3/4-unit grad students or undergraduates.**

Neatly print your name(s), NetID(s), and the alias(es) you used for Homework 0 in the boxes above. Please also tell us whether you are an undergraduate, 3/4-unit grad student, or 1-unit grad student by circling U, $^3\!/\!_4$, or 1, respectively. Staple this sheet to the top of your homework.

Required Problems

1. (a) Prove that P $\subseteq$ co-NP.

   (b) Show that if NP $\neq$ co-NP, then *no* NP-complete problem is a member of co-NP.

2. 2SAT is a special case of the formula satisfiability problem, where the input formula is in conjunctive normal form and every clause has at most *two* literals. Prove that 2SAT is in P.

3. Describe an algorithm that solves the following problem, called 3SUM, as quickly as possible: Given a set of n numbers, does it contain three elements whose sum is zero? For example, your algorithm should answer TRUE for the set $\{-5, -17, 7, -4, 3, -2, 4\}$, since $-5+7+(-2) = 0$, and FALSE for the set $\{-6, 7, -4, -13, -2, 5, 13\}$.

4. (a) Show that the problem of deciding whether one undirected graph is a subgraph of another is NP-complete.
   (b) Show that the problem of deciding whether an unweighted undirected graph has a path of length greater than k is NP-complete.

5. (a) Consider the following problem: Given a set of axis-aligned rectangles in the plane, decide whether any point in the plane is covered by k or more rectangles. Now also consider the CLIQUE problem. Describe and analyze a reduction of one problem to the other.
   (b) Finding the largest clique in an arbitrary graph is NP-hard. What does this fact imply about the complexity of finding a point that lies inside the largest number of rectangles?

6. *[This problem is required only for graduate students taking CS 373 for a full unit; anyone else can submit a solution for extra credit.]*

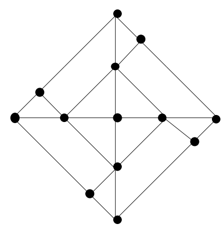
   PARTITION is the problem of deciding, given a set $S = \{s_1, s_2, \ldots, s_n\}$ of numbers, whether there is a subset T containing half the 'weight' of S, *i.e.*, such that $\sum T = \frac{1}{2} \sum S$. SUBSETSUM is the problem of deciding, given a set $S = \{s_1, s_2, \ldots, s_n\}$ of numbers and a target sum t, whether there is a subset $T \subseteq S$ such that $\sum T = t$. Give two reductions between these two problems, one in each direction.

Practice Problems

1. What is the *exact* worst case number of comparisons needed to find the median of 5 numbers? For 6 numbers?

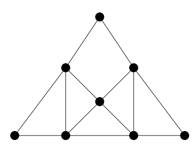
2. The ExactCoverByThrees problem is defined as follows: given a finite set X and a collection C of 3-element subsets of X, does C contain an *exact cover* for X, that is, a subcollection $C' \subseteq C$ where every element of X occurs in exactly one member of C'? Given that ExactCoverByThrees is NP-complete, show that the similar problem ExactCoverByFours is also NP-complete.

3. Using 3Color and the 'gadget' below, prove that the problem of deciding whether a planar graph can be 3-colored is NP-complete. *[Hint: Show that the gadget can be 3-colored, and then replace any crossings in a planar embedding with the gadget appropriately.]*



Crossing gadget for Planar3Color.

4. Using the previous result, and the 'gadget' below, prove that the problem of deciding whether a planar graph with no vertex of degree greater than four can be 3-colored is NP-complete. *[Hint: Show that you can replace any vertex with degree greater than 4 with a collection of gadgets connected in such a way that no degree is greater than four.]*



Degree gadget for Degree4Planar3Color

5. Show that an algorithm that makes at most a constant number of calls to polynomial-time subroutines runs in polynomial time, but that a polynomial number of calls to polynomial-time subroutines may result in an exponential-time algorithm.

6. (a) Prove that if G is an undirected bipartite graph with an odd number of vertices, then G is nonhamiltonian. Give a polynomial time algorithm algorithm for finding a hamiltonian cycle in an undirected bipartite graph or establishing that it does not exist.
   (b) Show that the hamiltonian-path problem can be solved in polynomial time on directed acyclic graphs.
   (c) Explain why the results in previous questions do not contradict the fact that both HamiltonianCycle and HamiltonianPath are NP-complete problems.

7. Consider the following pairs of problems:

(a) MIN SPANNING TREE and MAX SPANNING TREE

(b) SHORTEST PATH and LONGEST PATH

(c) TRAVELING SALESMAN and VACATION TOUR (the longest tour is sought).

(d) MIN CUT and MAX CUT (between s and t)

(e) EDGE COVER and VERTEX COVER

(f) TRANSITIVE REDUCTION and MIN EQUIVALENT DIGRAPH

(All of these seem dual or opposites, except the last, which are just two versions of minimal representation of a graph.) Which of these pairs are polytime equivalent and which are not?

★8. Consider the problem of deciding whether one graph is isomorphic to another.

(a) Give a brute force algorithm to decide this.

(b) Give a dynamic programming algorithm to decide this.

(c) Give an efficient probabilistic algorithm to decide this.

★(d) Either prove that this problem is NP-complete, give a poly time algorithm for it, or prove that neither case occurs.

*9. Prove that PRIMALITY (Given n, is n prime?) is in NP $\cap$ co-NP. Showing that PRIMALITY is in co-NP is easy. (What's a certificate for showing that a number is composite?) For NP, consider a certificate involving primitive roots and recursively their primitive roots. Show that this tree of primitive roots can be checked to be correct and used to show that n is prime, and that this check takes polynomial time.

10. How much wood would a woodchuck chuck if a woodchuck could chuck wood?

CS 373: Combinatorial Algorithms, Fall 2000

Midterm 1 — October 3, 2000

Name:		
Net ID:	Alias:	U $^{3}/_{4}$ 1

This is a closed-book, closed-notes exam!

If you brought anything with you besides writing instruments and your $8\frac{1}{2}'' \times 11''$ cheat sheet, please leave it at the front of the classroom.

- Print your name, netid, and alias in the boxes above. Circle U if you are an undergrad, $^{3}/_{4}$ if you are a 3/4-unit grad student, or 1 if you are a 1-unit grad student. Print your name at the top of every page (in case the staple falls out!).
- **Answer four of the five questions on the exam.** Each question is worth 10 points. If you answer more than four questions, the one with the lowest score will be ignored. **1-unit graduate students must answer question 5.**
- Please write your final answers on the front of the exam pages. Use the backs of the pages as scratch paper. Let us know if you need more paper.
- Unless we specifically say otherwise, proofs are not required. However, they may help us give you partial credit.
- Read the entire exam before writing anything. Make sure you understand what the questions are asking. If you give a beautiful answer to the wrong question, you'll get no credit. If any question is unclear, please ask one of us for clarification.
- Don't spend too much time on any single problem. If you get stuck, move on to something else and come back later.
- Write something down for every problem. Don't panic and erase large chunks of work. Even if you think it's absolute nonsense, it might be worth partial credit.
- Relax. Breathe. Kick some ass.

#	Score	Grader
1		
2		
3		
4		
5		
Total		

1. **Multiple Choice**

Every question below has one of the following answers.

(a) $\Theta(1)$ (b) $\Theta(\log n)$ (c) $\Theta(n)$ (d) $\Theta(n \log n)$ (e) $\Theta(n^2)$

For each question, write the letter that corresponds to your answer. You do not need to justify your answers. Each correct answer earns you 1 point, but each incorrect answer *costs* you $\frac{1}{2}$ point. You cannot score below zero.

☐ What is $\sum_{i=1}^{n} \log i$?

☐ What is $\sum_{i=1}^{n} \frac{n}{i}$?

☐ How many digits do you need to write 2^n in decimal?

☐ What is the solution of the recurrence $T(n) = 25T(n/5) + n$?

☐ What is the solution of the recurrence $T(n) = T(n-1) + \dfrac{1}{2^n}$?

☐ What is the solution of the recurrence $T(n) = 3T\left(\left\lceil \frac{n+51}{3} \right\rceil\right) + 17n - \sqrt[7]{\lg \lg n} - 2^{2^{\log^* n}} + \pi$?

☐ What is the worst-case running time of randomized quicksort?

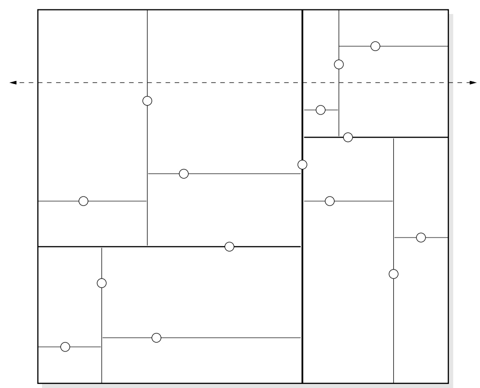
☐ The expected time for inserting one item into an n-node randomized treap is $O(\log n)$. What is the worst-case time for a sequence of n insertions into an initially empty treap?

☐ The amortized time for inserting one item into an n-node scapegoat tree is $O(\log n)$. What is the worst-case time for a sequence of n insertions into an initially empty scapegoat tree?

☐ In the worst case, how many nodes can be in the root list of a Fibonacci heap storing n keys, immediately after a DECREASEKEY operation?

☐ Every morning, an Amtrak train leaves Chicago for Champaign, 200 miles away. The train can accelerate or decelerate at 10 miles per hour per second, and it has a maximum speed of 60 miles an hour. Every 50 miles, the train must stop for five minutes while a school bus crosses the tracks. Every hour, the conductor stops the train for a union-mandated 10-minute coffee break. How long does it take the train to reach Champaign?

2. Suppose we have n points scattered inside a two-dimensional box. A *kd-tree* recursively subdivides the rectangle as follows. First we split the box into two smaller boxes with a *vertical* line, then we split each of those boxes with *horizontal* lines, and so on, always alternating between horizontal and vertical splits. Each time we split a box, the splitting line passes through some point inside the box (*not* on the boundary) and partitions the rest of the interior points as evenly as possible. If a box doesn't contain any points, we don't split it any more; these final empty boxes are called *cells*.



A kd-tree for 15 points. The dashed line crosses four cells.

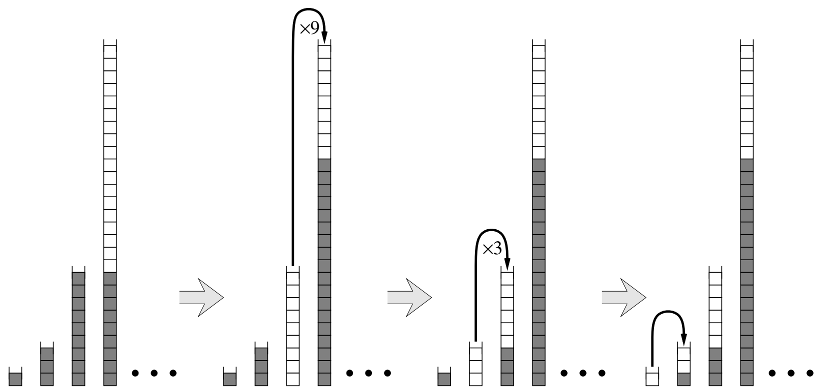
(a) *[2 points]* How many cells are there, as a function of n? Prove your answer is correct.

(b) *[8 points]* In the worst case, exactly how many cells can a horizontal line cross, as a function of n? Prove your answer is correct. Assume that $n = 2^k - 1$ for some integer k. *[For full credit, you must give an exact answer. A tight asymptotic bound (with proof) is worth 5 points. A correct recurrence is worth 3 points.]*

(c) *[5 points extra credit]* In the worst case, how many cells can a *diagonal* line cross?

Incidentally, 'kd-tree' originally meant 'k-dimensional tree'—for example, the specific data structure described here used to be called a '2d-tree'—but current usage ignores this etymology. The phrase 'd-dimensional kd-tree' is now considered perfectly standard, even though it's just as redundant as 'ATM machine', 'PIN number', 'HIV virus', 'PDF format', 'Mt. Fujiyama', 'Sahara Desert', 'The La Brea Tar Pits', or 'and etc.' On the other hand, 'BASIC code' is *not* redundant; 'Beginner's All-Purpose Instruction Code' is a backronym. Hey, aren't you supposed to be taking a test?

3. A *multistack* consists of an infinite series of stacks $S_0, S_1, S_2, \ldots$, where the ith stack S_i can hold up to 3^i elements. Whenever a user attempts to push an element onto any full stack S_i, we first move all the elements in S_i to stack S_{i+1} to make room. But if S_{i+1} is already full, we first move all its members to S_{i+2}, and so on. Moving a single element from one stack to the next takes $O(1)$ time.



Making room for one new element in a multistack.

(a) *[1 point]* In the worst case, how long does it take to push one more element onto a multistack containing n elements?

(b) *[9 points]* Prove that the amortized cost of a push operation is $O(\log n)$, where n is the maximum number of elements in the multistack. You can use any method you like.

4. After graduating with a computer science degree, you find yourself working for a software company that publishes a word processor. The program stores a document containing n characters, grouped into p paragraphs. Your manager asks you to implement a 'Sort Paragraphs' command that rearranges the paragraphs into alphabetical order.

   Design and analyze and efficient paragraph-sorting algorithm, using the following pair of routines as black boxes.

   - COMPAREPARAGRAPHS(i, j) compares the ith and jth paragraphs, and returns i or j depending on which paragraph should come first in the final sorted output. (Don't worry about ties.) This function runs in $O(1)$ time, since almost any two paragraphs can be compared by looking at just their first few characters!
   - MOVEPARAGRAPH(i, j) 'cuts' out the ith paragraph and 'pastes' it back in as the jth paragraph. This function runs in $O(n_i)$ time, where n_i is the number of characters in the ith paragraph. (So in particular, $n_1 + n_2 + \cdots + n_p = n$.)

     Here is an example of MOVEPARAGRAPH$(7, 2)$:

Congress shall make no law respecting...		Congress shall make no law respecting...
No soldier shall, in time of peace...		**A well regulated militia, being...**
The right of the people to be secure...		No soldier shall, in time of peace...
No person shall be held to answer for...		The right of the people to be secure...
In all criminal prosecutions, the...	$\Longrightarrow$	No person shall be held to answer for...
In suits at common law, where the...		In all criminal prosecutions, the...
A well regulated militia, being...		In suits at common law, where the...
Excessive bail shall not be required...		Excessive bail shall not be required...
The enumeration in the Constitution...		The enumeration in the Constitution...
The powers not delegated to the...		The powers not delegated to the...

   [Hint: For full credit, your algorithm should run in $o(n \log n)$ time when $p = o(n)$.]

5. *[1-unit grad students must answer this question.]*

   Describe and analyze an algorithm to randomly shuffle an array of n items, so that each of the $n!$ possible permutations is equally likely. Assume that you have a function RANDOM(i, j) that returns a random integer from the set $\{i, i + 1, \ldots, j\}$ in constant time.

   [Hint: As a sanity check, you might want to confirm that for $n = 3$, all six permutations have probability $1/6$. For full credit, your algorithm must run in $\Theta(n)$ time. A correct algorithm that runs in $\Theta(n \log n)$ time is worth 7 points.]

From: "Josh Pepper" <jwpepper@uiuc.edu>
To: "Chris Neihengen" <neihenge@uiuc.edu>
Subject: FW: proof
Date: Fri, 29 Sep 2000 09:34:56 -0500

thought you might like this.

Problem: To prove that computer science 373 is indeed the work of Satan.

Proof: First, let us assume that everything in "Helping Yourself with Numerology", by Helyn Hitchcock, is true.

Second, let us apply divide and conquer to this problem. There are main parts:

1. The name of the course: "Combinatorial Algorithms"
2. The most important individual in the course, the "Recursion Fairy"
3. The number of this course: 373.

We examine these sequentially.

The name of the course. "Combinatorial Algorithms" can actually be expressed as a single integer - 23 - since it has 23 letters. The most important individual, the Recursion Fairy, can also be expressed as a single integer - 14 - since it has 14 letters. In other words:

COMBINATORIAL ALGORITHMS = 23
RECURSION FAIRY = 14

As a side note, a much shorter proof has already been published showing that the Recursion Fairy is Lucifer, and that any class involving the Fairy is from Lucifer, however, that proofs numerological significance is slight.

Now we can move on to an analysis of the number of course, which holds great meaning. The first assumtion we make is that the number of the course, 373, is not actually a base 10 number. We can prove this inductively by making a reasonable guess for the actual base, then finding a new way to express the nature of the course, and if the answer

confirms what we assumed, then we're right. That's the way induction works.

What is a reasonable guess for the base of the course? The answer is trivial, since the basest of all beings is the Recursion Fairy, the base is 14. So a true base 10 representation of 373 (base 14) is 689. So we see:

373 (base 14) = 689 (base 10)

Now since the nature of the course has absolutely nothing to do with combinatorial algorithms (instead having much to do with the work of the devil), we can subtract from the above result everything having to do with combinatorial algorithms just by subtracting 23. Here we see that:

689 - 23 = 666

QED.

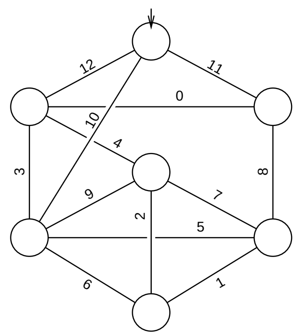
1. Using any method you like, compute the following subgraphs for the weighted graph below. Each subproblem is worth 3 points. Each incorrect edge costs you 1 point, but you cannot get a negative score for any subproblem.

   (a) a depth-first search tree, starting at the top vertex;

   (b) a breadth-first search tree, starting at the top vertex;

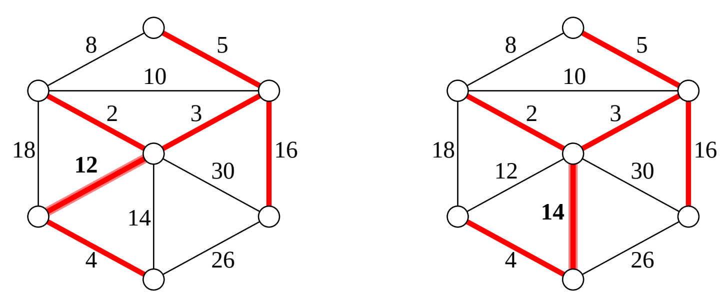
   (c) a shortest path tree, starting at the top vertex;

   (d) the minimum spanning tree.



2. Suppose you are given a weighted undirected graph G (represented as an adjacency list) and its minimum spanning tree T (which you already know how to compute). Describe and analyze and algorithm to find the *second-minimum spanning tree* of G, *i.e.*, the spanning tree of G with smallest total weight except for T.

   The minimum spanning tree and the second-minimum spanning tree differ by exactly one edge. But *which* edge is different, and *how* is it different? That's what your algorithm has to figure out!



The minimum spanning tree and the second-minimum spanning tree of a graph.

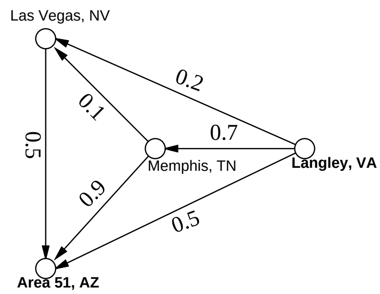
3. (a) *[4 pts]* Prove that a connected acyclic graph with V vertices has exactly $V - 1$ edges. ("It's a tree!" is not a proof.)

   (b) *[4 pts]* Describe and analyze an algorithm that determines whether a given graph is a tree, where the graph is represented by an adjacency list.

   (c) *[2 pts]* What is the running time of your algorithm from part (b) if the graph is represented by an adjacency matrix?

4. Mulder and Scully have computed, for every road in the United States, the exact probability that someone driving on that road *won't* be abducted by aliens. Agent Mulder needs to drive from Langley, Virginia to Area 51, Nevada. What route should he take so that he has the least chance of being abducted?

   More formally, you are given a directed graph $G = (V, E)$, where every edge e has an independent safety probability $p(e)$. The *safety* of a path is the product of the safety probabilities of its edges. Design and analyze an algorithm to determine the safest path from a given start vertex s to a given target vertex t.



   With the probabilities shown above, if Mulder tries to drive directly from Langley to Area 51, he has a 50% chance of getting there without being abducted. If he stops in Memphis, he has a $0.7 \times 0.9 = 63\%$ chance of arriving safely. If he stops first in Memphis and then in Las Vegas, he has a $1 - 0.7 \times 0.1 \times 0.5 = 96.5\%$ chance of being abducted![1]

5. *[1-unit grad students must answer this question.]*

   Many string matching applications allow the following *wild card* characters in the pattern.

   - The wild card `?` represents an arbitrary single character. For example, the pattern `s?r?ng` matches the strings `string`, `sprung`, and `sarong`.
   - The wild card `*` represents an arbitrary string of zero or more characters. For example, the pattern `te*st*` matches the strings `test`, `tensest`, and `technostructuralism`.

   Both wild cards can occur in a single pattern. For example, the pattern `f*a??` matches the strings `face`, `football`, and `flippityfloppitydingdongdang`. On the other hand, neither wild card can occur in the text.

   Describe how to modify the Knuth-Morris-Pratt algorithm to support patterns with these wild cards, and analyze the modified algorithm. Your algorithm should find the first substring in the text that matches the pattern. An algorithm that supports only one of the two wild cards is worth 5 points.

[1]That's how they got Elvis, you know.

1. **True, False, or Maybe**

   Indicate whether each of the following statments is always true, sometimes true, always false, or unknown. Some of these questions are deliberately tricky, so read them carefully. Each correct choice is worth $+1$, and each incorrect choice is worth -1. **Guessing will hurt you!**

   (a) Suppose SmartAlgorithm runs in $\Theta(n^2)$ time and DumbAlgorithm runs in $\Theta(2^n)$ time for all inputs of size n. (Thus, for each algorithm, the best-case and worst-case running times are the same.) SmartAlgorithm is faster than DumbAlgorithm.

   ☐ True ☐ False ☐ Sometimes ☐ Nobody Knows

   (b) QuickSort runs in $O(n^6)$ time.

   ☐ True ☐ False ☐ Sometimes ☐ Nobody Knows

   (c) $\lfloor \log_2 n \rfloor \geq \lceil \log_2 n \rceil$

   ☐ True ☐ False ☐ Sometimes ☐ Nobody Knows

   (d) The recurrence $F(n) = n + 2\sqrt{n} \cdot F(\sqrt{n})$ has the solution $F(n) = \Theta(n \log n)$.

   ☐ True ☐ False ☐ Sometimes ☐ Nobody Knows

   (e) A Fibonacci heap with n nodes has depth $\Omega(\log n)$.

   ☐ True ☐ False ☐ Sometimes ☐ Nobody Knows

   (f) Suppose a graph G is represented by an adjacency matrix. It is possible to determine whether G is an independent set without looking at every entry of the adjacency matrix.

   ☐ True ☐ False ☐ Sometimes ☐ Nobody Knows

   (g) NP $\neq$ co-NP

   ☐ True ☐ False ☐ Sometimes ☐ Nobody Knows

   (h) Finding the smallest clique in a graph is NP-hard.

   ☐ True ☐ False ☐ Sometimes ☐ Nobody Knows

   (i) A polynomial-time reduction from X to 3SAT proves that X is NP-hard.

   ☐ True ☐ False ☐ Sometimes ☐ Nobody Knows

   (j) The correct answer for exactly three of these questions is "False".

   ☐ True ☐ False

2. **Convex Hull**

   Suppose you are given the convex hull of a set of n points, and one additional point (x, y). The convex hull is represented by an array of vertices in counterclockwise order, starting from the leftmost vertex. Describe how to test in $O(\log n)$ time whether or not the additional point (x, y) is inside the convex hull.

3. **Finding the Largest Block**

   In your new job, you are working with screen images. These are represented using two dimensional arrays where each element is a 1 or a 0, indicating whether that position of the screen is illuminated. Design and analyze an efficient algorithm to find the largest rectangular block of ones in such an array. For example, the largest rectangular block of ones in the array shown below is in rows 2–4 and columns 2–3. *[Hint: Use dynamic programming.]*

1	0	1	0	0
1	1	1	0	1
0	1	1	1	1
1	1	1	0	0

4. **The Hogwarts Sorting Hat**

   Every year, upon their arrival at Hogwarts School of Witchcraft and Wizardry, new students are sorted into one of four houses (Gryffindor, Hufflepuff, Ravenclaw, or Slytherin) by the Hogwarts Sorting Hat. The student puts the Hat on their head, and the Hat tells the student which house they will join. This year, a failed experiment by Fred and George Weasley filled almost all of Hogwarts with sticky brown goo, mere moments before the annual Sorting. As a result, the Sorting had to take place in the basement hallways, where there was so little room to move that the students had to stand in a long line.

   After everyone learned what house they were in, the students tried to group together by house, but there was too little room in the hallway for more than one student to move at a time. Fortunately, the Sorting Hat took CS 373 many years ago, so it knew how to group the students as quickly as possible. What method did the Sorting Hat use?

   More formally, you are given an array of n items, where each item has one of four possible values, possibly with a pointer to some additional data. Design and analyze an algorithm that rearranges the items into four clusters in $O(n)$ time using only $O(1)$ extra space.

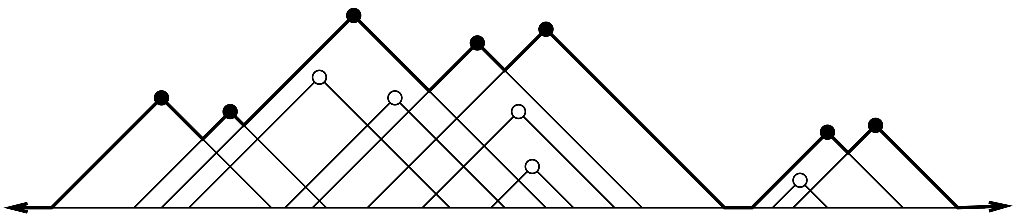
G	H	R	R	G	G	R	G	H	H	R	S	R	R	H	G	S	H	G	G
Harry	Ann	Bob	Tina	Chad	Bill	Lisa	Ekta	Bart	Jim	John	Jeff	Liz	Mary	Dawn	Nick	Kim	Fox	Dana	Mel

$\Downarrow$

G	G	G	G	G	G	G	H	H	H	H	H	R	R	R	R	R	R	S	S
Harry	Ekta	Bill	Chad	Nick	Mel	Dana	Fox	Ann	Jim	Dawn	Bart	Lisa	Tina	John	Bob	Liz	Mary	Kim	Jeff

5. **The Egyptian Skyline**

Suppose you are given a set of n pyramids in the plane. Each pyramid is a isosceles triangle with two $45°$ edges and a horizontal edge on the x-axis. Each pyramid is represented by the x- and y-coordinates of its topmost point. Your task is to compute the "skyline" formed by these pyramids (the dark line shown below).



The skyline formed by these 12 pyramids has 16 vertices.

(a) Describe and analyze an algorithm that determines which pyramids are visible on the skyline. These are the pyramids with black points in the figure above; the pyramids with white points are not visible. *[Hint: You've seen this problem before.]*

(b) One you know which pyramids are visible, how would you compute the *shape* of the skyline? Describe and analyze an algorithm to compute the left-to-right sequence of skyline vertices, including the vertices between the pyramids and on the ground.

6. **DNF-SAT**

A boolean formula is in *disjunctive normal form* (DNF) if it consists of clauses of conjunctions (ANDs) joined together by disjunctions (ORs). For example, the formula

$$(\overline{a} \wedge b \wedge \overline{c}) \vee (b \wedge c) \vee (a \wedge \overline{b} \wedge \overline{c})$$

is in disjunctive normal form. DNF-SAT is the problem that asks, given a boolean formula in disjunctive normal form, whether that formula is satisfiable.

(a) Show that DNF-SAT is in P.

(b) What is wrong with the following argument that P=NP?

> Suppose we are given a boolean formula in conjunctive normal form with at most three literals per clause, and we want to know if it is satisfiable. We can use the distributive law to construct an equivalent formula in disjunctive normal form. For example,
>
> $$(a \vee b \vee \overline{c}) \wedge (\overline{a} \vee \overline{b}) \iff (a \wedge \overline{b}) \vee (b \wedge \overline{a}) \vee (\overline{c} \wedge \overline{a}) \vee (\overline{c} \wedge \overline{b})$$
>
> Now we can use the answer to part (a) to determine, in polynomial time, whether the resulting DNF formula is satisfiable. We have just solved 3SAT in polynomial time! Since 3SAT is NP-hard, we must conclude that P=NP.

7. **Magic 3-Coloring** *[1-unit graduate students must answer this question.]*

The recursion fairy's distant cousin, the reduction genie, shows up one day with a magical gift for you—a box that determines in constant time whether or not a graph is 3-colorable. (A graph is 3-colorable if you can color each of the vertices red, green, or blue, so that every edge has do different colors.) The magic box does not tell you *how* to color the graph, just whether or not it can be done. Devise and analyze an algorithm to 3-color any graph **in polynomial time** using this magic box.

CS 373: Combinatorial Algorithms, Spring 2001
Homework 0, due January 23, 2001 at the beginning of class

Name:	
Net ID:	Alias:

Neatly print your name (first name first, with no comma), your network ID, and a short alias into the boxes above. **Do not sign your name. Do not write your Social Security number.** Staple this sheet of paper to the top of your homework.

Grades will be listed on the course web site by alias give us, so your alias should not resemble your name or your Net ID. If you don't give yourself an alias, we'll give you one that you won't like.

This homework tests your familiarity with the prerequisite material from CS 173, CS 225, and CS 273—many of these problems have appeared on homeworks or exams in those classes—primarily to help you identify gaps in your knowledge. **You are responsible for filling those gaps on your own.** Parberry and Chapters 1–6 of CLR should be sufficient review, but you may want to consult other texts as well.

Before you do anything else, read the Homework Instructions and FAQ on the CS 373 course web page (http://www-courses.cs.uiuc.edu/~cs373/hw/faq.html), and then check the box below. This web page gives instructions on how to write and submit homeworks—staple your solutions together in order, write your name and netID on every page, don't turn in source code, analyze everything, use good English and good logic, and so forth.

☐ I have read the CS 373 Homework Instructions and FAQ.

Required Problems

1. (a) Prove that any positive integer can be written as the sum of distinct powers of 2. For example: $42 = 2^5 + 2^3 + 2^1$, $25 = 2^4 + 2^3 + 2^0$, $17 = 2^4 + 2^0$. *[Hint: 'Write the number in binary' is* not *a proof; it just restates the problem.]*

   (b) Prove that any positive integer can be written as the sum of distinct *nonconsecutive* Fibonacci numbers—if F_n appears in the sum, then neither F_{n+1} nor F_{n-1} will. For example: $42 = F_9 + F_6$, $25 = F_8 + F_4 + F_2$, $17 = F_7 + F_4 + F_2$.

   (c) Prove that *any* integer (positive, negative, or zero) can be written in the form $\sum_i \pm 3^i$, where the exponents i are distinct non-negative integers. For example: $42 = 3^4 - 3^3 - 3^2 - 3^1$, $25 = 3^3 - 3^1 + 3^0$, $17 = 3^3 - 3^2 - 3^0$.

2. Sort the following 20 functions from asymptotically smallest to asymptotically largest, indicating ties if there are any. You do not need to turn in proofs (in fact, please *don't* turn in proofs), but you should do them anyway just for practice.

1	n	n^2	$\lg n$	$\lg^* n$
$2^{2^{\lg\lg n+1}}$	$\lg^* 2^n$	$2^{\lg^* n}$	$\lfloor \lg(n!) \rfloor$	$\lfloor \lg n \rfloor!$
$n^{\lg n}$	$(\lg n)^n$	$(\lg n)^{\lg n}$	$n^{1/\lg n}$	$n^{\lg\lg n}$
$\log_{1000} n$	$\lg^{1000} n$	$\lg^{(1000)} n$	$\left(1+\frac{1}{1000}\right)^n$	$n^{1/1000}$

To simplify notation, write $f(n) \ll g(n)$ to mean $f(n) = o(g(n))$ and $f(n) \equiv g(n)$ to mean $f(n) = \Theta(g(n))$. For example, the functions n^2, n, $\binom{n}{2}$, n^3 could be sorted either as $n \ll n^2 \equiv \binom{n}{2} \ll n^3$ or as $n \ll \binom{n}{2} \equiv n^2 \ll n^3$.

3. Solve the following recurrences. State tight asymptotic bounds for each function in the form $\Theta(f(n))$ for some recognizable function $f(n)$. You do not need to turn in proofs (in fact, please *don't* turn in proofs), but you should do them anyway just for practice. Assume reasonable but nontrivial base cases if none are supplied. Extra credit will be given for more exact solutions.

   (a) $A(n) = 5A(n/3) + n \log n$

   (b) $B(n) = \min\limits_{0<k<n} \big(B(k) + B(n-k) + 1\big)$.

   (c) $C(n) = 4C(\lfloor n/2 \rfloor + 5) + n^2$

   (d) $D(n) = D(n-1) + 1/n$

   *(e) $E(n) = n + 2\sqrt{n} \cdot E(\sqrt{n})$

4. This problem asks you to simplify some recursively defined boolean formulas as much as possible. In each case, prove that your answer is correct. Each proof can be just a few sentences long, but it must be a *proof.*

   (a) Suppose $\alpha_0 = p$, $\alpha_1 = q$, and $\alpha_n = (\alpha_{n-2} \wedge \alpha_{n-1})$ for all $n \geq 2$. Simplify α_n as much as possible. *[Hint: What is α_5?]*

   (b) Suppose $\beta_0 = p$, $\beta_1 = q$, and $\beta_n = (\beta_{n-2} \Leftrightarrow \beta_{n-1})$ for all $n \geq 2$. Simplify β_n as much as possible. *[Hint: What is β_5?]*

   (c) Suppose $\gamma_0 = p$, $\gamma_1 = q$, and $\gamma_n = (\gamma_{n-2} \Rightarrow \gamma_{n-1})$ for all $n \geq 2$. Simplify γ_n as much as possible. *[Hint: What is γ_5?]*

   (d) Suppose $\delta_0 = p$, $\delta_1 = q$, and $\delta_n = (\delta_{n-2} \bowtie \delta_{n-1})$ for all $n \geq 2$, where $\bowtie$ is some boolean function with two arguments. Find a boolean function $\bowtie$ such that $\delta_n = \delta_m$ if and only if $n - m$ is a multiple of 4. *[Hint: There is only one such function.]*

5. Every year, upon their arrival at Hogwarts School of Witchcraft and Wizardry, new students are sorted into one of four houses (Gryffindor, Hufflepuff, Ravenclaw, or Slytherin) by the Hogwarts Sorting Hat. The student puts the Hat on their head, and the Hat tells the student which house they will join. This year, a failed experiment by Fred and George Weasley filled almost all of Hogwarts with sticky brown goo, mere moments before the annual Sorting. As a result, the Sorting had to take place in the basement hallways, where there was so little room to move that the students had to stand in a long line.

   After everyone learned what house they were in, the students tried to group together by house, but there was too little room in the hallway for more than one student to move at a time. Fortunately, the Sorting Hat took CS 373 many years ago, so it knew how to group the students as quickly as possible. What method did the Sorting Hat use?

   More formally, you are given an array of n items, where each item has one of four possible values, possibly with a pointer to some additional data. Describe an algorithm[1] that rearranges the items into four clusters in $O(n)$ time using only $O(1)$ extra space.

G	H	R	R	G	G	R	G	H	H	R	S	R	R	H	G	S	H	G	G
Harry	Ann	Bob	Tina	Chad	Bill	Lisa	Ekta	Bart	Jim	John	Jeff	Liz	Mary	Dawn	Nick	Kim	Fox	Dana	Mel

$$\Downarrow$$

G	G	G	G	G	G	G	H	H	H	H	H	R	R	R	R	R	R	S	S
Harry	Ekta	Bill	Chad	Nick	Mel	Dana	Fox	Ann	Jim	Dawn	Bart	Lisa	Tina	John	Bob	Liz	Mary	Kim	Jeff

6. *[This problem is required only for graduate students taking CS 373 for a full unit; anyone else can submit a solution for extra credit.]*

   Penn and Teller have a special deck of fifty-two cards, with no face cards and nothing but clubs—the ace, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, ..., 52 of clubs. (They're big cards.) Penn shuffles the deck until each each of the 52! possible orderings of the cards is equally likely. He then takes cards one at a time from the top of the deck and gives them to Teller, stopping as soon as he gives Teller the three of clubs.

   (a) On average, how many cards does Penn give Teller?

   (b) On average, what is the smallest-numbered card that Penn gives Teller?

   *(c) On average, what is the largest-numbered card that Penn gives Teller?

   [Hint: Solve for an n-card deck and then set $n = 52$.] In each case, give *exact* answers and prove that they are correct. If you have to appeal to "intuition" or "common sense", your answers are probably wrong!

[1] Since you've read the Homework Instructions, you know what the phrase 'describe an algorithm' means. Right?

Practice Problems

The remaining problems are entirely for your benefit; similar questions will appear in every homework. Don't turn in solutions—we'll just throw them out—but feel free to ask us about practice questions during office hours and review sessions. Think of them as potential exam questions (hint, hint). We'll post solutions to *some* of the practice problems after the homeworks are due.

1. Recall the standard recursive definition of the Fibonacci numbers: $F_0 = 0$, $F_1 = 1$, and $F_n = F_{n-1} + F_{n-2}$ for all $n \geq 2$. Prove the following identities for all positive integers n and m.

   (a) F_n is even if and only if n is divisible by 3.

   (b) $\sum_{i=0}^{n} F_i = F_{n+2} - 1$

   (c) $F_n^2 - F_{n+1}F_{n-1} = (-1)^{n+1}$

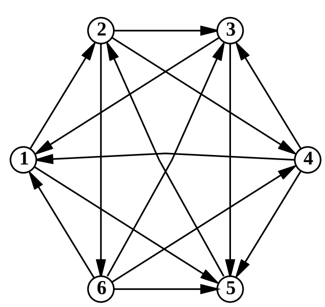
   ★(d) If n is an integer multiple of m, then F_n is an integer multiple of F_m.

2. (a) Prove the following identity by induction:

$$\binom{2n}{n} = \sum_{k=0}^{n} \binom{n}{k}\binom{n}{n-k}.$$

   (b) Give a non-inductive combinatorial proof of the same identity, by showing that the two sides of the equation count exactly the same thing in two different ways. There is a correct one-sentence proof.

3. A *tournament* is a directed graph with exactly one edge between every pair of vertices. (Think of the nodes as players in a round-robin tournament, where each edge points from the winner to the loser.) A *Hamiltonian path* is a sequence of directed edges, joined end to end, that visits every vertex exactly once. Prove that every tournament contains at least one Hamiltonian path.



A six-vertex tournament containing the Hamiltonian path $6 \rightarrow 4 \rightarrow 5 \rightarrow 2 \rightarrow 3 \rightarrow 1$.

4. Solve the following recurrences. State tight asymptotic bounds for each function in the form $\Theta(f(n))$ for some recognizable function $f(n)$. You do not need to turn in proofs (in fact, please *don't* turn in proofs), but you should do them anyway just for practice. Assume reasonable but nontrivial base cases if none are supplied. Extra credit will be given for more exact solutions.

   (a) $A(n) = A(n/2) + n$

   (b) $B(n) = 2B(n/2) + n$

   ★(c) $C(n) = n + \frac{1}{2}\big(C(n-1) + C(3n/4)\big)$

   (d) $D(n) = \max\limits_{n/3<k<2n/3} \big(D(k) + D(n-k) + n\big)$

   ⋆(e) $E(n) = 2E(n/2) + n/\lg n$

   ⋆(f) $F(n) = \dfrac{F(n-1)}{F(n-2)}$, where $F(1) = 1$ and $F(2) = 2$.

   ⋆(g) $G(n) = G(n/2) + G(n/4) + G(n/6) + G(n/12) + n$ [Hint: $\frac{1}{2} + \frac{1}{4} + \frac{1}{6} + \frac{1}{12} = 1$.]

   ⋆(h) $H(n) = n + \sqrt{n} \cdot H(\sqrt{n})$

   ⋆(i) $I(n) = (n-1)(I(n-1) + I(n-2))$, where $F(0) = F(1) = 1$

   ⋆(j) $J(n) = 8J(n-1) - 15J(n-2) + 1$

5. (a) Prove that $2^{\lceil \lg n \rceil + \lfloor \lg n \rfloor} = \Theta(n^2)$.

   (b) Prove or disprove: $2^{\lfloor \lg n \rfloor} = \Theta\big(2^{\lceil \lg n \rceil}\big)$.

   (c) Prove or disprove: $2^{2^{\lfloor \lg \lg n \rfloor}} = \Theta\big(2^{2^{\lceil \lg \lg n \rceil}}\big)$.

   (d) Prove or disprove: If $f(n) = O(g(n))$, then $\log(f(n)) = O(\log(g(n)))$.

   (e) Prove or disprove: If $f(n) = O(g(n))$, then $2^{f(n)} = O(2^{g(n)})$.

   ⋆(f) Prove that $\log^k n = o(n^{1/k})$ for any positive integer k.

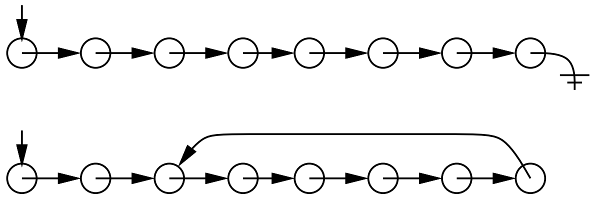
6. Evaluate the following summations; simplify your answers as much as possible. Significant partial credit will be given for answers in the form $\Theta(f(n))$ for some recognizable function $f(n)$.

   (a) $\displaystyle\sum_{i=1}^{n} \sum_{j=1}^{i} \sum_{k=j}^{i} \frac{1}{i}$

   ⋆(b) $\displaystyle\sum_{i=1}^{n} \sum_{j=1}^{i} \sum_{k=j}^{i} \frac{1}{j}$

   (c) $\displaystyle\sum_{i=1}^{n} \sum_{j=1}^{i} \sum_{k=j}^{i} \frac{1}{k}$

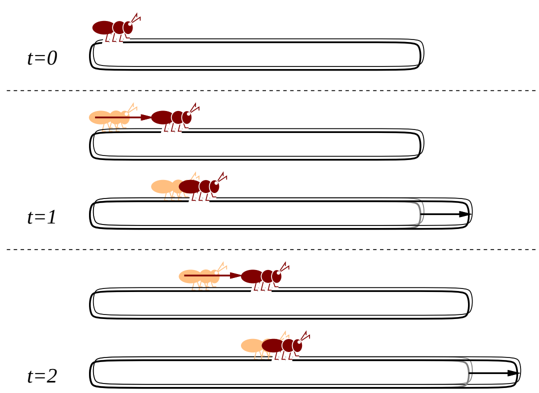
7. Suppose you have a pointer to the head of singly linked list. Normally, each node in the list only has a pointer to the next element, and the last node's pointer is NULL. Unfortunately, your list might have been corrupted by a bug in somebody else's code[2], so that the last node has a pointer back to some other node in the list instead.



Top: A standard linked list. Bottom: A corrupted linked list.

Describe an algorithm that determines whether the linked list is corrupted or not. Your algorithm must not modify the list. For full credit, your algorithm should run in $O(n)$ time, where n is the number of nodes in the list, and use $O(1)$ extra space (not counting the list itself).

*8. An ant is walking along a rubber band, starting at the left end. Once every second, the ant walks one inch to the right, and then you make the rubber band one inch longer by pulling on the right end. The rubber band stretches uniformly, so stretching the rubber band also pulls the ant to the right. The initial length of the rubber band is n inches, so after t seconds, the rubber band is $n + t$ inches long.



Every second, the ant walks an inch, and then the rubber band is stretched an inch longer.

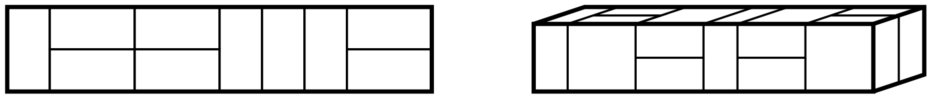
(a) How far has the ant moved after t seconds, as a function of n and t? Set up a recurrence and (for full credit) give an *exact* closed-form solution. [Hint: What *fraction* of the rubber band's length has the ant walked?]

(b) How long does it take the ant to get to the right end of the rubber band? For full credit, give an answer of the form $f(n) + \Theta(1)$ for some explicit function $f(n)$.

9. (a) A *domino* is a 2×1 or 1×2 rectangle. How many different ways are there to completely fill a $2 \times n$ rectangle with n dominos? Set up a recurrence relation and give an *exact* closed-form solution.

[2] After all, *your* code is always completely 100% bug-free. Isn't that right, Mr. Gates?

(b) A *slab* is a three-dimensional box with dimensions $1 \times 2 \times 2$, $2 \times 1 \times 2$, or $2 \times 2 \times 1$. How many different ways are there to fill a $2 \times 2 \times n$ box with n slabs? Set up a recurrence relation and give an *exact* closed-form solution.



A 2×10 rectangle filled with ten dominos, and a $2 \times 2 \times 10$ box filled with ten slabs.

10. Professor George O'Jungle has a favorite 26-node binary tree, whose nodes are labeled by letters of the alphabet. The preorder and postorder sequences of nodes are as follows:

    preorder: `M N H C R S K W T G D X I Y A J P O E Z V B U L Q F`
    postorder: `C W T K S G R H D N A O E P J Y Z I B Q L F U V X M`

    Draw Professor O'Jungle's binary tree, and give the inorder sequence of nodes.

11. Alice and Bob each have a fair n-sided die. Alice rolls her die once. Bob then repeatedly throws his die until he rolls a number at least as big as the number Alice rolled. Each time Bob rolls, he pays Alice \$1. (For example, if Alice rolls a 5, and Bob rolls a 4, then a 3, then a 1, then a 5, the game ends and Alice gets \$4. If Alice rolls a 1, then no matter what Bob rolls, the game will end immediately, and Alice will get \$1.)

    Exactly how much money does Alice expect to win at this game? Prove that your answer is correct. If you have to appeal to 'intuition' or 'common sense', your answer is probably wrong!

12. Prove that for any nonnegative parameters a and b, the following algorithms terminate and produce identical output.

```
SlowEuclid(a, b):
  if b > a
      return SlowEuclid(b, a)
  else if b = 0
      return a
  else
      return SlowEuclid(b, a − b)
```

```
FastEuclid(a, b):
  if b = 0
      return a
  else
      return FastEuclid(b, a mod b)
```

CS 373: Combinatorial Algorithms, Spring 2001
Homework 1 (due Thursday, February 1, 2001 at 11:59:59 p.m.)

Name:		
Net ID:	Alias:	U $^3\!/_4$ 1

Name:		
Net ID:	Alias:	U $^3\!/_4$ 1

Name:		
Net ID:	Alias:	U $^3\!/_4$ 1

Starting with Homework 1, homeworks may be done in teams of up to three people. Each team turns in just one solution, and every member of a team gets the same grade. Since 1-unit graduate students are required to solve problems that are worth extra credit for other students, **1-unit grad students may not be on the same team as 3/4-unit grad students or undergraduates.**

Neatly print your name(s), NetID(s), and the alias(es) you used for Homework 0 in the boxes above. Please also tell us whether you are an undergraduate, 3/4-unit grad student, or 1-unit grad student by circling U, $^3\!/_4$, or 1, respectively. Staple this sheet to the top of your homework.

Required Problems

1. Suppose you are a simple shopkeeper living in a country with n different types of coins, with values $1 = c[1] < c[2] < \cdots < c[n]$. (In the U.S., for example, $n = 6$ and the values are $1, 5, 10, 25, 50$ and 100 cents.) Your beloved and belevolent dictator, El Generalissimo, has decreed that whenever you give a customer change, you must use the smallest possible number of coins, so as not to wear out the image of El Generalissimo lovingly engraved on each coin by servants of the Royal Treasury.

   (a) In the United States, there is a simple greedy algorithm that always results in the smallest number of coins: subtract the largest coin and recursively give change for the remainder. El Generalissimo does not approve of American capitalist greed. Show that there is a set of coin values for which the greedy algorithm does *not* always give the smallest possible of coins.

   (b) Describe and analyze a dynamic programming algorithm to determine, given a target amount A and a sorted array $c[1..n]$ of coin values, the smallest number of coins needed to make A cents in change. You can assume that $c[1] = 1$, so that it is possible to make change for any amount A.

2. Consider the following sorting algorithm:

```
STUPIDSORT(A[0 .. n − 1]) :
  if n = 2 and A[0] > A[1]
      swap A[0] ↔ A[1]
  else if n > 2
      m ← ⌈2n/3⌉
      STUPIDSORT(A[0 .. m − 1])
      STUPIDSORT(A[n − m .. n − 1])
      STUPIDSORT(A[0 .. m − 1])
```

   (a) Prove that StupidSort actually sorts its input.

   (b) Would the algorithm still sort correctly if we replaced the line $m \leftarrow \lceil 2n/3 \rceil$ with $m \leftarrow \lfloor 2n/3 \rfloor$? Justify your answer.

   (c) State a recurrence (including the base case(s)) for the number of comparisons executed by StupidSort.

   (d) Solve the recurrence, and prove that your solution is correct. [Hint: Ignore the ceiling.] Does the algorithm deserve its name?

   *(e) Show that the number of *swaps* executed by StupidSort is at most $\binom{n}{2}$.

3. The following randomized algorithm selects the rth smallest element in an unsorted array $A[1 .. n]$. For example, to find the smallest element, you would call RandomSelect$(A, 1)$; to find the median element, you would call RandomSelect$(A, \lfloor n/2 \rfloor)$. Recall from lecture that Partition splits the array into three parts by comparing the pivot element $A[p]$ to every other element of the array, using $n - 1$ comparisons altogether, and returns the new index of the pivot element.

```
RANDOMSELECT(A[1 .. n], r) :
  p ← RANDOM(1, n)
  k ← PARTITION(A[1 .. n], p)
  if r < k
      return RANDOMSELECT(A[1 .. k − 1], r)
  else if r > k
      return RANDOMSELECT(A[k + 1 .. n], r − k)
  else
      return A[k]
```

   (a) State a recurrence for the expected running time of RandomSelect, as a function of n and r.

   (b) What is the *exact* probability that RandomSelect compares the ith smallest and jth smallest elements in the input array? The correct answer is a simple function of i, j, and r. [Hint: Check your answer by trying a few small examples.]

   (c) Show that for any n and r, the expected running time of RandomSelect is $\Theta(n)$. You can use either the recurrence from part (a) or the probabilities from part (b). For extra credit, find the exact expected number of comparisons, as a function of n and r.

   (d) What is the expected number of times that RandomSelect calls itself recursively?

4. What excitement! The Champaign Spinners and the Urbana Dreamweavers have advanced to meet each other in the World Series of Basketweaving! The World Champions will be decided by a best-of- $2n-1$ series of head-to-head weaving matches, and the first to win n matches will take home the coveted Golden Basket (for example, a best-of-7 series requiring four match wins, but we will keep the generalized case). We know that for any given match there is a constant probability p that Champaign will win, and a subsequent probability $q = 1 - p$ that Urbana will win.

   Let $P(i, j)$ be the probability that Champaign will win the series given that they still need i more victories, whereas Urbana needs j more victories for the championship. $P(0, j) = 1$, $1 \leq j \leq n$, because Champaign needs no more victories to win. $P(i, 0) = 0$, $1 \leq i \leq n$, as Champaign cannot possibly win if Urbana already has. $P(0, 0)$ is meaningless. Champaign wins any particular match with probability p and loses with probability q, so

$$P(i, j) = p \cdot P(i - 1, j) + q \cdot P(i, j - 1)$$

   for any $i \geq 1$ and $j \geq 1$.

   Create and analyze an $O(n^2)$-time dynamic programming algorithm that takes the parameters n, p and q and returns the probability that Champaign will win the series (that is, calculate $P(n, n)$).

5. The traditional Devonian/Cornish drinking song "The Barley Mow" has the following pseudolyrics[1], where *container*$[i]$ is the name of a container that holds 2^i ounces of beer.[2]

```
BarleyMow(n):
  "Here's a health to the barley-mow, my brave boys,"
  "Here's a health to the barley-mow!"

  "We'll drink it out of the jolly brown bowl,"
  "Here's a health to the barley-mow!"
  "Here's a health to the barley-mow, my brave boys,"
  "Here's a health to the barley-mow!"

  for i ← 1 to n
      "We'll drink it out of the container[i], boys,"
      "Here's a health to the barley-mow!"
      for j ← i downto 1
          "The container[j],"
      "And the jolly brown bowl!"
      "Here's a health to the barley-mow!"
      "Here's a health to the barley-mow, my brave boys,"
      "Here's a health to the barley-mow!"
```

   (a) Suppose each container name *container*$[i]$ is a single word, and you can sing four words a second. How long would it take you to sing BarleyMow(n)? (Give a tight asymptotic bound.)

   (b) If you want to sing this song for $n > 20$, you'll have to make up your own container names, and to avoid repetition, these names will get progressively longer as n increases[3]. Suppose *container*$[n]$ has $\Theta(\log n)$ syllables, and you can sing six syllables per second. Now how long would it take you to sing BarleyMow(n)? (Give a tight asymptotic bound.)

   (c) Suppose each time you mention the name of a container, you drink the corresponding amount of beer: one ounce for the jolly brown bowl, and 2^i ounces for each *container*$[i]$. Assuming for purposes of this problem that you are at least 21 years old, *exactly* how many ounces of beer would you drink if you sang BarleyMow(n)? (Give an *exact* answer, not just an asymptotic bound.)

[1] Pseudolyrics are to lyrics as pseudocode is to code.

[2] One version of the song uses the following containers: nipperkin, gill pot, half-pint, pint, quart, pottle, gallon, half-anker, anker, firkin, half-barrel, barrel, hogshead, pipe, well, river, and ocean. Every container in this list is twice as big as its predecessor, except that a firkin is actually 2.25 ankers, and the last three units are just silly.

[3] "We'll drink it out of the hemisemidemiyottapint, boys!"

6. *[This problem is required only for graduate students taking CS 373 for a full unit; anyone else can submit a solution for extra credit.]*

   Suppose we want to display a paragraph of text on a computer screen. The text consists of n words, where the ith word is p_i pixels wide. We want to break the paragraph into several lines, each exactly P pixels long. Depending on which words we put on each line, we will need to insert different amounts of white space between the words. The paragraph should be fully justified, meaning that the first word on each line starts at its leftmost pixel, and *except for the last line*, the last character on each line ends at its rightmost pixel. There must be at least one pixel of whitespace between any two words on the same line.

   Define the *slop* of a paragraph layout as the sum over all lines, *except the last*, of the cube of the number of extra white-space pixels in each line (not counting the one pixel required between every adjacent pair of words). Specifically, if a line contains words i through j, then the amount of extra white space on that line is $P - j + i - \sum_{k=i}^{j} p_k$. Describe a dynamic programming algorithm to print the paragraph with minimum slop.

Practice Problems

1. Give an $O(n^2)$ algorithm to find the longest increasing subsequence of a sequence of numbers. The elements of the subsequence need not be adjacent in the sequence. For example, the sequence $\langle 1, 5, 3, 2, 4 \rangle$ has longest increasing subsequence $\langle 1, 3, 4 \rangle$.

2. You are at a political convention with n delegates. Each delegate is a member of exactly one political party. It is impossible to tell which political party a delegate belongs to. However, you can check whether any two delegates are in the *same* party or not by introducing them to each other. (Members of the same party always greet each other with smiles and friendly handshakes; members of different parties always greet each other with angry stares and insults.)

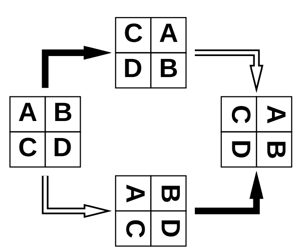
   (a) Suppose a majority (more than half) of the delegates are from the same political party. Give an efficient algorithm that identifies a member of the majority party.

   (b) Suppose exactly k political parties are represented at the convention and one party has a *plurality*: more delegates belong to that party than to any other. Present a practical procedure to pick a person from the plurality party as parsimoniously as possible. (Please.)

3. Give an algorithm that finds the *second* smallest of n elements in at most $n + \lceil \lg n \rceil - 2$ comparisons. [Hint: divide and conquer to find the smallest; where is the second smallest?]

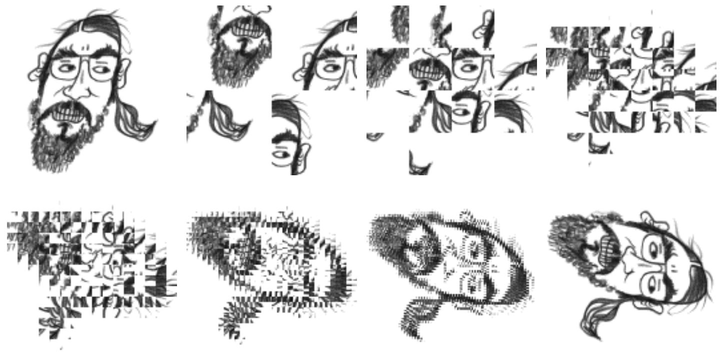
4. Some graphics hardware includes support for an operation called *blit*, or **bl**ock **t**ransfer, which quickly copies a rectangular chunk of a pixelmap (a two-dimensional array of pixel values) from one location to another. This is a two-dimensional version of the standard C library function `memcpy()`.

   Suppose we want to rotate an $n \times n$ pixelmap 90° clockwise. One way to do this is to split the pixelmap into four $n/2 \times n/2$ blocks, move each block to its proper position using a sequence of five blits, and then recursively rotate each block. Alternately, we can first recursively rotate the blocks and blit them into place afterwards.



Two algorithms for rotating a pixelmap.
Black arrows indicate blitting the blocks into place.
White arrows indicate recursively rotating the blocks.

The following sequence of pictures shows the first algorithm (blit then recurse) in action.



In the following questions, assume n is a power of two.

(a) Prove that both versions of the algorithm are correct. [Hint: If you exploit all the available symmetries, your proof will only be a half of a page long.]

(b) *Exactly* how many blits does the algorithm perform?

(c) What is the algorithm's running time if a $k \times k$ blit takes $O(k^2)$ time?

(d) What if a $k \times k$ blit takes only $O(k)$ time?

5. A company is planning a party for its employees. The employees in the company are organized into a strict hierarchy, that is, a tree with the company president at the root. The organizers of the party have assigned a real number to each employee measuring how 'fun' the employee is. In order to keep things social, there is one restriction on the guest list: an employee cannot attend the party if their immediate supervisor is present. On the other hand, the president of the company *must* attend the party, even though she has a negative fun rating; it's her company, after all. Give an algorithm that makes a guest list for the party that maximizes the sum of the 'fun' ratings of the guests.

6. Suppose you have a subroutine that can find the median of a set of n items (*i.e.*, the $\lfloor n/2 \rfloor$ smallest) in $O(n)$ time. Give an algorithm to find the kth biggest element (for arbitrary k) in $O(n)$ time.

7. You're walking along the beach and you stub your toe on something in the sand. You dig around it and find that it is a treasure chest full of gold bricks of different (integral) weight. Your knapsack can only carry up to weight n before it breaks apart. You want to put as much in it as possible without going over, but you *cannot* break the gold bricks up.

   (a) Suppose that the gold bricks have the weights $1, 2, 4, 8, \ldots, 2^k$, $k \geq 1$. Describe and prove correct a greedy algorithm that fills the knapsack as much as possible without going over.

   (b) Give a set of 3 weight values for which the greedy algorithm does *not* yield an optimal solution and show why.

   (c) Give a dynamic programming algorithm that yields an optimal solution for an arbitrary set of gold brick values.

CS 373: Combinatorial Algorithms, Spring 2001
Homework 2 (due Thu. Feb. 15, 2001 at 11:59 PM)

Name:		
Net ID:	Alias:	U $^3\!/\!_4$ 1

Name:		
Net ID:	Alias:	U $^3\!/\!_4$ 1

Name:		
Net ID:	Alias:	U $^3\!/\!_4$ 1

Starting with Homework 1, homeworks may be done in teams of up to three people. Each team turns in just one solution, and every member of a team gets the same grade. Since 1-unit graduate students are required to solve problems that are worth extra credit for other students, **1-unit grad students may not be on the same team as 3/4-unit grad students or undergraduates.**

Neatly print your name(s), NetID(s), and the alias(es) you used for Homework 0 in the boxes above. Please also tell us whether you are an undergraduate, 3/4-unit grad student, or 1-unit grad student by circling U, $^3\!/\!_4$, or 1, respectively. Staple this sheet to the top of your homework.

Required Problems

1. Suppose we are given two sorted arrays $A[1\,..\,n]$ and $B[1\,..\,n]$ and an integer k. Describe an algorithm to find the kth smallest element in the union of A and B. (For example, if $k = 1$, your algorithm should return the smallest element of $A \cup B$; if $k = n$, our algorithm should return the median of $A \cup B$.) You can assume that the arrays contain no duplicates. For full credit, your algorithm should run in $\Theta(\log n)$ time. *[Hint: First try to solve the special case $k = n$.]*

2. Say that a binary search tree is *augmented* if every node v also stores $|v|$, the size of its subtree.

   (a) Show that a rotation in an augmented binary tree can be performed in constant time.

   (b) Describe an algorithm ScapegoatSelect(k) that selects the kth smallest item in an augmented scapegoat tree in $O(\log n)$ *worst-case* time.

   (c) Describe an algorithm SplaySelect(k) that selects the kth smallest item in an augmented splay tree in $O(\log n)$ *amortized* time.

(d) Describe an algorithm TreapSelect(k) that selects the kth smallest item in an augmented treap in $O(\log n)$ *expected* time.

3. (a) Prove that only one subtree gets rebalanced in a scapegoat tree insertion.

   (b) Prove that $I(v) = 0$ in every node of a perfectly balanced tree. (Recall that $I(v) = \max\{0, |T| - |s| - 1\}$, where T is the child of greater height and s the child of lesser height, and $|v|$ is the number of nodes in subtree v. A perfectly balanced tree has two perfectly balanced subtrees, each with as close to half the nodes as possible.)

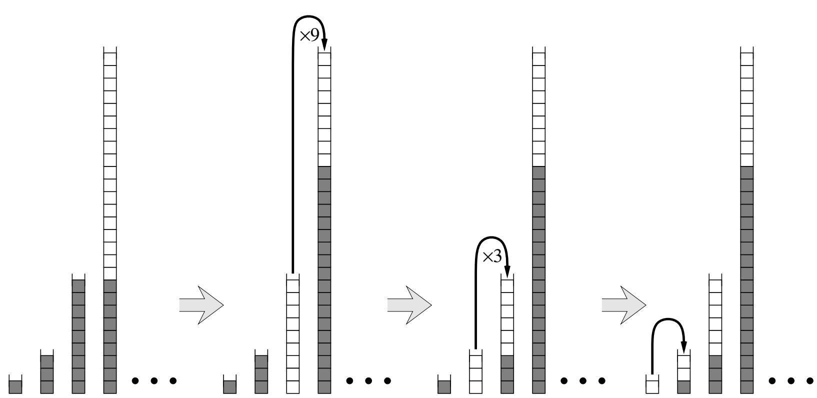
   *(c) Show that you can rebuild a fully balanced binary tree from an unbalanced tree in $O(n)$ time using only $O(\log n)$ additional memory.

4. Suppose we can insert or delete an element into a hash table in constant time. In order to ensure that our hash table is always big enough, without wasting a lot of memory, we will use the following global rebuilding rules:

   - After an insertion, if the table is more than 3/4 full, we allocate a new table twice as big as our current table, insert everything into the new table, and then free the old table.
   - After a deletion, if the table is less than 1/4 full, we we allocate a new table half as big as our current table, insert everything into the new table, and then free the old table.

   Show that for any sequence of insertions and deletions, the amortized time per operation is still a constant. Do *not* use the potential method (it makes it much more difficult).

5. A *multistack* consists of an infinite series of stacks $S_0, S_1, S_2, \ldots$, where the ith stack S_i can hold up to 3^i elements. Whenever a user attempts to push an element onto any full stack S_i, we first move all the elements in S_i to stack S_{i+1} to make room. But if S_{i+1} is already full, we first move all its members to S_{i+2}, and so on. Moving a single element from one stack to the next takes $O(1)$ time.



Making room for one new element in a multistack.

   (a) *[1 point]* In the worst case, how long does it take to push one more element onto a multistack containing n elements?

   (b) *[9 points]* Prove that the amortized cost of a push operation is $O(\log n)$, where n is the maximum number of elements in the multistack. You can use any method you like.

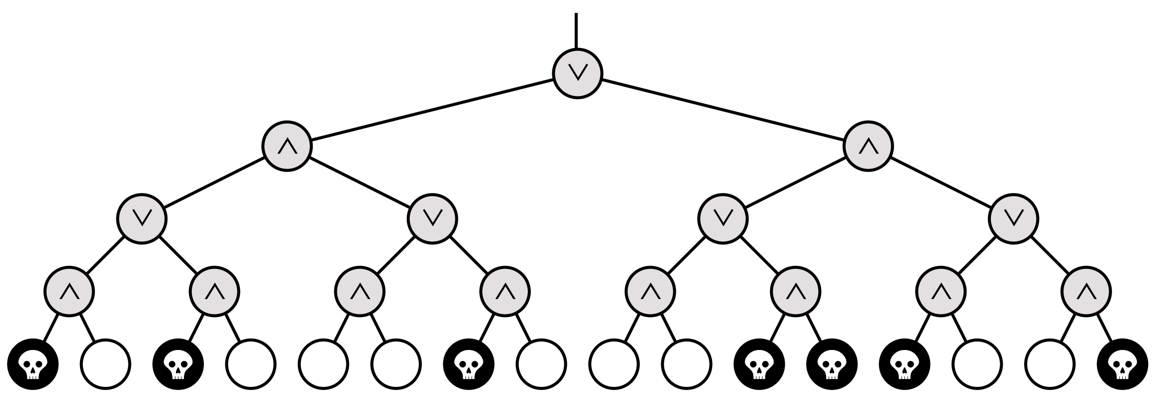
6. *[This problem is required only for graduate students taking CS 373 for a full unit; anyone else can submit a solution for extra credit.]*

   Death knocks on your door one cold blustery morning and challenges you to a game. Death knows that you are an algorithms student, so instead of the traditional game of chess, Death presents you with a complete binary tree with 4^n leaves, each colored either black or white. There is a token at the root of the tree. To play the game, you and Death will take turns moving the token from its current node to one of its children. The game will end after $2n$ moves, when the token lands on a leaf. If the final leaf is black, you die; if it's white, you will live forever. You move first, so Death gets the last turn.

   You can decide whether it's worth playing or not as follows. Imagine that the nodes at even levels (where it's your turn) are OR gates, the nodes at odd levels (where it's Death's turn) are *and* gates. Each gate gets its input from its children and passes its output to its parent. White and black stand for TRUE and FALSE. If the output at the top of the tree is TRUE, then you can win and live forever! If the output at the top of the tree is FALSE, you should challenge Death to a game of Twister instead.

   (a) (2 pts) Describe and analyze a deterministic algorithm to determine whether or not you can win. *[Hint: This is easy!]*

   (b) (8 pts) Unfortunately, Death won't let you even look at every node in the tree. Describe a *randomized* algorithm that determines whether you can win in $\Theta(3^n)$ expected time. *[Hint: Consider the case $n = 1$.]*



Practice Problems

1. (a) Show that it is possible to transform any n-node binary search tree into any other n-node binary search tree using at most $2n - 2$ rotations.

   *(b) Use fewer than $2n - 2$ rotations. Nobody knows how few rotations are required in the worst case. There is an algorithm that can transform any tree to any other in at most $2n - 6$ rotations, and there are pairs of trees that are $2n - 10$ rotations apart. These are the best bounds known.

2. Faster Longest Increasing Subsequence(LIS)
   Give an $O(n \log n)$ algorithm to find the longest increasing subsequence of a sequence of numbers. [Hint: In the dynamic programming solution, you don't really have to look back at all previous items. There was a practice problem on HW 1 that asked for an $O(n^2)$ algorithm for this. If you are having difficulty, look at the solution provided in the HW 1 solutions.]

3. Amortization

   (a) Modify the binary double-counter (see class notes Sept 12) to support a new operation SIGN, which determines whether the number being stored is positive, negative, or zero, in constant time. The amortized time to increment or decrement the counter should still be a constant.

   [Hint: Suppose p is the number of significant bits in P, and n is the number of significant bits in N. For example, if $P = 17 = 10001_2$ and $N = 0$, then $p = 5$ and $n = 0$. Then $p - n$ always has the same sign as $P - N$. Assume you can update p and n in $O(1)$ time.]

   *(b) Do the same but now you can't assume that p and n can be updated in $O(1)$ time.

*4. Amortization
   Suppose instead of powers of two, we represent integers as the sum of Fibonacci numbers. In other words, instead of an array of bits, we keep an array of 'fits', where the ith least significant fit indicates whether the sum includes the ith Fibonacci number F_i. For example, the fit string 101110 represents the number $F_6 + F_4 + F_3 + F_2 = 8 + 3 + 2 + 1 = 14$. Describe algorithms to increment and decrement a fit string in constant amortized time. [Hint: Most numbers can be represented by more than one fit string. This is *not* the same representation as on Homework 0.]

5. Detecting overlap

   (a) You are given a list of ranges represented by min and max (*e.g.*, [1,3], [4,5], [4,9], [6,8], [7,10]). Give an $O(n \log n)$-time algorithm that decides whether or not a set of ranges contains a pair that overlaps. You need not report all intersections. If a range completely covers another, they are overlapping, even if the boundaries do not intersect.

   (b) You are given a list of rectangles represented by min and max x- and y-coordinates. Give an $O(n \log n)$-time algorithm that decides whet her or not a set of rectangles contains a pair that overlaps (with the same qualifications as above). [Hint: sweep a vertical line from left to right, performing some processing whenever an end-point is encountered. Use a balanced search tree to maintain any extra info you might need.]

6. Comparison of Amortized Analysis Methods
   A sequence of n operations is performed on a data structure. The ith operation costs i if i is an exact power of 2, and 1 otherwise. That is operation i costs $f(i)$, where:

$$f(i) = \begin{cases} i, & i = 2^k, \\ 1, & \text{otherwise} \end{cases}$$

   Determine the amortized cost per operation using the following methods of analysis:

   (a) Aggregate method
   (b) Accounting method
   *(c) Potential method

CS 373: Combinatorial Algorithms, Spring 2001

Homework 3 (due Thursday, March 8, 2001 at 11:59.99 p.m.)

Name:		
Net ID:	Alias:	U $^3/_4$ 1

Name:		
Net ID:	Alias:	U $^3/_4$ 1

Name:		
Net ID:	Alias:	U $^3/_4$ 1

Starting with Homework 1, homeworks may be done in teams of up to three people. Each team turns in just one solution, and every member of a team gets the same grade. Since 1-unit graduate students are required to solve problems that are worth extra credit for other students, **1-unit grad students may not be on the same team as 3/4-unit grad students or undergraduates.**

Neatly print your name(s), NetID(s), and the alias(es) you used for Homework 0 in the boxes above. Please also tell us whether you are an undergraduate, 3/4-unit grad student, or 1-unit grad student by circling U, $^3/_4$, or 1, respectively. Staple this sheet to the top of your homework.

Required Problems

1. Hashing:
   A hash table of size m is used to store n items with $n \leq m/2$. Open addressing is used for collision resolution.

   (a) Assuming uniform hashing, show that for $i = 1, 2, \ldots, n$, the probability that the i^{th} insertion requires strictly more than k probes is at most 2^{-k}.

   (b) Show that for $i = 1, 2, \ldots, n$, the probability that the i^{th} insertion requires more than $2 \lg n$ probes is at most $1/n^2$.

   Let the random variable X_i denote the number of probes required by the i^{th} insertion. You have shown in part (b) that $\Pr\{X_i > 2 \lg n\} \leq 1/n^2$. Let the random variable $X = \max_{1 \leq i \leq n} X_i$ denote the maximum number of probes required by any of the n insertions.

   (c) Show that $\Pr\{X > 2 \lg n\} \leq 1/n$.

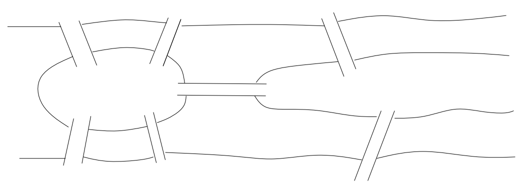
   (d) Show that the expected length of the longest probe sequence is $E[X] = O(\lg n)$.

2. Reliable Network:
Suppose you are given a graph of a computer network $G = (V, E)$ and a function $r(u, v)$ that gives a reliability value for every edge $(u, v) \in E$ such that $0 \leq r(u, v) \leq 1$. The reliability value gives the probability that the network connection corresponding to that edge will *not* fail. Describe and analyze an algorithm to find the most reliable path from a given source vertex s to a given target vertex t.

3. Aerophobia:
After graduating you find a job with Aerophobes-R′-Us, the leading traveling agency for aerophobic people. Your job is to build a system to help customers plan airplane trips from one city to another. All of your customers are afraid of flying so the trip should be as short as possible.

   In other words, a person wants to fly from city A to city B in the shortest possible time. S/he turns to the traveling agent who knows all the departure and arrival times of all the flights on the planet. Give an algorithm that will allow the agent to choose an optimal route to minimize the total time in transit. Hint: rather than modify Dijkstra's algorithm, modify the data. The total transit time is from departure to arrival at the destination, so it will include layover time (time waiting for a connecting flight).

4. The Seven Bridges of Königsberg:
During the eighteenth century the city of Königsberg in East Prussia was divided into four sections by the Pregel river. Seven bridges connected these regions, as shown below. It was said that residents spent their Sunday walks trying to find a way to walk about the city so as to cross each bridge exactly once and then return to their starting point.



   (a) Show how the residents of the city could accomplish such a walk or prove no such walk exists.
   (b) Given any undirected graph $G = (V, E)$, give an algorithm that finds a cycle in the graph that visits every edge exactly once, or says that it can't be done.

5. Minimum Spanning Tree changes:
Suppose you have a graph G and an MST of that graph (i.e. the MST has already been constructed).

   (a) Give an algorithm to update the MST when an edge is added to G.
   (b) Give an algorithm to update the MST when an edge is deleted from G.
   (c) Give an algorithm to update the MST when a vertex (and possibly edges to it) is added to G.

6. Nesting Envelopes
[This problem is required only for graduate students taking CS 373 for a full unit; anyone else can submit a solution for extra credit.] You are given an unlimited number of each of n different types of envelopes. The dimensions of envelope type i are $x_i \times y_i$. In nesting envelopes inside one another, you can place envelope A inside envelope B if and only if the dimensions A are *strictly smaller* than the dimensions of B. Design and analyze an algorithm to determine the largest number of envelopes that can be nested inside one another.

Practice Problems

1. Makefiles:
In order to facilitate recompiling programs from multiple source files when only a small number of files have been updated, there is a UNIX utility called 'make' that only recompiles those files that were changed after the most recent compilation, *and* any intermediate files in the compilation that depend on those that were changed. A Makefile is typically composed of a list of source files that must be compiled. Each of these source files is dependent on some of the other files which are listed. Thus a source file must be recompiled if a file on which it depends is changed.

   Assuming you have a list of which files have been recently changed, as well as a list for each source file of the files on which it depends, design an algorithm to recompile only those necessary. DO NOT worry about the details of parsing a Makefile.

★2. Let the hash function for a table of size m be

$$h(x) = \lfloor Amx \rfloor \bmod m$$

   where $A = \hat{\phi} = \frac{\sqrt{5}-1}{2}$. Show that this gives the best possible spread, i.e. if the x are hashed in order, $x+1$ will be hashed in the largest remaining contiguous interval.

3. The incidence matrix of an undirected graph $G = (V, E)$ is a $|V| \times |E|$ matrix $B = (b_{ij})$ such that

$$b_{ij} = \begin{cases} 1 & (i,j) \in E, \\ 0 & (i,j) \notin E. \end{cases}$$

   (a) Describe what all the entries of the matrix product BB^T represent (B^T is the matrix transpose). Justify.

   (b) Describe what all the entries of the matrix product B^TB represent. Justify.

   ★(c) Let $C = BB^T - 2A$. Let C' be C with the first row and column removed. Show that $\det C'$ is the number of spanning trees. (A is the adjacency matrix of G, with zeroes on the diagonal).

4. $o(V^2)$ Adjacency Matrix Algorithms

   (a) Give an $O(V)$ algorithm to decide whether a directed graph contains a *sink* in an adjacency matrix representation. A sink is a vertex with in-degree $V-1$.

(b) An undirected graph is a scorpion if it has a vertex of degree 1 (the sting) connected to a vertex of degree two (the tail) connected to a vertex of degree $V - 2$ (the body) connected to the other $V - 3$ vertices (the feet). Some of the feet may be connected to other feet.

Design an algorithm that decides whether a given adjacency matrix represents a scorpion by examining only $O(V)$ of the entries.

(c) Show that it is impossible to decide whether G has at least one edge in $O(V)$ time.

5. Shortest Cycle:
Given an **undirected** graph $G = (V, E)$, and a weight function $f : E \to \mathbf{R}$ on the **edges**, give an algorithm that finds (in time polynomial in V and E) a cycle of smallest weight in G.

6. Longest Simple Path:
Let graph $G = (V, E)$, $|V| = n$. A *simple path* of G, is a path that does not contain the same vertex twice. Use dynamic programming to design an algorithm (not polynomial time) to find a simple path of maximum length in G. Hint: It can be done in $O(n^c 2^n)$ time, for some constant c.

7. Minimum Spanning Tree:
Suppose all edge weights in a graph G are equal. Give an algorithm to compute an MST.

8. Transitive reduction:
Give an algorithm to construct a *transitive reduction* of a directed graph G, i.e. a graph G^{TR} with the fewest edges (but with the same vertices) such that there is a path from a to b in G iff there is also such a path in G^{TR}.

9. (a) What is $5^{2^{29^{5^0}+23^{4^1}+17^{3^2}+11^{2^3}+5^{1^4}}}$ mod 6?

(b) What is the capital of Nebraska? Hint: It is not Omaha. It is named after a famous president of the United States that was not George Washington. The distance from the Earth to the Moon averages roughly 384,000 km.

CS 373: Combinatorial Algorithms, Spring 2001

http://www-courses.cs.uiuc.edu/~cs373

Homework 4 (due Thu. March 29, 2001 at 11:59:59 pm)

Name:		
Net ID:	Alias:	U $^3/_4$ 1

Name:		
Net ID:	Alias:	U $^3/_4$ 1

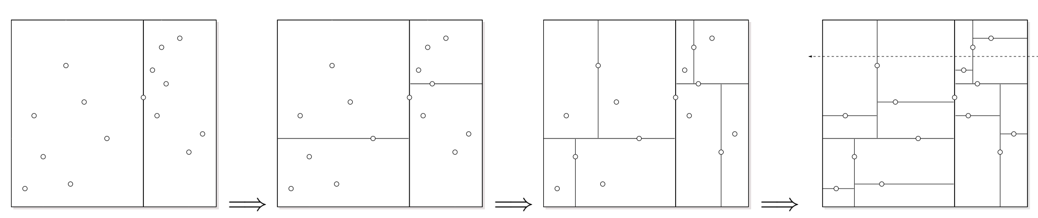
Name:		
Net ID:	Alias:	U $^3/_4$ 1

Homeworks may be done in teams of up to three people. Each team turns in just one solution, and every member of a team gets the same grade. Since 1-unit graduate students are required to solve problems that are worth extra credit for other students, **1-unit grad students may not be on the same team as 3/4-unit grad students or undergraduates.**

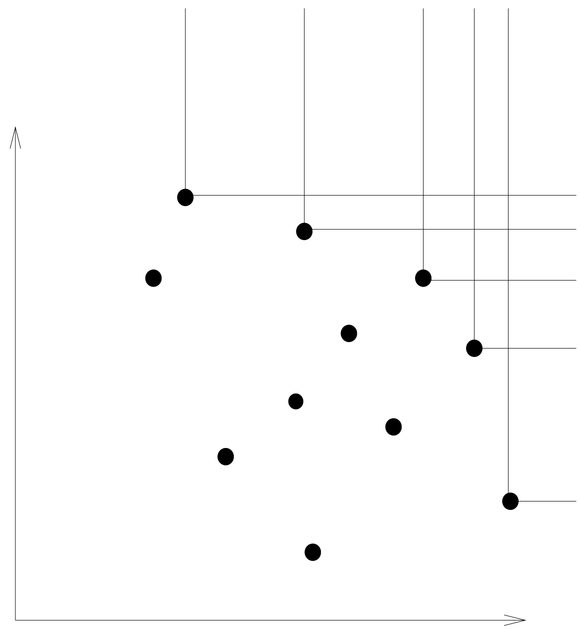
Neatly print your name(s), NetID(s), and the alias(es) you used for Homework 0 in the boxes above. Please also tell us whether you are an undergraduate, 3/4-unit grad student, or 1-unit grad student by circling U, $^3/_4$, or 1, respectively. Staple this sheet to the top of your homework.

Required Problems

1. Suppose we have n points scattered inside a two-dimensional box. A *kd-tree* recursively subdivides the rectangle as follows. First we split the box into two smaller boxes with a *vertical* line, then we split each of those boxes with *horizontal* lines, and so on, always alternating between horizontal and vertical splits. Each time we split a box, the splitting line partitions the rest of the interior points *as evenly as possible* by passing through a median point inside the box (*not* on the boundary). If a box doesn't contain any points, we don't split it any more; these final empty boxes are called *cells*.



Successive divisions of a kd-tree for 15 points. The dashed line crosses four cells.



An example staircase as in problem 3.

(a) How many cells are there, as a function of n? Prove your answer is correct.

(b) In the worst case, *exactly* how many cells can a horizontal line cross, as a function of n? Prove your answer is correct. Assume that $n = 2^k - 1$ for some integer k.

(c) Suppose we have n points stored in a kd-tree. Describe an algorithm that counts the number of points above a horizontal line (such as the dashed line in the figure) in $O(\sqrt{n})$ time.

*(d) *[Optional: 5 pts extra credit]* Find an algorithm that counts the number of points that lie inside a rectangle R and show that it takes $O(\sqrt{n})$ time. You may assume that the sides of the rectangle are parallel to the sides of the box.

2. Circle Intersection *[This problem is worth 20 points]*
Describe an algorithm to decide, given n circles in the plane, whether any two of them intersect, in $O(n \log n)$ time. Each circle is specified by three numbers: its radius and the x- and y-coordinates of its center.

We only care about intersections between circle boundaries; concentric circles do not intersect. What general position assumptions does your algorithm require? *[Hint: Modify an algorithm for detecting line segment intersections, but describe your modifications very carefully! There are at least two very different solutions.]*

3. Staircases
You are given a set of points in the first quadrant. A *left-up* point of this set is defined to be a point that has no points both greater than it in both coordinates. The left-up subset of a set of points then forms a *staircase* (see figure).

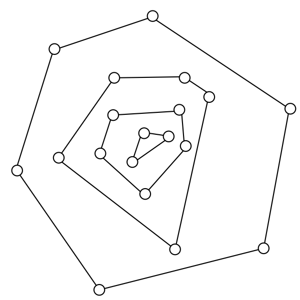
(a) Prove that left-up points do not necessarily lie on the convex hull.

(b) Give an $O(n \log n)$ algorithm to find the staircase of a set of points.

(c) Assume that points are chosen uniformly at random within a rectangle. What is the average number of points in a staircase? Justify. Hint: you will be able to give an exact answer rather than just asymptotics. You have seen the same analysis before.

4. Convex Layers
Given a set Q of points in the plane, define the *convex layers* of Q inductively as follows: The first convex layer of Q is just the convex hull of Q. For all $i > 1$, the ith convex layer is the convex hull of Q after the vertices of the first $i - 1$ layers have been removed.

Give an $O(n^2)$-time algorithm to find all convex layers of a given set of n points.



A set of points with four convex layers.

5. *[This problem is required only for graduate students taking CS 373 for a full unit; anyone else can submit a solution for extra credit.]* Solve the travelling salesman problem for points in convex position (ie, the vertices of a convex polygon). Finding the shortest cycle that visits every point is easy – it's just the convex hull. Finding the shortest path that visits evey point is a little harder, because the path can cross through the interior.

(a) Show that the optimal path cannot be one that crosses itself.

(b) Describe an $O(n^2)$ time dynamic programming algorithm to solve the problem.

Practice Problems

1. Basic Computation (assume two dimensions and *exact* arithmetic)

   (a) Intersection: Extend the basic algorithm to determine if two line segments intersect by taking care of *all* degenerate cases.

   (b) Simplicity: Give an $O(n \log n)$ algorithm to determine whether an n-vertex polygon is simple.

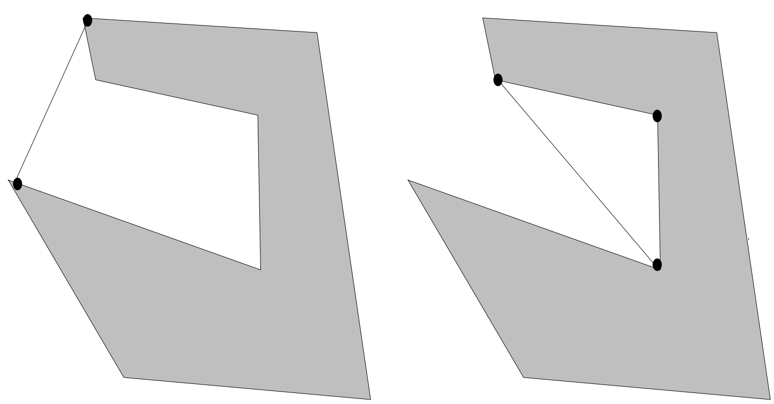
   (c) Area: Give an algorithm to compute the area of a simple n-polygon (not necessarily convex) in $O(n)$ time.

   (d) Inside: Give an algorithm to determine whether a point is within a simple n-polygon (not necessarily convex) in $O(n)$ time.

2. External Diagonals and Mouths

   (a) A pair of polygon vertices defines an *external diagonal* if the line segment between them is completely outside the polygon. Show that every nonconvex polygon has at least one external diagonal.

   (b) Three consective polygon vertices p, q, r form a *mouth* if p and r define an external diagonal. Show that every nonconvex polygon has at least one mouth.



An external diagonal   A mouth

3. On-Line Convex Hull
   We are given the set of points one point at a time. After receiving each point, we must compute the convex hull of all those points so far. Give an algorithm to solve this problem in $O(n^2)$ (We could obviously use Graham's scan n times for an $O(n^2 \log n)$ algorithm). Hint: How do you maintain the convex hull?

4. Another On-Line Convex Hull Algorithm

   (a) Given an n-polygon and a point outside the polygon, give an algorithm to find a tangent.

   *(b) Suppose you have found both tangents. Give an algorithm to remove the points from the polygon that are within the angle formed by the tangents (as segments!) and the opposite side of the polygon.

(c) Use the above to give an algorithm to compute the convex hull on-line in $O(n \log n)$

5. Order of the size of the convex hull
   The convex hull on $n \geq 3$ points can have anywhere from 3 to n points. The average case depends on the distribution.

   (a) Prove that if a set of points is chosen randomly within a given rectangle then the average size of the convex hull is $O(\log n)$.

   *(b) Prove that if a set of points is chosen randomly within a given circle then the average size of the convex hull is $O(n^{1/3})$.

6. Ghostbusters and Ghosts
   A group of n ghostbusters is battling n ghosts. Each ghostbuster can shoot a single energy beam at a ghost, eradicating it. A stream goes in a straight line and terminates when it hits a ghost. The ghostbusters must all fire at the same time and no two energy beams may cross (it would be bad). The positions of the ghosts and ghostbusters is fixed in the plane (assume that no three points are collinear).

   (a) Prove that for any configuration of ghosts and ghostbusters there exists such a non-crossing matching.

   (b) Show that there exists a line passing through one ghostbuster and one ghost such that the number of ghostbusters on one side of the line equals the number of ghosts on the same side. Give an efficient algorithm to find such a line.

   (c) Give an efficient divide and conquer algorithm to pair ghostbusters and ghosts so that no two streams cross.

CS 373: Combinatorial Algorithms, Spring 2001

http://www-courses.cs.uiuc.edu/~cs373

Homework 5 (due Tue. Apr. 17, 2001 at 11:59 pm)

Name:		
Net ID:	Alias:	U $^3\!/_4$ 1

Name:		
Net ID:	Alias:	U $^3\!/_4$ 1

Name:		
Net ID:	Alias:	U $^3\!/_4$ 1

Neatly print your name(s), NetID(s), and the alias(es) you used for Homework 0 in the boxes above. Please also tell us whether you are an undergraduate, 3/4-unit grad student, or 1-unit grad student by circling U, $^3\!/_4$, or 1, respectively. Staple this sheet to the top of your homework.

RequiredProblems

1. Prove that finding the second smallest of n elements takes EXACTLY $n + \lceil \lg n \rceil - 2$ comparisons in the worst case. Prove for both upper and lower bounds. Hint: find the (first) smallest using an elimination tournament.

2. *Fibonacci strings* are defined as follows:

$$F_1 = \text{“b”}, \qquad F_2 = \text{“a”}, \qquad \text{and } F_n = F_{n-1}F_{n-2}, (n > 2)$$

where the recursive rule uses concatenation of strings, so F_3 is “ab”, F_4 is “aba”. Note that the length of F_n is the nth Fibonacci number.

   (a) Prove that in any Fibonacci string there are no two b’s adjacent and no three a’s.

   (b) Give the unoptimized and optimized ‘prefix’ (fail) function for F_7.

   (c) Prove that, in searching for the Fibonacci string F_k, the unoptimized KMP algorithm can shift $\lceil k/2 \rceil$ times in a row trying to match the last character of the pattern. In other words, prove that there is a chain of failure links $m \to fail[m] \to fail[fail[m]] \to \ldots$ of length $\lceil k/2 \rceil$, and find an example text T that would cause KMP to traverse this entire chain at a single text position.

(d) Prove that the unoptimized KMP algorithm can shift $k-2$ times in a row at the same text position when searching for F_k. Again, you need to find an example text T that would cause KMP to traverse this entire chain on the same text character.

(e) How do the failure chains in parts (c) and (d) change if we use the optimized failure function instead?

3. Two-stage sorting

(a) Suppose we are given an array $A[1..n]$ of distinct integers. Describe an algorithm that splits A into n/k subarrays, each with k elements, such that the elements of each subarray $A[(i-1)k+1..ik]$ are sorted. Your algorithm should run in $O(n \log k)$ time.

(b) Given an array $A[1..n]$ that is already split into n/k sorted subarrays as in part (a), describe an algorithm that sorts the entire array in $O(n \log(n/k))$ time.

(c) Prove that your algorithm from part (a) is optimal.

(d) Prove that your alogrithm from part (b) is optimal.

4	14	7	3	1	20	11	9	5	13	12	19	10	16	17	2	8	6	18	15

$$\Downarrow (a)$$

1	3	4	7	14	5	9	11	13	20	10	12	16	17	19	2	6	8	15	18

$$\Downarrow (b)$$

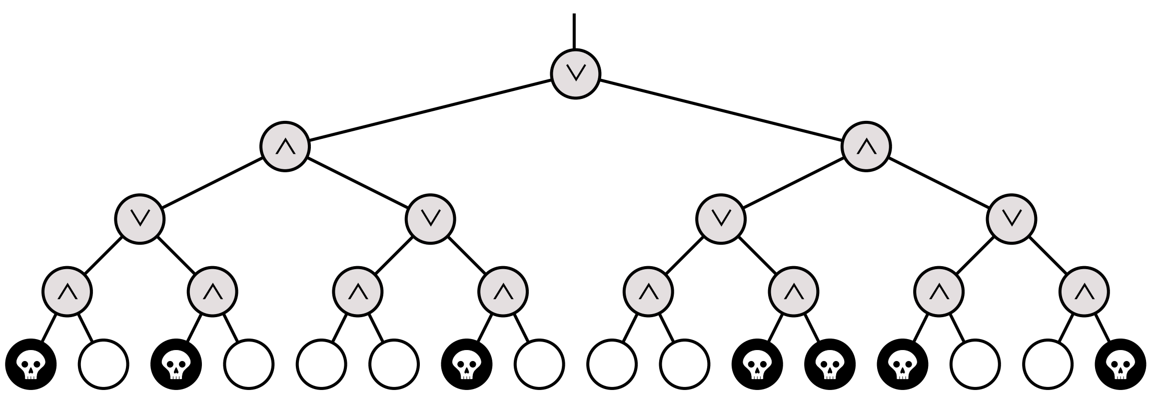
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20

4. Show how to extend the Rabin-Karp fingerprinting method to handle the problem of looking for a given $m \times m$ pattern in an $n \times n$ array of characters. (The pattern may be shifted horizontally and vertically, but it may not be rotated.)

5. Death knocks on your door once more on a warm spring day. He remembers that you are an algorithms student and that you soundly defeated him last time and are now living out your immortality. Death is in a bit of a quandry. He has been losing a lot and doesn't know why. He wants you to prove a lower bound on your deterministic algorithm so that he can reap more souls. If you have forgotten, the game goes like this: It is a complete binary tree with 4^n leaves, each colored black or white. There is a toke at the root of the tree. To play the game, you and Death took turns movin the token from its current node to one of its children. The game ends after $2n$ moves, when the token lands on a leaf. If the final leaf is black, the player dies; if it's white, you will live forever. You move first, so Death gets the last turn.

   You decided whether it's worth playing or not as follows. Imagine that the nodes at even levels (where it's your turn) are OR gates, the nodes at odd levels (where it's Death's turn) are *and* gates. Each gate gets its input from its children and passes its output to its parent. White and black stand for TRUE and FALSE. If the output at the top of the tree is TRUE, then you can win and live forever! If the output at the top of the tree is FALSE, you should've challenge Death to a game of Twister instead.

   Prove that *any* deterministic algorithm must examine *every* leaf of the tree in the worst case. Since there are 4^n leaves, this implies that any deterministic algorithm must take $\Omega(4^n)$ time in the worst case. Use an adversary argument, or in other words, assume Death cheats.



6. *[This problem is required only for graduate students taking CS 373 for a full unit; anyone else can submit a solution for extra credit.]*

   Lower Bounds on Adjacency Matrix Representations of Graphs

   (a) Prove that the time to determine if an undirected graph has a cycle is $\Omega(V^2)$.

   (b) Prove that the time to determine if there is a path between two nodes in an undirected graph is $\Omega(V^2)$.

Practice Problems

1. String matching with wild-cards
   Suppose you have an alphabet for patterns that includes a 'gap' or wild-card character; any length string of any characters can match this additional character. For example if '$*$' is the wild-card, then the pattern 'foo*bar*nad' can be found in 'foofoowangbarnad'. Modify the computation of the prefix function to correctly match strings using KMP.

2. Prove that there is no comparison sort whose running time is linear for at least $1/2$ of the $n!$ inputs of length n. What about at least $1/n$? What about at least $1/2^n$?.

3. Prove that $2n - 1$ comparisons are necessary in the worst case to merge two sorted lists containing n elements each.

4. Find asymptotic upper and lower bounds to $\lg(n!)$ without Stirling's approximation (Hint: use integration).

5. Given a sequence of n elements of n/k blocks (k elements per block) all elements in a block are less than those to the right in sequence, show that you cannot have the whole sequence sorted in better than $\Omega(n \lg k)$. Note that the entire sequence would be sorted if each of the n/k blocks were individually sorted in place. Also note that combining the lower bounds for each block is not adequate (that only gives an upper bound).

6. Show how to find the occurrences of pattern P in text T by computing the prefix function of the string PT (the concatenation of P and T).

CS 373: Combinatorial Algorithms, Spring 2001

`http://www-courses.cs.uiuc.edu/~cs373`

Homework 6 (due Tue. May 1, 2001 at 11:59.99 p.m.)

Name:		
Net ID:	Alias:	U $^3\!/_4$ 1

Name:		
Net ID:	Alias:	U $^3\!/_4$ 1

Name:		
Net ID:	Alias:	U $^3\!/_4$ 1

Starting with Homework 1, homeworks may be done in teams of up to three people. Each team turns in just one solution, and every member of a team gets the same grade. Since 1-unit graduate students are required to solve problems that are worth extra credit for other students, **1-unit grad students may not be on the same team as 3/4-unit grad students or undergraduates.**

Neatly print your name(s), NetID(s), and the alias(es) you used for Homework 0 in the boxes above. Please also tell us whether you are an undergraduate, 3/4-unit grad student, or 1-unit grad student by circling U, $^3\!/_4$, or 1, respectively. Staple this sheet to the top of your homework.

Note: You will be held accountable for the appropriate responses for answers (e.g. give models, proofs, analyses, etc). For NP-complete problems you should prove everything rigorously, i.e. for showing that it is in NP, give a description of a certificate and a poly time algorithm to verify it, and for showing NP-hardness, you must show that your reduction is polytime (by similarly proving something about the algorithm that does the transformation) and proving both directions of the 'if and only if' (a solution of one is a solution of the other) of the many-one reduction.

Required Problems

1. Complexity
   (a) Prove that P $\subseteq$ co-NP.
   (b) Show that if NP $\neq$ co-NP, then *every* NP-complete problem is *not* a member of co-NP.

2. 2-CNF-SAT
   Prove that deciding satisfiability when all clauses have at most 2 literals is in P.

3. Graph Problems

(a) SUBGRAPH-ISOMORPHISM
Show that the problem of deciding whether one graph is a subgraph of another is NP-complete.

(b) LONGEST-PATH
Show that the problem of deciding whether an unweighted undirected graph has a path of length greater than k is NP-complete.

4. PARTITION, SUBSET-SUM
PARTITION is the problem of deciding, given a set of numbers, whether there exists a subset whose sum equals the sum of the complement, i.e. given $S = s_1, s_2 \ldots, s_n$, does there exist a subset S' such that $\sum_{s \in S'} s = \sum_{t \in S - S'} t$. SUBSET-SUM is the problem of deciding, given a set of numbers and a target sum, whether there exists a subset whose sum equals the target, i.e. given $S = s_1, s_2 \ldots, s_n$ and k, does there exist a subset S' such that $\sum_{s \in S'} s = k$. Give two reduction, one in both directions.

5. BIN-PACKING Consider the bin-packing problem: given a finite set U of n items and the positive integer size $s(u)$ of each item $u \in U$, can U be partitioned into k disjoint sets $U_1, \ldots, U_k$ such that the sum of the sizes of the items in each set does not exceed B? Show that the bin-packing problem is NP-Complete. [Hint: Use the result from the previous problem.]

6. 3SUM
[This problem is required only for graduate students taking CS 373 for a full unit; anyone else can submit a solution for extra credit.]
Describe an algorithm that solves the following problem as quickly as possible: Given a set of n numbers, does it contain three elements whose sum is zero? For example, your algorithm should answer TRUE for the set $\{-5, -17, 7, -4, 3, -2, 4\}$, since $-5+7+(-2) = 0$, and FALSE for the set $\{-6, 7, -4, -13, -2, 5, 13\}$.

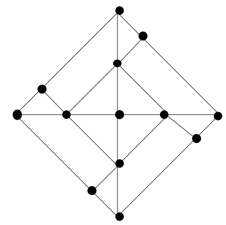


Figure 1. Gadget for PLANAR-3-COLOR.

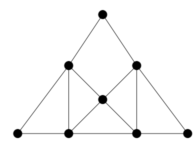


Figure 2. Gadget for DEGREE-4-PLANAR-3-COLOR.

Practice Problems

1. Consider finding the median of 5 numbers by using only comparisons. What is the exact worst case number of comparisons needed to find the median. Justify (exhibit a set that cannot be done in one less comparisons). Do the same for 6 numbers.

2. EXACT-COVER-BY-4-SETS
The EXACT-COVER-BY-3-SETS problem is defines as the following: given a finite set X with $|X| = 3q$ and a collection C of 3-element subsets of X, does C contain an *exact cover* for X, that is, a subcollection $C' \subseteq C$ such that every element of X occurs in exactly one member of C'?

   Given that EXACT-COVER-BY-3-SETS is NP-complete, show that EXACT-COVER-BY-4-SETS is also NP-complete.

3. PLANAR-3-COLOR
Using 3-COLOR, and the 'gadget' in figure 3, prove that the problem of deciding whether a planar graph can be 3-colored is NP-complete. Hint: show that the gadget can be 3-colored, and then replace any crossings in a planar embedding with the gadget appropriately.

4. DEGREE-4-PLANAR-3-COLOR
Using the previous result, and the 'gadget' in figure 4, prove that the problem of deciding whether a planar graph with no vertex of degree greater than four can be 3-colored is NP-complete. Hint: show that you can replace any vertex with degree greater than 4 with a collection of gadgets connected in such a way that no degree is greater than four.

5. Poly time subroutines can lead to exponential algorithms
Show that an algorithm that makes at most a constant number of calls to polynomial-time subroutines runs in polynomial time, but that a polynomial number of calls to polynomial-time subroutines may result in an exponential-time algorithm.

6. (a) Prove that if G is an undirected bipartite graph with an odd number of vertices, then G is nonhamiltonian. Give a polynomial time algorithm algorithm for finding a **hamiltonian cycle** in an undirected bipartite graph or establishing that it does not exist.
   (b) Show that the **hamiltonian-path** problem can be solved in polynomial time on directed acyclic graphs by giving an efficient algorithm for the problem.
   (c) Explain why the results in previous questions do not contradict the facts that both HAM-CYCLE and HAM-PATH are NP-complete problems.

7. Consider the following pairs of problems:
   (a) MIN SPANNING TREE and MAX SPANNING TREE
   (b) SHORTEST PATH and LONGEST PATH
   (c) TRAVELING SALESMAN PROBLEM and VACATION TOUR PROBLEM (the longest tour is sought).
   (d) MIN CUT and MAX CUT (between s and t)
   (e) EDGE COVER and VERTEX COVER
   (f) TRANSITIVE REDUCTION and MIN EQUIVALENT DIGRAPH

   (all of these seem dual or opposites, except the last, which are just two versions of minimal representation of a graph).

   Which of these pairs are polytime equivalent and which are not? Why?

★8. GRAPH-ISOMORPHISM
   Consider the problem of deciding whether one graph is isomorphic to another.
   (a) Give a brute force algorithm to decide this.
   (b) Give a dynamic programming algorithm to decide this.
   (c) Give an efficient probabilistic algorithm to decide this.
   (d) Either prove that this problem is NP-complete, give a poly time algorithm for it, or prove that neither case occurs.

9. Prove that PRIMALITY (Given n, is n prime?) is in NP $\cap$ co-NP. Hint: co-NP is easy (what's a certificate for showing that a number is composite?). For NP, consider a certificate involving primitive roots and recursively their primitive roots. Show that knowing this tree of primitive roots can be checked to be correct and used to show that n is prime, and that this check takes poly time.

10. How much wood would a woodchuck chuck if a woodchuck could chuck wood?

Write your answers in the separate answer booklet.

1. **Multiple Choice:** Each question below has one of the following answers.

   (a) $\Theta(1)$ (b) $\Theta(\log n)$ (c) $\Theta(n)$ (d) $\Theta(n \log n)$ (e) $\Theta(n^2)$

   For each question, write the letter that corresponds to your answer. You do not need to justify your answers. Each correct answer earns you 1 point, but each incorrect answer *costs* you $\frac{1}{2}$ point. You cannot score below zero.

   (a) What is $\sum_{i=1}^{n} H_i$?

   (b) What is $\sum_{i=1}^{\lg n} 2^i$?

   (c) How many digits do you need to write $n!$ in decimal?

   (d) What is the solution of the recurrence $T(n) = 16T(n/4) + n$?

   (e) What is the solution of the recurrence $T(n) = T(n-2) + \lg n$?

   (f) What is the solution of the recurrence $T(n) = 4T\left(\left\lceil \frac{n+51}{4} \right\rceil - \sqrt{n}\right) + 17n - 2^{8\log^*(n^2)} + 6$?

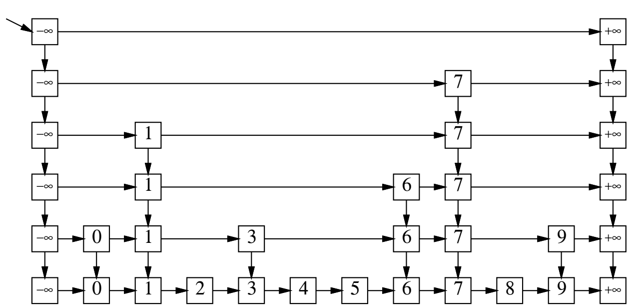
   (g) What is the worst-case running time of randomized quicksort?

   (h) The expected time for inserting one item into a treap is $O(\log n)$. What is the worst-case time for a sequence of n insertions into an initially empty treap?

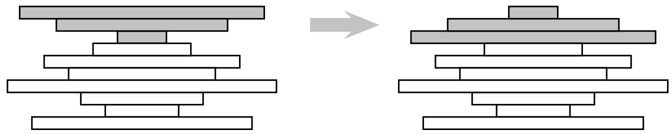
   (i) The amortized time for inserting one item into an n-node splay tree is $O(\log n)$. What is the worst-case time for a sequence of n insertions into an initially empty splay tree?

   (j) In the worst case, how long does it take to solve the traveling salesman problem for $10,000,000,000,000,000$ cities?

2. What is the *exact* expected number of nodes in a skip list storing n keys, *not* counting the sentinel nodes at the beginning and end of each level? Justify your answer. A correct $\Theta()$ bound (with justification) is worth 5 points.



3. Suppose we have a stack of n pancakes of all different sizes. We want to sort the pancakes so that smaller pancakes are on top of larger pancakes. The only operation we can perform is a *flip* — insert a spatula under the top k pancakes, for some k between 1 and n, turn them all over, and put them back on top of the stack.



Flipping the top three pancakes

   (a) (3 pts) Describe an algorithm to sort an arbitrary stack of n pancakes using flips.

   (b) (3 pts) Prove that your algorithm is correct.

   (c) (2 pts) Exactly how many flips does your sorting algorithm perform in the worst case? A correct $\Theta()$ bound is worth one point.

   (d) (2 pts) Suppose one side of each pancake is burned. Exactly how many flips do you need to sort the pancakes, so that the burned side of every pancake is on the bottom? A correct $\Theta()$ bound is worth one point.

4. Suppose we want to maintain a set of values in a data structure subject to the following operations:

   - INSERT(x): Add x to the set (if it isn't already there).
   - DELETERANGE(a, b): Delete every element x in the range $a \le x \le b$. For example, if the set was $\{1, 5, 3, 4, 8\}$, then DELETERANGE$(4, 6)$ would change the set to $\{1, 3, 8\}$.

   Describe and analyze a data structure that supports these operations, such that the amortized cost of either operation is $O(\log n)$. *[Hint: Use a data structure you saw in class. If you use the same* INSERT *algorithm, just say so—you don't need to describe it again in your answer.]*

5. *[1-unit grad students must answer this question.]*

   A *shuffle* of two strings X and Y is formed by interspersing the characters into a new string, keeping the characters of X and Y in the same order. For example, 'bananaananas' is a shuffle of 'banana' and 'ananas' in several different ways.

   $$\mathsf{banana}_{\mathsf{ananas}} \qquad \mathsf{ban}_{\mathsf{ana}}{}^{\mathsf{ana}}{}_{\mathsf{nas}} \qquad \mathsf{b}_{\mathsf{an}}{}^{\mathsf{an}}{}_{\mathsf{a}}{}^{\mathsf{a}}{}_{\mathsf{na}}{}^{\mathsf{na}}{}_{\mathsf{s}}$$

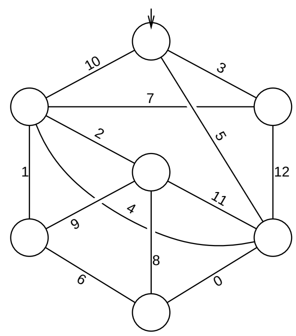
   The strings 'prodgyrnamammiincg' and 'dyprongarmammicing' are both shuffles of 'dynamic' and 'programming':

   $$\mathsf{pro}^{\mathsf{d}}{}_{\mathsf{g}}{}^{\mathsf{y}}{}_{\mathsf{r}}{}^{\mathsf{nam}}{}_{\mathsf{ammi}}{}^{\mathsf{i}}{}_{\mathsf{n}}{}^{\mathsf{c}}{}_{\mathsf{g}} \qquad \mathsf{dy}_{\mathsf{pro}}{}^{\mathsf{n}}{}_{\mathsf{g}}{}^{\mathsf{a}}{}_{\mathsf{r}}{}^{\mathsf{m}}{}_{\mathsf{amm}}{}^{\mathsf{ic}}{}_{\mathsf{ing}}$$

   Given three strings $A[1 \,..\, m]$, $B[1 \,..\, n]$, and $C[1 \,..\, m+n]$, describe and analyze an algorithm to determine whether C is a shuffle of A and B. For full credit, your algorithm should run in $\Theta(mn)$ time.

1. Using any method you like, compute the following subgraphs for the weighted graph below. Each subproblem is worth 3 points. Each incorrect edge costs you 1 point, but you cannot get a negative score for any subproblem.

   (a) a depth-first search tree, starting at the top vertex;
   (b) a breadth-first search tree, starting at the top vertex;
   (c) a shortest path tree, starting at the top vertex;
   (d) the **maximum** spanning tree.

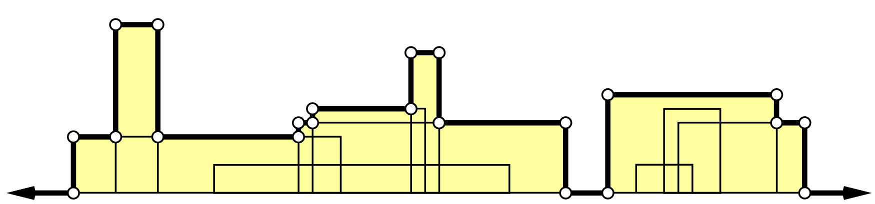


2. (a) *[4 pts]* Prove that a connected acyclic undirected graph with V vertices has exactly $V-1$ edges. ("It's a tree!" is not a proof.)

   (b) *[4 pts]* Describe and analyze an algorithm that determines whether a given undirected graph is a tree, where the graph is represented by an adjacency list.

   (c) *[2 pts]* What is the running time of your algorithm from part (b) if the graph is represented by an adjacency matrix?

3. Suppose we want to sketch the Manhattan skyline (minus the interesting bits like the Empire State and Chrysler builings). You are given a set of n rectangles, each rectangle represented by its left and right x-coordinates and its height. The bottom of each rectangle is on the x-axis. Describe and analyze an efficient algorithm to compute the vertices of the skyline.



A set of rectangles and its skyline. Compute the sequence of white points.

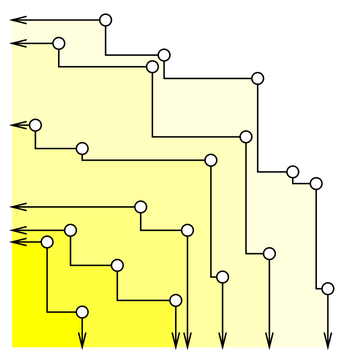
4. Suppose we model a computer network as a weighted undirected graph, where each vertex represents a computer and each edge represents a *direct* network connection between two computers. The weight of each edge represents the *bandwidth* of that connection—the number of bytes that can flow from one computer to the other in one second.[1] We want to implement a point-to-point network protocol that uses a single dedicated path to communicate between any pair of computers. Naturally, when two computers need to communciate, we should use the path with the highest bandwidth. The bandwidth of a *path* is the *minimum* bandwidth of its edges.

   Describe an algorithm to compute the maximum bandwidth path between *every* pair of computers in the network. Assume that the graph is represented as an adjacency list.

5. *[1-unit grad students must answer this question.]*

   Let P be a set of points in the plane. Recall that the *staircase* of P contains all the points in P that have no other point in P both above and to the right. We can define the *staircase layers* of P recursively as follows. The first staircase layer is just the staircase; for all $i > 1$, the ith staircase layer is the staircase of P after the first $i - 1$ staircase layers have been deleted.

   Describe and analyze an algorithm to compute the staircase layers of P in $O(n^2)$ time.[2] Your algorithm should label each point with an integer describing which staircase layer it belongs to. You can assume that no two points have the same x- or y-coordinates.



A set of points and its six staircase layers.

[1]Notice the bandwidth is symmetric; there are no cable modems or wireless phones. Don't worry about systems-level stuff like network load and latency. After all, this is a theory class!

[2]This is *not* the fastest possible running time for this problem.

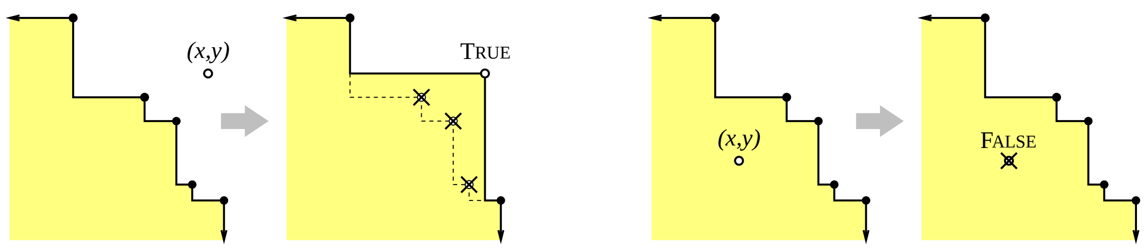
You must turn in this question sheet with your answers.

1. **Déjà vu**

   Prove that any positive integer can be written as the sum of distinct *nonconsecutive* Fibonacci numbers—if F_n appears in the sum, then neither F_{n+1} nor F_{n-1} will. For example: $42 = F_9 + F_6$, $25 = F_8 + F_4 + F_2$, and $17 = F_7 + F_4 + F_2$. *You must give a complete, self-contained proof, not just a reference to the posted homework solutions.*

2. **L'esprit d'escalier**

   Recall that the *staircase* of a set of points consists of the points with no other point both above and to the right. Describe a method to maintain the staircase as new points are added to the set. Specifically, describe and analyze a data structure that stores the staircase of a set of points, and an algorithm INSERT(x, y) that adds the point (x, y) to the set and returns TRUE or FALSE to indicate whether the staircase has changed. Your data structure should use $O(n)$ space, and your INSERT algorithm should run in $O(\log n)$ amortized time.



3. **Engage le jeu que je le gagne**

   A palindrome is a text string that is exactly the same as its reversal, such as DEED, RACECAR, or SAIPPUAKAUPPIAS.[1]

   (a) Describe and analyze an algorithm to find the longest *prefix* of a given string that is also a palindrome. For example, the longest palindrome prefix of ILLINOISURBANACHAMPAIGN is ILLI, and the longest palindrome prefix of HYAKUGOJYUUICHI[2] is the single letter S. For full credit, your algorithm should run in $O(n)$ time.

   (b) Describe and analyze an algorithm to find the length of the longest *subsequence* of a given string that is also a palindrome. For example, the longest palindrome subsequence of ILLINOISURBANACHAMPAIGN is NIAACAAIN (or NIAAHAAIN), and the longest palindrome subsequence of HYAKUGOJYUUICHI is HUUUH[3] (or HUGUH or HUYUH or...). You do not need to compute the actual subsequence; just its length. For full credit, your algorithm should run in $O(n^2)$ time.

[1]Finnish for 'soap dealer'.
[2]Japanese for 'one hundred fifty-one'.
[3]English for 'What the heck are you talking about?'

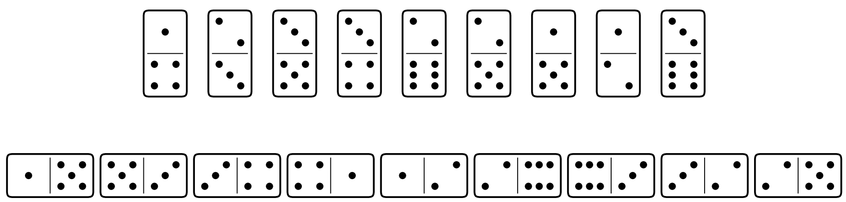
4. **Toute votre base sont appartiennent à nous**

   Prove that *exactly* $2n - 1$ comparisons are required in the worst case to merge two sorted arrays, each with n distinct elements. Describe and analyze an algorithm to prove the upper bound, and use an adversary argument to prove the lower bound. *You must give a complete, self-contained solution, not just a reference to the posted homework solutions.*[4]

5. **Plus ça change, plus ça même chose**

   A domino is a 2×1 rectangle divided into two squares, with a certain number of pips (dots) in each square. In most domino games, the players lay down dominos at either end of a single chain. Adjacent dominos in the chain must have matching numbers. (See the figure below.)

   Describe and analyze an efficient algorithm, or prove that it is NP-hard, to determine whether a given set of n dominos can be lined up in a single chain. For example, for the set of dominos shown below, the correct output is TRUE.



Top: A set of nine dominos
Bottom: The entire set lined up in a single chain

6. **Ceci n'est pas une pipe**

   Consider the following pair of problems:

   - BOXDEPTH: Given a set of n axis-aligned rectangles in the plane and an integer k, decide whether any point in the plane is covered by k or more rectangles.
   - MAXCLIQUE: Given a graph with n vertices and an integer k, decide whether the graph contains a clique with k or more vertices.

   (a) Describe and analyze a reduction of one of these problems to the other.

   (b) MAXCLIQUE is NP-hard. What does your answer to part (a) imply about the complexity of BOXDEPTH?

7. **C'est magique!** *[1-unit graduate students must answer this question.]*

   The recursion fairy's cousin, the reduction genie, shows up one day with a magical gift for you—a box that determines in constant time the size of the largest clique in any given graph. (Recall that a clique is a subgraph where every pair of vertices is joined by an edge.) The magic box does not tell you *where* the largest clique is, only its size. Describe and analyze an algorithm to actually find the largest clique in a given graph **in polynomial time**, using this magic box.

[4]The posted solution for this Homework 5 practice problem was incorrect. So don't use it!

CS 373: Combinatorial Algorithms, Fall 2002

Homework 0, due September 5, 2002 at the beginning of class

Name:		
Net ID:	Alias:	U G

Neatly print your name (first name first, with no comma), your network ID, and an alias of your choice into the boxes above. Circle U if you are an undergraduate, and G if you are a graduate student. **Do not sign your name. Do not write your Social Security number.** Staple this sheet of paper to the top of your homework.

Grades will be listed on the course web site by alias give us, so your alias should not resemble your name or your Net ID. If you don't give yourself an alias, we'll give you one that you won't like.

Before you do anything else, please read the Homework Instructions and FAQ on the CS 373 course web page (http://www-courses.cs.uiuc.edu/~cs373/hwx/faq.html) and then check the box below. There are 300 students in CS 373 this semester; we are quite serious about giving zeros to homeworks that don't follow the instructions.

☐ I have read the CS 373 Homework Instructions and FAQ.

Every CS 373 homework has the same basic structure. There are six required problems, some with several subproblems. Each problem is worth 10 points. Only graduate students are required to answer problem 6; undergraduates can turn in a solution for extra credit. There are several practice problems at the end. Stars indicate problems we think are hard.

This homework tests your familiarity with the prerequisite material from CS 173, CS 225, and CS 273, primarily to help you identify gaps in your knowledge. **You are responsible for filling those gaps on your own.** Rosen (the 173/273 textbook), CLRS (especially Chapters 1–7, 10, 12, and A–C), and the lecture notes on recurrences should be sufficient review, but you may want to consult other texts as well.

Required Problems

1. Sort the following functions from asymptotically smallest to asymptotically largest, indicating ties if there are any. Please *don't* turn in proofs, but you should do them anyway to make sure you're right (and for practice).

$$\begin{array}{ccccc} 1 & n & n^2 & \lg n & n \lg n \\ n^{\lg n} & (\lg n)^n & (\lg n)^{\lg n} & n^{\lg \lg n} & n^{1/\lg n} \\ \log_{1000} n & \lg^{1000} n & \lg^{(1000)} n & \lg(n^{1000}) & \left(1 + \frac{1}{1000}\right)^n \end{array}$$

   To simplify notation, write $f(n) \ll g(n)$ to mean $f(n) = o(g(n))$ and $f(n) \equiv g(n)$ to mean $f(n) = \Theta(g(n))$. For example, the functions n^2, n, $\binom{n}{2}$, n^3 could be sorted either as $n \ll n^2 \equiv \binom{n}{2} \ll n^3$ or as $n \ll \binom{n}{2} \equiv n^2 \ll n^3$.

2. Solve these recurrences. State tight asymptotic bounds for each function in the form $\Theta(f(n))$ for some recognizable function $f(n)$. Please *don't* turn in proofs, but you should do them anyway just for practice. Assume reasonable but nontrivial base cases, and state them if they affect your solution. Extra credit will be given for more exact solutions. *[Hint: Most of these are very easy.]*

$$\begin{array}{ll} A(n) = 2A(n/2) + n & F(n) = 9F(\lfloor n/3 \rfloor + 9) + n^2 \\ B(n) = 3B(n/2) + n & G(n) = 3G(n-1)/5G(n-2) \\ C(n) = 2C(n/3) + n & H(n) = 2H(\sqrt{n}) + 1 \\ D(n) = 2D(n-1) + 1 & I(n) = \min\limits_{1 \le k \le n/2} \big(I(k) + I(n-k) + k\big) \\ E(n) = \max\limits_{1 \le k \le n/2} \big(E(k) + E(n-k) + n\big) & {}^*J(n) = \max\limits_{1 \le k \le n/2} \big(J(k) + J(n-k) + k\big) \end{array}$$

3. Recall that a binary tree is *full* if every node has either two children (an internal node) or no children (a leaf). Give at least *four different* proofs of the following fact:

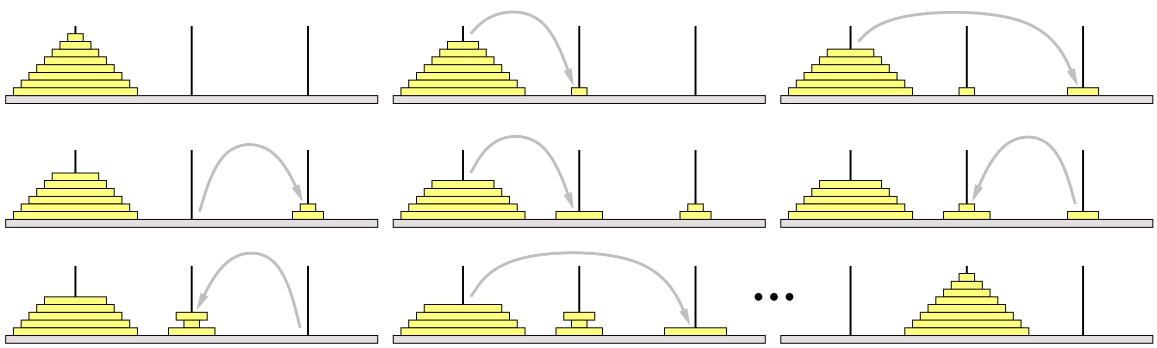
   In any full binary tree, the number of leaves is exactly one more than the number of internal nodes.

   For full credit, each proof must be self-contained, the proof must be substantially different from each other, and at least one proof must *not* use induction. For each n, your nth correct proof is worth n points, so you need four proofs to get full credit. Each correct proof beyond the fourth earns you extra credit. *[Hint: I know of at least six different proofs.]*

4. Most of you are probably familiar with the story behind the Tower of Hanoï puzzle:[1]

> At the great temple of Benares, there is a brass plate on which three vertical diamond shafts are fixed. On the shafts are mounted n golden disks of decreasing size.[2] At the time of creation, the god Brahma placed all of the disks on one pin, in order of size with the largest at the bottom. The Hindu priests unceasingly transfer the disks from peg to peg, one at a time, never placing a larger disk on a smaller one. When all of the disks have been transferred to the last pin, the universe will end.

Recently the temple at Benares was relocated to southern California, where the monks are considerably more laid back about their job. At the "Towers of Hollywood", the golden disks were replaced with painted plywood, and the diamond shafts were replaced with Plexiglas. More importantly, the restriction on the order of the disks was relaxed. While the disks are being moved, the *bottom* disk on any pin must be the *largest* disk on that pin, but disks further up in the stack can be in any order. However, after all the disks have been moved, they must be in sorted order again.



The Towers of Hollywood.

Describe an algorithm[3] that moves a stack of n disks from one pin to the another using the smallest possible number of moves. For full credit, your algorithm should be non-recursive, but a recursive algorithm is worth significant partial credit. *Exactly* how many moves does your algorithm perform? *[Hint: The Hollywood monks can bring about the end of the universe quite a bit faster than the original monks at Benares could.]*

The problem of computing the minimum number of moves was posed in the most recent issue of the *American Mathematical Monthly* (August/September 2002). No solution has been published yet.

[1] The puzzle and the accompanying story were both invented by the French mathematician Eduoard Lucas in 1883. See http://www.cs.wm.edu/~pkstoc/toh.html

[2] In the original legend, $n = 64$. In the 1883 wooden puzzle, $n = 8$.

[3] Since you've read the Homework Instructions, you know exactly what this phrase means.

5. On their long journey from Denmark to England, Rosencrantz and Guildenstern amuse themselves by playing the following game with a fair coin. First Rosencrantz flips the coin over and over until it comes up tails. Then Guildenstern flips the coin over and over until he gets as many heads in a row as Rosencrantz got on his turn. Here are three typical games:

   Rosencrantz: H H T
   Guildenstern: H T H H

   Rosencrantz: T
   Guildenstern: (no flips)

   Rosencrantz: H H H T
   Guildenstern: T H H T H H T H T H H H

   (a) What is the expected number of flips in one of Rosencrantz's turns?

   (b) Suppose Rosencrantz flips k heads in a row on his turn. What is the expected number of flips in Guildenstern's next turn?

   (c) What is the expected total number of flips (by both Rosencrantz and Guildenstern) in a single game?

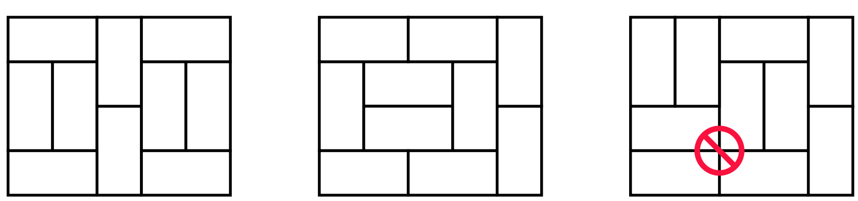
   Prove your answers are correct. If you have to appeal to "intuition" or "common sense", your answer is almost certainly wrong! You must give exact answers for full credit, but asymptotic bounds are worth significant partial credit.

6. *[This problem is required only for graduate students (including I2CS students); undergrads can submit a solution for extra credit.]*

   Tatami are rectangular mats used to tile floors in traditional Japanese houses. Exact dimensions of tatami mats vary from one region of Japan to the next, but they are always twice as long in one dimension than in the other. (In Tokyo, the standard size is 180cm×90cm.)

   (a) How many different ways are there to tile a $2 \times n$ rectangular room with 1×2 tatami mats? Set up a recurrence and derive an *exact* closed-form solution. *[Hint: The answer involves a familiar recursive sequence.]*

   (b) According to tradition, tatami mats are always arranged so that four corners never meet. Thus, the first two arrangements below are traditional, but not the third.



Two traditional tatami arrangements and one non-traditional arrangement.

   How many different *traditional* ways are there to tile a $3 \times n$ rectangular room with 1×2 tatami mats? Set up a recurrence and derive an *exact* closed-form solution.

   *(c) **[5 points extra credit]** How many different *traditional* ways are there to tile an $n \times n$ square with 1×2 tatami mats? Prove your answer is correct.

Practice Problems

These problems are only for your benefit; other problems can be found in previous semesters' homeworks on the course web site. You are *strongly* encouraged to do some of these problems as additional practice. Think of them as potential exam questions (hint, hint). Feel free to ask about any of these questions on the course newsgroup, during office hours, or during review sessions.

1. Removing any edge from a binary tree with n nodes partitions it into two smaller binary trees. If both trees have at least $\lceil (n-1)/3 \rceil$ nodes, we say that the partition is *balanced*.

   (a) Prove that every binary tree with more than one vertex has a balanced partition. *[Hint: I know of at least two different proofs.]*

   (b) If each smaller tree has more than $\lfloor n/3 \rfloor$ nodes, we say that the partition is *strictly balanced*. Show that for every n, there is an n-node binary tree with *no* strictly balanced partition.

2. Describe an algorithm $\textsc{CountToTenToThe}(n)$ that prints the integers from 1 to 10^n.

   Assume you have a subroutine $\textsc{PrintDigit}(d)$ that prints any integer d between 0 and 9, and another subroutine $\textsc{PrintSpace}$ that prints a space character. Both subroutines run in $O(1)$ time. You may want to write (and analyze) a separate subroutine $\textsc{PrintInteger}$ to print an arbitrary integer.

   Since integer variables cannot store arbitrarily large values in most programming languages, your algorithm must not store any value larger than $\max\{10, n\}$ in any single integer variable. Thus, the following algorithm is **not correct**:

```
BOGUSCOUNTTOTENTOTHE(n):
  for i ← 1 to POWER(10, n)
      PRINTINTEGER(i)
```

   (So what exactly *can* you pass to $\textsc{PrintInteger}$?)

   What is the running time of your algorithm (as a function of n)? How many digits and spaces does it print? How much space does it use?

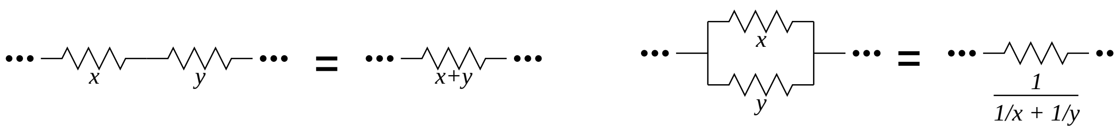
3. I'm sure you remember the following simple rules for taking derivatives:

   - Simple cases: $\frac{d}{dx}\alpha = 0$ for any constant α, and $\frac{d}{dx}x = 1$
   - Linearity: $\frac{d}{dx}(f(x) + g(x)) = f'(x) + g'(x)$
   - The product rule: $\frac{d}{dx}(f(x) \cdot g(x)) = f'(x) \cdot g(x) + f(x) \cdot g'(x)$
   - The chain rule: $\frac{d}{dx}(f(g(x)) = f'(g(x)) \cdot g'(x)$

   Using *only* these rules and induction, prove that $\frac{d}{dx}x^c = cx^{c-1}$ for any integer $c \neq -1$. Do not use limits, integrals, or any other concepts from calculus, except for the simple identities listed above. *[Hint: Don't forget about negative values of c!]*

4. This problem asks you to calculate the total resistance between two points in a series-parallel resistor network. Don't worry if you haven't taken a circuits class; everything you need to know can be summed up in two sentences and a picture.

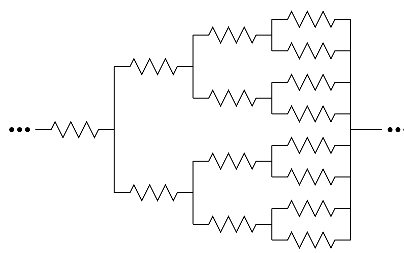
   - The total resistance of two resistors in *series* is the sum of their individual resistances.
   - The total resistance of two resistors in *parallel* is the reciprocal of the sum of the reciprocals of their individual resistances.



Equivalence laws for series-parallel resistor networks.

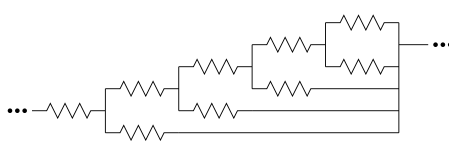
What is the *exact* total resistance[4] of the following resistor networks as a function of n? Prove your answers are correct. *[Hint: Use induction. Duh.]*

(a) A complete binary tree with depth n, with a 1Ω resistor at every node, and a common wire joining all the leaves. Resistance is measured between the root and the leaves.



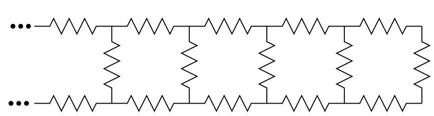
A balanced binary resistor tree with depth 3.

(b) A totally unbalanced full binary tree with depth n (every internal node has two children, one of which is a leaf) with a 1Ω resistor at every node, and a common wire joining all the leaves. Resistance is measured between the root and the leaves.



A totally unbalanced binary resistor tree with depth 4.

*(c) A ladder with n rungs, with a 1Ω resistor on every edge. Resistance is measured between the bottom of the legs.



A resistor ladder with 5 rungs.

[4] The ISO standard unit of resistance is the Ohm, written with the symbol Ω. Don't confuse this with the asymptotic notation $\Omega(f(n))$!

CS 373: Combinatorial Algorithms, Fall 2002
Homework 1, due September 17, 2002 at 23:59:59

Name:	
Net ID:	Alias:
Name:	
Net ID:	Alias:
Name:	
Net ID:	Alias:
Undergrads	Grads

This homework is to be submitted in groups of up to three people. Graduate and undergraduate students are *not* allowed to work in the same group. Please indicate above whether you are undergraduate or graduate students. Only *one* submission per group will be accepted.

Required Problems

1. The traditional Devonian/Cornish drinking song "The Barley Mow" has the following pseudolyrics, where *container*[i] is the name of a container [1] that holds 2^i ounces of beer.

```
BarleyMow(n):
    "Here's a health to the barley-mow, my brave boys,"
    "Here's a health to the barley-mow!"

    "We'll drink it out of the jolly brown bowl,"
    "Here's a health to the barley-mow!"
    "Here's a health to the barley-mow, my brave boys,"
    "Here's a health to the barley-mow!"

    for i ← 1 to n
        "We'll drink it out of the container[i], boys,"
        "Here's a health to the barley-mow!"
        for j ← i downto 1
            "The container[j],"
        "And the jolly brown bowl!"
        "Here's a health to the barley-mow, my brave boys,"
        "Here's a health to the barley-mow!"
```

   (a) Suppose each container name *container*[i] is a single word, and you can sing four words a second. How long would it take you to sing BarleyMow(n)? (Give a tight asymptotic bound.)

[1] One version of the song uses the following containers: nipperkin, gill pot, half-pint, pint, quart, pottle, gallon, half-anker, anker, firkin, half-barrel, barrel, hogshead, pipe, well, river, and ocean. Every container in this list is twice as big as its predecessor, except that a firkin is actually 2.25 ankers, and the last three units are just silly.

(b) Suppose *container*$[n]$ has $\Theta(\log n)$ syllables, and you can sing six syllables per second. Now how long would it take you to sing BARLEYMOW(n)? (Give a tight asymptotic bound.)

(c) Suppose each time you mention the name of a container, you drink the corresponding amount of beer: one ounce for the jolly brown bowl, and 2^i ounces for each *container*$[i]$. Assuming for purposes of this problem that you are over 21, *exactly* how many ounces of beer would you drink if you sang BARLEYMOW(n)? (Give an *exact* answer, not just an asymptotic bound.)

2. Suppose you have a set S of n numbers. Given two elements you *cannot* determine which is larger. However, you are given an oracle that will tell you the median of a set of three elements.

   (a) Give a linear time algorithm to find the pair of the largest and smallest numbers in S.

   (b) Give an algorithm to sort S in $O(n \lg n)$ time.

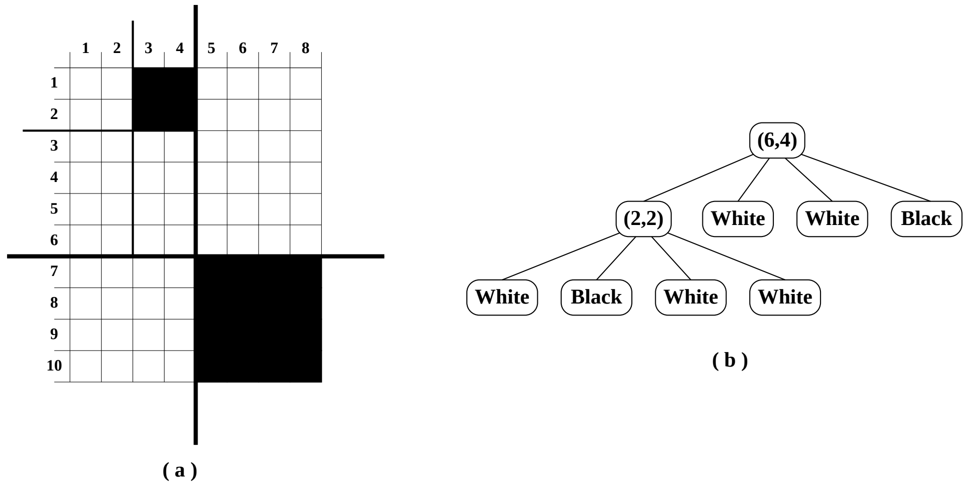
3. Given a black and white pixel image $A[1 \ldots m][1 \ldots n]$, our task is to represent A with a search tree T. Given a query (x, y), a simple search on T should return the color of pixel $A[x][y]$. The algorithm to construct T will be as follows.

```
ConstructSearchTree(A[1...m][1...n]):
    //Base Case
    if A contains only one color
        return a leaf node labeled with that color

    //Recurse on Subtrees
    (i, j) ← ChooseCut(A[1...m][1...n])
    T1 ← ConstructSearchTree(A[1...i][1...j])
    T2 ← ConstructSearchTree(A[1...i][j+1...n])
    T3 ← ConstructSearchTree(A[i+1...m][1...j])
    T4 ← ConstructSearchTree(A[i+1...m][j+1...n])

    //Construct the Root
    T.cut ← (i, j)
    T.children ← T1, T2, T3, T4
    return T
```

That is, this algorithm divides a multicolor image into quadrants and recursively constructs the search tree for each quadrant. Upon a query (x, y) of T (assuming $1 \leq x \leq m$ and $1 \leq y \leq n$), the appropriate subtree is searched. When the correct leaf node is reached, the pixel color is returned. Here's a toy example.

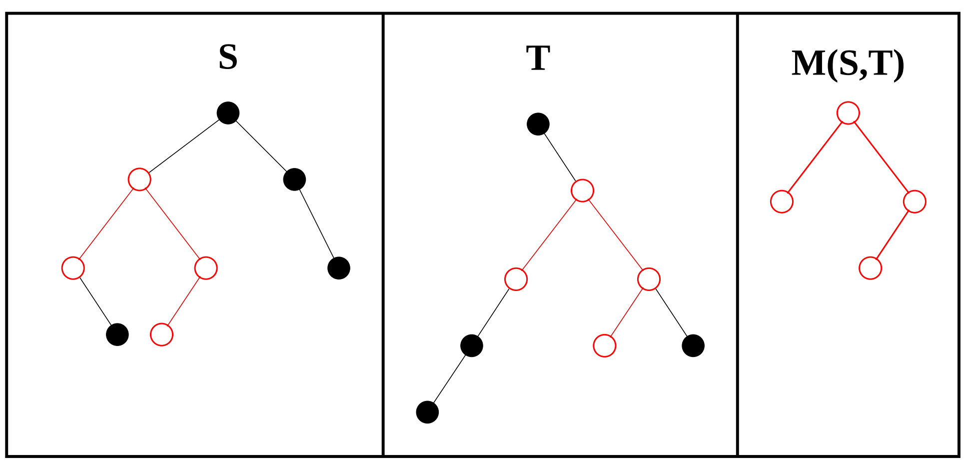


(a) An image A and the chosen cuts. (b) The corresponding search tree.

Your job in this problem is to give an algorithm for ChooseCut. The sequence of chosen cuts must result in an optimal search tree T. That is, the expected search depth of a uniformly chosen pixel must be minimized. You may use any external data structures ($i.e.$a global table) that you find necessary. You may also preprocess in order to initialize these structures before the initial call to ConstructSearchTree($A[1 \ldots m][1 \ldots n]$).

4. Let A be a set of n positive integers, all of which are no greater than some constant $M > 0$. Give an $O(n^2 M)$ time algorithm to determine whether or not it is possible to split A into two subsets such that the sum of the numbers in each subset are equal.

5. Let S and T be two binary trees. A *matching* of S and T is a tree M which is isomorphic to some subtree in each of S and T. Here's an illustration.



A matching $M(S,T)$ of binary trees S and T.

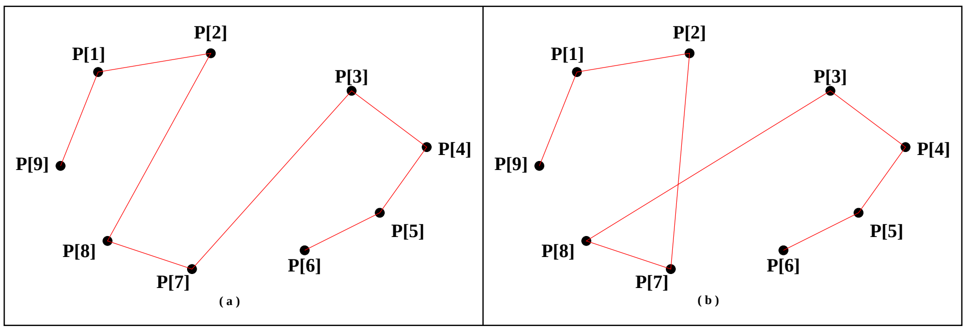
A *maximal* matching is a matching which contains at least as many vertices as any other matching. Give an algorithm to compute a maximal matching given the roots of two binary trees. Your algorithm should return the size of the match as well as the two roots of the matched subtrees of S and T.

6. *[This problem is required only for graduate students (including I2CS students); undergrads can submit a solution for extra credit.]*

   Let $P[1,\ldots,n]$ be a set of n *convex* points in the plane. Intuitively, if a rubber band were stretched to surround P then each point would touch the rubber band. Furthermore, suppose that the points are labeled such that $P[1],\ldots,P[n]$ is a simple path along the convex hull ($i.e.P[i]$ is adjacent to $P[i+1]$ along the rubber band).

   (a) Give a simple algorithm to compute a shortest *cyclic* tour of P.

   (b) A *monotonic* tour of P is a tour that never crosses itself. Here's an illustration.

   (a) A monotonic tour of P. (b) A non-monotonic tour of P.

   Prove that any shortest tour of P must be monotonic.

   (c) Given an algorithm to compute a shortest tour of P starting at point $P[1]$ and finishing on point $P[\lfloor \frac{n}{2} \rfloor]$.

Practice Problems

These remaining practice problems are entirely for your benefit. Don't turn in solutions—we'll just throw them out—but feel free to ask us about these questions during office hours and review sessions. Think of these as potential exam questions (hint, hint).

1. Suppose that you are given an $n \times n$ checkerboard and a checker. You must move the checker from the bottom edge of the board to the top edge of the board according to the following rule. At each step you may move the checker to one of three squares:

   (a) the square immediately above,

   (b) the square that is one up and one left (but only if the checker is not already in the leftmost column),

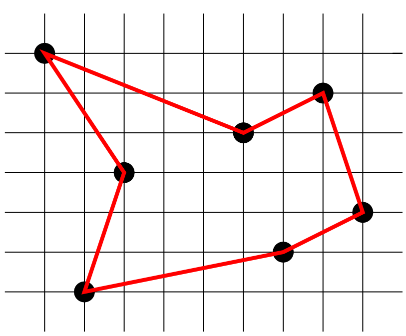
   (c) the square that is one up and one right (but only if the checker is not already in the rightmost column).

   Each time you move from square x to square y, you receive $p(x, y)$ dollars. You are given $p(x, y)$ for all pairs (x,y) for which a move from x to y is legal. Do not assume that $p(x, y)$ is positive.

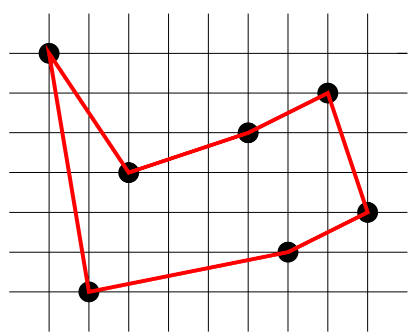
   Give an algorithm that figures out the set of moves that will move the checker from somewhere along the bottom edge to somewhere along the top edge while gathering as many dollars as possible. Your algorithm is free to pick any square along the bottom edge as a starting point and any square along the top edge as a destination in order to maximize the number of dollars gathered along the way. What is the running time of your algorithm?

2. (CLRS 15-1) The *euclidean traveling-salesman problem* is the problem of determining the shortest closed tour that connects a given set of n points in the plane. Figure (a) below shows the solution to a 7-point problem. The general problem is NP-complete, and its solution is therefore believed to require more than polynomial time.

   J.L. Bentley has suggested that we simplify the problem by restricting our attention to *bitonic tours* (Figure (b) below). That is, tours that start at the leftmost point, go strictly left to right to the rightmost point, and then go strictly right to left back to the starting point. In this case, a polynomial-time algorithm is possible.



(a)



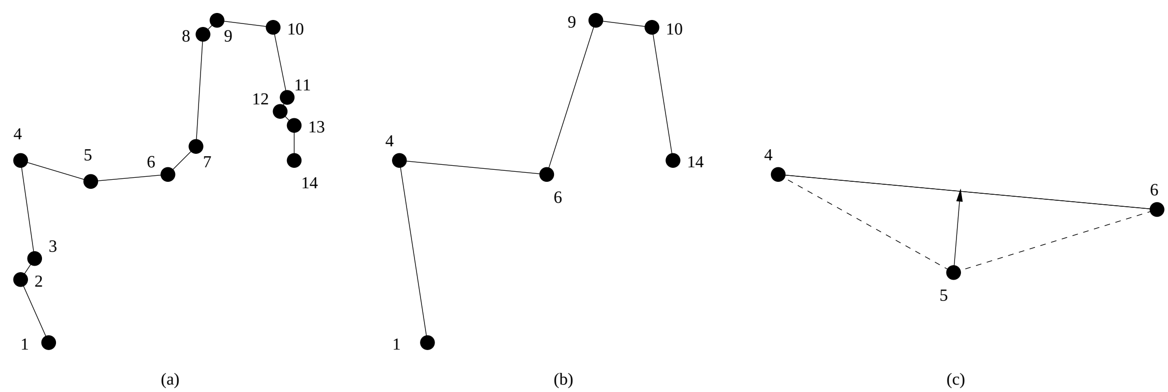
(b)

Seven points in the plane, shown on a unit grid. (a) The shortest closed tour, with length approximately 24.89. This tour is not bitonic. (b) The shortest bitonic tour for the same set of points. It's length is approximately 25.58.

Describe an $O(n^2)$-time algorithm for determining an optimal bitonic tour. You may assume that no two points have the same x-coordinate. [Hint: Scan left to right, maintaining optimal possibilities for the two parts of the tour.]

3. You are given a polygonal line γ made out of n vertices in the plane. Namely, you are given a list of n points in the plane $p_1, \ldots, p_n$, where $p_i = (x_i, y_i)$. You need to display this polygonal line on the screen, however, you realize that you might be able to draw a polygonal line with considerably less vertices that looks identical on the screen (because of the limited resolution of the screen). It is crucial for you to minimize the number of vertices of the polygonal line. (Because, for example, your display is a remote Java applet running on the user computer, and for each vertex of the polygon you decide to draw, you need to send the coordinates of the points through the network which takes a *long long long time*. So the fewer vertices you send, the snappier your applet would be.)

   So, given such a polygonal line γ, and a parameter k, you would like to select k vertices of γ that yield the "best" polygonal line that looks like γ.



(a) The original polygonal line with 14 vertices. (b) A new polygonal line with 6 vertices. (c) The distance between p_5 on the original polygonal line and the simplification segment p_4p_6. The error of p_5 is $\mathrm{error}(p_5) = \mathrm{dist}(p_5, p_4p_6)$.

   Namely, you need to build a new polygonal line γ' and minimize the difference between the two polygonal-lines. The polygonal line γ' is built by selecting k vertices $\{p_{i_1}, p_{i_2}, \ldots, p_{i_k}\}$ from γ. It is required that $i_1 = 1$, $i_k = n$, and $i_j < i_{j+1}$ for $j = 1, 2, \ldots, k-1$.

   We define the error between γ and γ' by how far from γ' are the vertices of γ. More formally, The difference between the two polygonal lines is

$$error(\gamma, \gamma') = \sum_{j=1}^{k-1} \sum_{m=i_j+1}^{i_{j+1}-1} \mathrm{dist}(p_m, p_{i_j} p_{i_{j+1}}).$$

   Namely, for every vertex not in the simplification, its associated error, is the distance to the corresponding simplified segment (see (c) in above figure). The overall error is the sum over all vertices.

   You can assume that you are provided with a subroutine that can calculate $\mathrm{dist}(u, vw)$ in constant time, where $\mathrm{dist}(u, vw)$ is the distance between the point u and the segment vw.

   Give an $O(n^3)$ time algorithm to find the γ' that minimizes $\mathrm{error}(\gamma, \gamma')$.

CS 373: Combinatorial Algorithms, Fall 2002
Homework 2 (due Thursday, September 26, 2002 at 11:59:59 p.m.)

Name:		
Net ID:	Alias:	U G

Name:		
Net ID:	Alias:	U G

Name:		
Net ID:	Alias:	U G

Homeworks may be done in teams of up to three people. Each team turns in just one solution, and every member of a team gets the same grade. Since graduate students are required to solve problems that are worth extra credit for other students, **Grad students may not be on the same team as undergraduates.**

Neatly print your name(s), NetID(s), and the alias(es) you used for Homework 0 in the boxes above. Please also tell us whether you are an undergraduate or 1-unit grad student by circling U or G, respectively. Staple this sheet to the top of your homework. **NOTE: You must use different sheet(s) of paper for each problem assigned.**

Required Problems

1. For each of the following problems, the input is a set of n nuts and n bolts. For each bolt, there is exactly one nut of the same size. Direct comparisons between nuts or between bolts are not allowed, but you can compare a nut and a bolt in constant time.

   (a) Describe and analyze a deterministic algorithm to find the largest bolt. *Exactly* how many comparisons does your algorithm perform in the worst case? *[Hint: This is very easy.]*

   (b) Describe and analyze a randomized algorithm to find the largest bolt. What is the *exact* expected number of comparisons performed by your algorithm?

   (c) Describe and analyze an algorithm to find the largest and smallest bolts. Your algorithm can be either deterministic or randomized. What is the *exact* worst-case expected number of comparisons performed by your algorithm? *[Hint: Running part (a) twice is definitely not the most efficient algorithm.]*

   In each case, to receive **full** credit, you need to describe the most efficient algorithm possible.

2. Consider the following algorithm:

```
SLOWSHUFFLE(A[1..n]) :
    for i ← 1 to n
        B[i] ←Null
    for i ← 1 to n
        index ← Random(1,n)
        while B[index] ≠ Null
            index ← Random(1,n)
        B[index] ← A[i]
    for i ← 1 to n
        A[i] ← B[i]
```

Suppose that Random(i,j) will return a random number between i and j inclusive in constant time. SLOWSHUFFLE will shuffle the input array into a random order such that every permutation is equally likely.

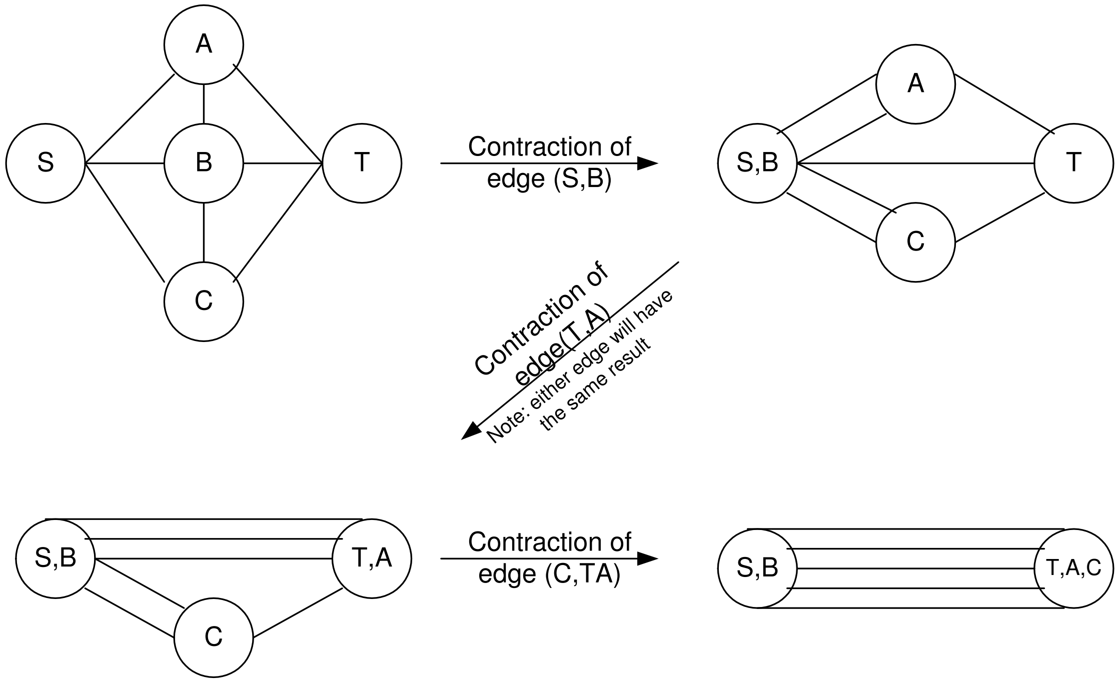
(a) What is the expected running time of the above algorithm. Justify your answer and give a tight asymptotic bound.

(b) Describe an algorithm that randomly shuffles an n-element array, so that every permutation is *equally* likely, in $O(n)$ time.

3. Suppose we are given an undirected graph $G = (V, E)$ together with two distinguished vertices s and t. An **s-t min-cut** is a set of edges that once removed from the graph, will disconnect s from t. We want to find such a set with the minimum cardinality(The smallest number of edges). In other words, we want to find the smallest set of edges that will seperate s and t

To do this we repeat the following step $|V| - 2$ times: Uniformly at random, pick an edge from the set E which contains all edges in the graph excluding those that directly connects vertices s and t. Merge the two vertices that is connected by this randomly selected edge. If as a result there are several edges between some pair of vertices, retain them all. Edges that are between the two merged vertices are removed so that there are never any self-loops. We refer to this process of merging the two end-points of an edge into a single vertex as the *contraction* of that edge. Notice with each contraction the number of vertices of G decreases by one.

As this algorithm proceeds, the vertex s may get merged with a new vertex as the result of an edge being contracted. We call this vertex the s-vertex. Similarly, we have a t-vertex. During the contraction algorithm, we ensure that we never contract an edge between the s-vertex and the t-vertex.



(a) Give an example of a graph in which the probability that this algorithm finds an s-t min-cut is exponentially small($O(1/a^n)$). Justify your answers.
(Hint: Think multigraphs)

(b) Give an example of a graph such that there are $O(2^n)$ number of s-t min-cuts. Justify your answers.

4. Describe a modification of treaps that supports the following operations, each in $O(\log n)$ expected time:

- INSERT(x): Insert a new element x into the data structure.
- DELETE(x): Delete an element x from the data structure.
- COMPUTERANK(x): Return the number of elements in the data structure less than or equal to x.
- FINDBYRANK(r): Return the kth smallest element in the data structure.

Describe and analyze the algorithms that implement each of these operations. *[Hint: Don't reinvent the wheel!]*

5. A *meldable priority queue* stores a set of keys from some totally ordered universe (such as the integers) and supports the following operations:

   - MakeQueue: Return a new priority queue storing the empty set.
   - FindMin(Q): Return the smallest element stored in Q (if any).
   - DeleteMin(Q): Delete the smallest element stored in Q (if any).
   - Insert(Q, x): Insert element x into Q.
   - Meld(Q_1, Q_2): Return a new priority queue containing all the elements stored in Q_1 and Q_2. The component priority queues are destroyed.
   - DecreaseKey(Q, x, y): Replace an element x of Q with a smaller key y. (If $y > x$, the operation fails.) The input is a pointer directly to the node in Q storing x.
   - Delete(Q, x): Delete an element $x \in Q$. The input is a pointer directly to the node in Q storing x.

   A simple way to implement this data structure is to use a heap-ordered binary tree, where each node stores an element, a pointer to its left child, a pointer to its right child, and a pointer to its parent. Meld(Q_1, Q_2) can be implemented with the following randomized algorithm.

   - If either one of the queues is empty, return the other one.
   - If the root of Q_1 is smaller than the root of Q_2, then recursively Meld Q_2 with either right(Q_1) or left(Q_1), each with probability 1/2.
   - Similarly, if the root of Q_2 is smaller than the root of Q_1, then recursively Meld Q_1 with a randomly chosen child of Q_2.

   (a) Prove that for *any* heap-ordered trees Q_1 and Q_2, the expected running time of Meld(Q_1, Q_2) is $O(\log n)$, where $n = |Q_1| + |Q_2|$. *[Hint: How long is a random path in an n-node binary tree, if each left/right choice is made with equal probability?]* For extra credit, prove that the running time is $O(\log n)$ *with high probability.*

   (b) Show that each of the operations DeleteMin, Insert, DecreaseKey, and Delete can be implemented with one call to Meld and $O(1)$ additional time. (This implies that every operation takes $O(\log n)$ with high probability.)

6. *[This problem is required only for graduate students taking CS 373 for a full unit; anyone else can submit a solution for extra credit.]*

   The following randomized algorithm selects the rth smallest element in an unsorted array $A[1,..,n]$. For example, to find the smallest element, you would call RandomSelect(A, 1); to find the median element, you would call RandomSelect($A, \lfloor n/2 \rfloor$). Recall from lecture that Partition splits the array into three parts by comparing the pivot element $A[p]$ to every other element of the array, using $n - 1$ comparisons altogether, and returns the new index of the pivot element.

```
RandomSelect(A[1 .. n], r) :
    p ← Random(1, n)
    k ← Partition(A[1 .. n], p)

    if r < k
        return RandomSelect(A[1 .. k − 1], r)
    else if r > k
        return RandomSelect(A[k + 1 .. n], r − k)
    else
        return A[k]
```

(a) State a recurrence for the expected running time of RandomSelect, as a function of both n and r.

(b) What is the *exact* probability that RandomSelect compares the ith smallest and jth smallest elements in the input array? The correct answer is a simple function of i, j, and r. *[Hint: Check your answer by trying a few small examples.]*

(c) Show that for any n and r, the expected running time of RandomSelect is $\Theta(n)$. You can use either the recurrence from part (a) or the probabilities from part (b). For extra credit, find the *exact* expected number of comparisons, as a function of n and r.

(d) What is the expected number of times that RandomSelect calls itself recursively?

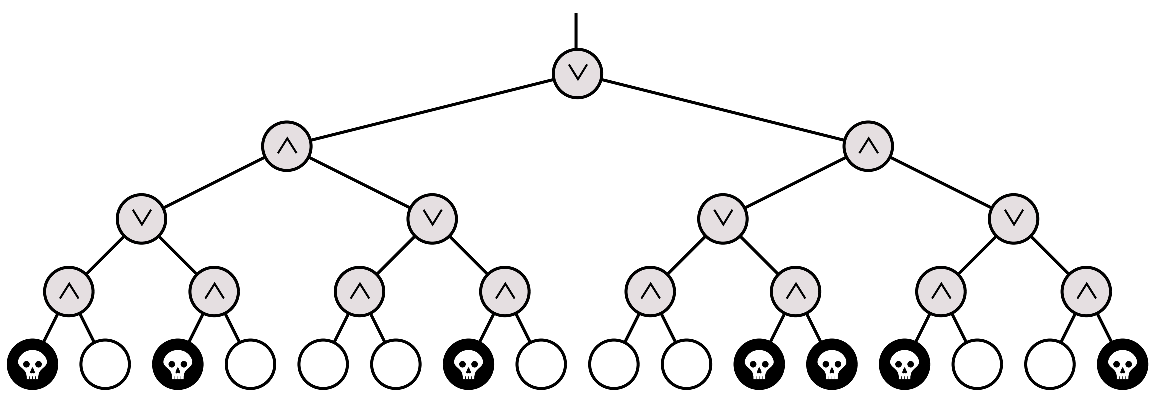
Practice Problems

1. Death knocks on your door one cold blustery morning and challenges you to a game. Death knows that you are an algorithms student, so instead of the traditional game of chess, Death presents you with a complete binary tree with 4^n leaves, each colored either black or white. There is a token at the root of the tree. To play the game, you and Death will take turns moving the token from its current node to one of its children. The game will end after $2n$ moves, when the token lands on a leaf. If the final leaf is black, you die; if it's white, you will live forever. You move first, so Death gets the last turn.

   You can decide whether it's worth playing or not as follows. Imagine that the nodes at even levels (where it's your turn) are OR gates, the nodes at odd levels (where it's Death's turn) are *and* gates. Each gate gets its input from its children and passes its output to its parent. White and black stand for TRUE and FALSE. If the output at the top of the tree is TRUE, then you can win and live forever! If the output at the top of the tree is FALSE, you should challenge Death to a game of Twister instead.

   (a) Describe and analyze a deterministic algorithm to determine whether or not you can win. *[Hint: This is easy!]*

   (b) Unfortunately, Death won't let you even look at every node in the tree. Describe a *randomized* algorithm that determines whether you can win in $\Theta(3^n)$ expected time. *[Hint: Consider the case $n = 1$.]*



2. WHat is the *exact* number of nodes in a skip list storing n keys, *not* counting the sentinel nodes at the beginning and end of each level? Justify your answer.

3. Suppose we are given two sorted arrays $A[1 \,..\, n]$ and $B[1 \,..\, n]$ and an integer k. Describe an algorithm to find the kth smallest element in the union of A and B. (For example, if $k = 1$, your algorithm should return the smallest element of $A \cup B$; if $k = n$, our algorithm should return the median of $A \cup B$.) You can assume that the arrays contain no duplicates. Your algorithm should be able to run in $\Theta(\log n)$ time. *[Hint: First try to solve the special case $k = n$.]*

CS 373: Combinatorial Algorithms, Fall 2002

Homework 3, due October 17, 2002 at 23:59:59

Name:	
Net ID:	Alias:
Name:	
Net ID:	Alias:
Name:	
Net ID:	Alias:
Undergrads	Grads

This homework is to be submitted in groups of up to three people. Graduate and undergraduate students are *not* allowed to work in the same group. Please indicate above whether you are undergraduate or graduate students. Only *one* submission per group will be accepted.

Required Problems

1. (a) Prove that only one subtree gets rebalanced in a scapegoat tree insertion.

   (b) Prove that $I(v) = 0$ in every node of a perfectly balanced tree. (Recall that $I(v) = \max\{0, |T| - |s| - 1\}$, where T is the child of greater height and s the child of lesser height, and $|v|$ is the number of nodes in subtree v. A perfectly balanced tree has two perfectly balanced subtrees, each with as close to half the nodes as possible.)

   *(c) Show that you can rebuild a fully balanced binary tree from an unbalanced tree in $O(n)$ time using only $O(\log n)$ additional memory.

2. Suppose we can insert or delete an element into a hash table in constant time. In order to ensure that our hash table is always big enough, without wasting a lot of memory, we will use the following global rebuilding rules:

   - After an insertion, if the table is more than 3/4 full, we allocate a new table twice as big as our current table, insert everything into the new table, and then free the old table.
   - After a deletion, if the table is less than 1/4 full, we we allocate a new table half as big as our current table, insert everything into the new table, and then free the old table.

   Show that for any sequence of insertions and deletions, the amortized time per operation is still a constant. Do *not* use the potential method (it makes it much more difficult).

3. A stack is a FILO/LIFO data structure that represents a stack of objects; access is only allowed at the top of the stack. In particular, a stack implements two operations:

   - PUSH(x): adds x to the top of the stack.

- POP: removes the top element and returns it.

A queue is a FIFO/LILO data structure that represents a row of objects; elements are added to the front and removed from the back. In particular, a queue implements two operations:

- ENQUEUE(x): adds x to the front of the queue.
- DEQUEUE: removes the element at the back of the queue and returns it.

Using two stacks and no more than O(1) additional space, show how to simulate a queue for which the operations ENQUEUE and DEQUEUE run in constant amortized time. You should treat each stack as a black box (i.e., you may call PUSH and POP, but you do not have access to the underlying stack implementation). Note that each PUSH and POP performed by a stack takes $O(1)$ time.

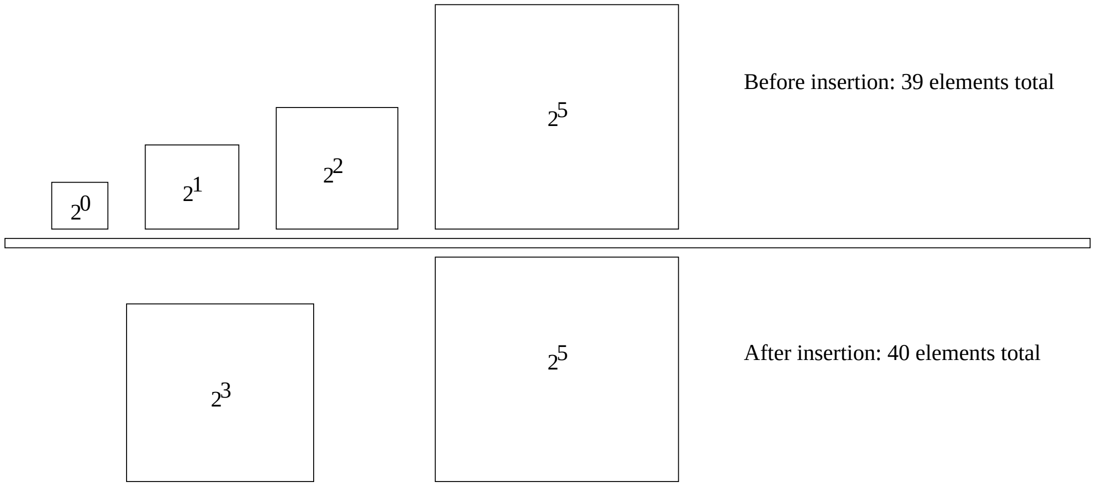
4. A data structure is *insertion-disabled* if there is no way to add elements to it. For the purposes of this problem, further assume that an insertion-disabled data structure implements the following operations with the given running times:

   - INITIALIZE(S): Return an insertion-disabled data structure that contains the elements of S. Running time: $O(n \log n)$.
   - SEARCH(D, x): Return TRUE if x is in D; return FALSE if not. Running time: $O(\log n)$.
   - RETURNALL(D, x): Return an unordered set of all elements in D. Running time: $O(n)$.
   - DELETE(D, x): Remove x from D if x is in D. Running time: $O(\log n)$.

   Using an approach known as the Bentley-Saxe Logarithmic Method (BSLM), it is possible to represent a dynamic (i.e., supports insertions) data structure with a collection of insertion-disabled data structures, where each insertion-disabled data structure stores a number of elements that is a distinct power of two. For example, to store $39 = 2^0 + 2^1 + 2^2 + 2^5$ elements in a BSLM data structure, we use four insertion-disabled data structures with 2^0, 2^1, 2^2, and 2^5 elements.

   To find an element in a BSLM data structure, we search the collection of insertion-disabled data structures until we find (or don't find) the element.

   To insert an element into a BSLM data structure, we think about adding a 2^0-size insertion-disabled data structure. However, an insertion-disabled data structure with 2^0 elements may already exist. In this case, we can combine two 2^0-size structures into a single 2^1-size structure. However, there may already be a 2^1-size structure, so we will need to repeat this process. In general, we do the following: Find the smallest i such that for all nonnegative $k < i$, there is a 2^k-sized structure in our collection. Create a 2^i-sized structure that contains the element to be inserted and all elements from 2^k-sized data structures for all $k < i$. Destroy all 2^k-sized data structures for $k < i$.



Inserting an element into a BSLM data structure.

We delete elements from the BSLM data structure lazily. To delete an element, we first search the collection of insertion-disabled data structures for it. Then we call DELETE to remove the element from its insertion-disabled data structure. This means that a 2^i-sized insertion-disabled data structure might store less than 2^i elements. That's okay; we just say that it stores 2^i elements and say that 2^i is its *pretend* size. We keep track of a single variable, called $Waste$, which is initially 0 and is incremented by 1 on each deletion. If $Waste$ exceeds three-quarters of the total pretend size of all insertion-disabled data structures in our collection (i.e., the total number of elements stored), we rebuild our collection of insertion-disabled data structures. In particular, we create a 2^m-sized insertion-disabled data structure, where 2^m is the smallest power that is greater than or equal to the total number of elements stored. All elements are stored in this 2^m-sized insertion-disabled data structure, and all other insertion-disabled data structures in our collection are destroyed. $Waste$ is reset to $2^m - n$, where n is the total number of elements stored in the BSLM data structure.

Your job is to prove the running times of the following three BSLM operations:

- SEARCHBSLM(D, x): Search for x in the collection of insertion-disabled data structures that represent the BSLM data structure D. Running time: $O(\log^2 n)$ worst-case.
- INSERTBSLM(D, x): Insert x into the collection of insertion-disabled data structures that represent the BSLM data structure D, modifying the collection as necessary. Running time: $O(\log^2 n)$ amortized.
- DELETEBSLM(D, x): Delete x from the collection of insertion-disabled data structures that represent the BSLM data structure D, rebuilding when there is a lot of wasted space. Running time: $O(\log^2 n)$ amortized.

5. Except as noted, the following sub-problems refer to a Union-Find data structure that uses both path compression and union by rank.

   (a) Prove that in a set of n elements, a sequence of n consecutive FIND operations takes $O(n)$ total time.

   (b) Show that any sequence of m MAKESET, FIND, and UNION operations takes only $O(m)$ time if all of the UNION operations occur before any of the FIND operations.

(c) Now consider part b with a Union-Find data structure that uses path compression but does *not* use union by rank. Is $O(m)$ time still correct? Prove your answer.

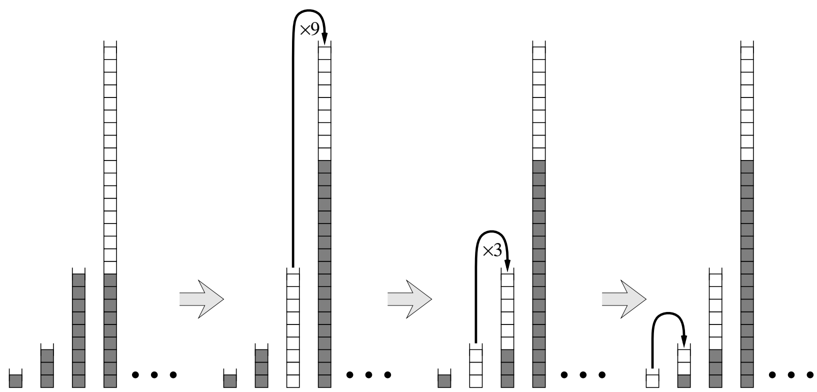
6. *[This problem is required only for graduate students (including I2CS students); undergrads can submit a solution for extra credit.]*

   Suppose instead of powers of two, we represent integers as the sum of Fibonacci numbers. In other words, instead of an array of bits, we keep an array of 'fits', where the ith least significant fit indicates whether the sum includes the ith Fibonacci number F_i. For example, the fit string 101110 represents the number $F_6 + F_4 + F_3 + F_2 = 8 + 3 + 2 + 1 = 14$. Describe algorithms to increment and decrement a fit string in constant amortized time. [Hint: Most numbers can be represented by more than one fit string.]

Practice Problems

These remaining practice problems are entirely for your benefit. Don't turn in solutions—we'll just throw them out—but feel free to ask us about these questions during office hours and review sessions. Think of these as potential exam questions (hint, hint).

1. A *multistack* consists of an infinite series of stacks $S_0, S_1, S_2, \ldots$, where the ith stack S_i can hold up to 3^i elements. Whenever a user attempts to push an element onto any full stack S_i, we first move all the elements in S_i to stack S_{i+1} to make room. But if S_{i+1} is already full, we first move all its members to S_{i+2}, and so on. Moving a single element from one stack to the next takes $O(1)$ time.



Making room for one new element in a multistack.

   (a) In the worst case, how long does it take to push one more element onto a multistack containing n elements?

   (b) Prove that the amortized cost of a push operation is $O(\log n)$, where n is the maximum number of elements in the multistack. You can use any method you like.

2. A hash table of size m is used to store n items with $n \leq m/2$. Open addressing is used for collision resolution.

   (a) Assuming uniform hashing, show that for $i = 1, 2, \ldots, n$, the probability that the i^{th} insertion requires strictly more than k probes is at most 2^{-k}.

   (b) Show that for $i = 1, 2, \ldots, n$, the probability that the i^{th} insertion requires more than $2 \lg n$ probes is at most $1/n^2$.

   Let the random variable X_i denote the number of probes required by the i^{th} insertion. You have shown in part (b) that $\Pr\{X_i > 2 \lg n\} \leq 1/n^2$. Let the random variable $X = \max_{1 \leq i \leq n} X_i$ denote the maximum number of probes required by any of the n insertions.

   (c) Show that $\Pr\{X > 2 \lg n\} \leq 1/n$.

   (d) Show that the expected length of the longest probe sequence is $E[X] = O(\lg n)$.

3. A sequence of n operations is performed on a data structure. The ith operation costs i if i is an exact power of 2, and 1 otherwise. That is operation i costs $f(i)$, where:

$$f(i) = \begin{cases} i, & i = 2^k, \\ 1, & \text{otherwise} \end{cases}$$

Determine the amortized cost per operation using the following methods of analysis:

(a) Aggregate method

(b) Accounting method

*(c) Potential method

CS 373: Combinatorial Algorithms, Fall 2002

Homework 4, due Thursday, October 31, 2002 at 23:59.99

Name:	
Net ID:	Alias:
Name:	
Net ID:	Alias:
Name:	
Net ID:	Alias:
Undergrads	Grads

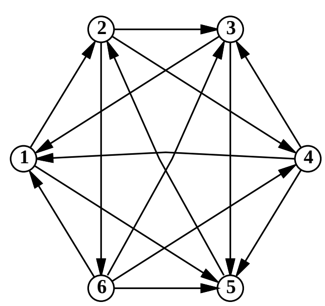
This homework is to be submitted in groups of up to three people. Graduate and undergraduate students are *not* allowed to work in the same group. Please indicate above whether you are undergraduate or graduate students. Only *one* submission per group will be accepted.

Required Problems

1. Tournament:

   A *tournament* is a directed graph with exactly one edge between every pair of vertices. (Think of the nodes as players in a round-robin tournament, where each edge points from the winner to the loser.) A *Hamiltonian path* is a sequence of directed edges, joined end to end, that visits every vertex exactly once.

   Prove that every tournament contains at least one Hamiltonian path.



A six-vertex tournament containing the Hamiltonian path $6 \to 4 \to 5 \to 2 \to 3 \to 1$.

2. Acrophobia:

   Consider a graph $G = (V, E)$ whose nodes are cities, and whose edges are roads connecting the cities. For each edge, the weight is assigned by h_e, the maximum altitude encountered when traversing the specified road. Between two cities s and t, we are interested in those paths whose maximum altitude is as low as possible. We will call a subgraph, G', of G an *acrophobic friendly subgraph*, if for any two nodes s and t the path of minimum altitude is always included in the subgraph. For simplicity, assume that the maximum altitude encountered on each road is unique.

(a) Prove that every graph of n nodes has an acrophobic friendly subgraph that has only $n-1$ edges.

(b) Construct an algorithm to find an acrophobic friendly subgraph given a graph $G = (V, E)$.

3. Refer to the lecture notes on single-source shortest paths. The GENERICSSSP algorithm described in class can be implemented using a stack for the 'bag'. Prove that the resulting algorithm, given a graph with n nodes as input, could perform $\Omega(2^n)$ relaxation steps before stopping. You need to describe, for any positive integer n, a specific weighted directed n-vertex graph that forces this exponential behavior. The easiest way to describe such a family of graphs is using an *algorithm*!

4. Neighbors:
Two spanning trees T and T' are defined as *neighbors* if T' can be obtained from T by swapping a single edge. More formally, there are two edges e and f such that T' is obtained from T by adding edge e and deleting edge f.

(a) Let T denote the minimum cost spanning tree and suppose that we want to find the second cheapest tree T' among all trees. Assuming unique costs for all edges, prove that T and T' are neighbors.

(b) Given a graph $G = (V, E)$, construct an algorithm to find the second cheapest tree, T'.

(c) Consider a graph, H, whose vertices are the spanning trees of the graph G. Two vertices are connected by an edge if and only if they are neighbors as previously defined. Prove that for any graph G this new graph H is connected.

5. Network Throughput:
Suppose you are given a graph of a (tremendously simplified) computer network $G = (V, E)$ such that a weight, b_e, is assigned to each edge representing the communication bandwidth of the specified channel in Kb/s and each node is assigned a value, l_v, representing the server latency measured in seconds/packet. Given a fixed packet size, and assuming all edge bandwidth values are a multiple of the packet size, your job is to build a system to decide which paths to route traffic between specfied servers.
More formally, a person wants to route traffic from server s to server t along the path of maximum throughput. Give an algorithm that will allow a network design engineer to choose an optimal path by which to route data traffic.

6. All-Pairs-Shortest-Path:
[This problem is required only for graduate students taking CS 373 for a full unit; anyone else can submit a solution for extra credit.]
Given an undirected, unweighted, connected graph $G = (V, E)$, we wish to solve the distance version of the all-pairs-shortest-path problem. The algorithm APD takes the $n \times n$ 0-1 adjacency matrix A and returns an $n \times n$ matrix D such that d_{ij} represents the shortest path between vertices i and j.

```
APD(A)
    Z ← A · A
    let B be an n × n matrix, where b_ij = 1 iff i ≠ j and (a_ij = 1 or z_ij > 0)
    if b_ij = 1 for all i ≠ j
        return D ← 2B − A
    T ← APD(B)
    X ← T · A
    foreach x_ij
        if x_ij ≥ t_ij · degree(j)
            d_ij ← 2t_ij
        else
            d_ij ← 2t_ij − 1
    return D
```

(a) In the APD algorithm above, what do the matrices Z, B, T, and X represent? Justify your answers.

(b) Prove that the APD algorithm correctly computes the matrix of shortest path distances. In other words, prove that in the output matrix D, each entry d_{ij} represents the shortest path distance between node i and node j.

(c) Suppose we can multiply two $n \times n$ matrices in $M(n)$ time, where $M(n) = \Omega(n^2)$.[1] Prove that APD runs in $O(M(n) \log n)$ time.

[1]The matrix multiplication algorithm you already know runs in $O(n^3)$ time, but this is not the fastest known. The current record is $M(n) = O(n^{2.376})$, due to Don Coppersmith and Shmuel Winograd. Determinign the smallest possible value of $M(n)$ is a long-standing open problem.

Practice Problems

1. Makefiles:
   In order to facilitate recompiling programs from multiple source files when only a small number of files have been updated, there is a UNIX utility called 'make' that only recompiles those files that were changed after the most recent compilation, *and* any intermediate files in the compilation that depend on those that were changed. A Makefile is typically composed of a list of source files that must be compiled. Each of these source files is dependent on some of the other files which are listed. Thus a source file must be recompiled if a file on which it depends is changed.

   Assuming you have a list of which files have been recently changed, as well as a list for each source file of the files on which it depends, design an algorithm to recompile only those necessary. DO NOT worry about the details of parsing a Makefile.

2. The incidence matrix of an undirected graph $G = (V, E)$ is a $|V| \times |E|$ matrix $B = (b_{ij})$ such that
   $$b_{ij} = \begin{cases} 1 & \text{if vertex } v_i \text{ is an endpoint of edge } e_j \\ 0 & \text{otherwise} \end{cases}$$

   (a) Describe what all the entries of the matrix product BB^T represent (B^T is the matrix transpose). Justify.

   (b) Describe what all the entries of the matrix product $B^T B$ represent. Justify.

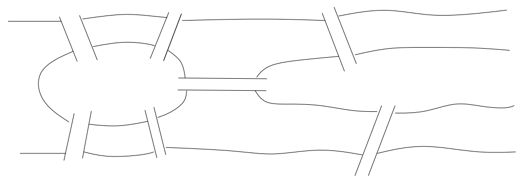
   ★(c) Let $C = BB^T - 2A$. Let C' be C with the first row and column removed. Show that $\det C'$ is the number of spanning trees. (A is the adjacency matrix of G, with zeroes on the diagonal).

3. Reliable Network:
   Suppose you are given a graph of a computer network $G = (V, E)$ and a function $r(u, v)$ that gives a reliability value for every edge $(u, v) \in E$ such that $0 \leq r(u, v) \leq 1$. The reliability value gives the probability that the network connection corresponding to that edge will *not* fail. Describe and analyze an algorithm to find the most reliable path from a given source vertex s to a given target vertex t.

4. Aerophobia:
   After graduating you find a job with Aerophobes-R′-Us, the leading traveling agency for aerophobic people. Your job is to build a system to help customers plan airplane trips from one city to another. All of your customers are afraid of flying so the trip should be as short as possible.

   In other words, a person wants to fly from city A to city B in the shortest possible time. S/he turns to the traveling agent who knows all the departure and arrival times of all the flights on the planet. Give an algorithm that will allow the agent to choose an optimal route to minimize the total time in transit. Hint: rather than modify Dijkstra's algorithm, modify the data. The total transit time is from departure to arrival at the destination, so it will include layover time (time waiting for a connecting flight).

5. The Seven Bridges of Königsberg:
During the eighteenth century the city of Königsberg in East Prussia was divided into four sections by the Pregel river. Seven bridges connected these regions, as shown below. It was said that residents spent their Sunday walks trying to find a way to walk about the city so as to cross each bridge exactly once and then return to their starting point.



   (a) Show how the residents of the city could accomplish such a walk or prove no such walk exists.

   (b) Given any undirected graph $G = (V, E)$, give an algorithm that finds a cycle in the graph that visits every edge exactly once, or says that it can't be done.

6. Given an undirected graph $G = (V, E)$ with costs $c_e \geq 0$ on the edges $e \in E$ give an $O(|E|)$ time algorithm that tests if there is a minimum cost spanning tree that contains the edge e.

7. Combining Boruvka and Prim:
Give an algorithm that find the MST of a graph G in $O(m \log\log n)$ time by combining Boruvka's and Prim's algorithm.

8. Minimum Spanning Tree changes:
Suppose you have a graph G and an MST of that graph (i.e. the MST has already been constructed).

   (a) Give an algorithm to update the MST when an edge is added to G.

   (b) Give an algorithm to update the MST when an edge is deleted from G.

   (c) Give an algorithm to update the MST when a vertex (and possibly edges to it) is added to G.

9. Nesting Envelopes
You are given an unlimited number of each of n different types of envelopes. The dimensions of envelope type i are $x_i \times y_i$. In nesting envelopes inside one another, you can place envelope A inside envelope B if and only if the dimensions A are *strictly smaller* than the dimensions of B. Design and analyze an algorithm to determine the largest number of envelopes that can be nested inside one another.

10. $o(V^2)$ Adjacency Matrix Algorithms

   (a) Give an $O(V)$ algorithm to decide whether a directed graph contains a *sink* in an adjacency matrix representation. A sink is a vertex with in-degree $V - 1$.

(b) An undirected graph is a scorpion if it has a vertex of degree 1 (the sting) connected to a vertex of degree two (the tail) connected to a vertex of degree $V-2$ (the body) connected to the other $V-3$ vertices (the feet). Some of the feet may be connected to other feet.

Design an algorithm that decides whether a given adjacency matrix represents a scorpion by examining only $O(V)$ of the entries.

(c) Show that it is impossible to decide whether G has at least one edge in $O(V)$ time.

11. Shortest Cycle:
Given an **undirected** graph $G=(V,E)$, and a weight function $f: E \to \mathbf{R}$ on the **edges**, give an algorithm that finds (in time polynomial in V and E) a cycle of smallest weight in G.

12. Longest Simple Path:
Let graph $G=(V,E)$, $|V|=n$. A *simple path* of G, is a path that does not contain the same vertex twice. Use dynamic programming to design an algorithm (not polynomial time) to find a simple path of maximum length in G. Hint: It can be done in $O(n^c 2^n)$ time, for some constant c.

13. Minimum Spanning Tree:
Suppose all edge weights in a graph G are equal. Give an algorithm to compute an MST.

14. Transitive reduction:
Give an algorithm to construct a *transitive reduction* of a directed graph G, i.e. a graph G^{TR} with the fewest edges (but with the same vertices) such that there is a path from a to b in G iff there is also such a path in G^{TR}.

15. (a) What is $5^{2^{29^{5^{0}}+23^{4^{1}}+17^{3^{2}}+11^{2^{3}}+5^{1^{4}}}} \bmod 6$?

(b) What is the capital of Nebraska? Hint: It is not Omaha. It is named after a famous president of the United States that was not George Washington. The distance from the Earth to the Moon averages roughly 384,000 km.

CS 373: Combinatorial Algorithms, Fall 2002

http://www-courses.cs.uiuc.edu/~cs373

Homework 5 (due Thur. Nov. 21, 2002 at 11:59 pm)

Name:		
Net ID:	Alias:	U $^3\!/_4$ 1

Name:		
Net ID:	Alias:	U $^3\!/_4$ 1

Name:		
Net ID:	Alias:	U $^3\!/_4$ 1

Neatly print your name(s), NetID(s), and the alias(es) you used for Homework 0 in the boxes above. Please also tell us whether you are an undergraduate, 3/4-unit grad student, or 1-unit grad student by circling U, $^3\!/_4$, or 1, respectively. Staple this sheet to the top of your homework.

Required Problems

1. (10 points) Given two arrays, $A[1..n]$ and $B[1..m]$ we want to determine whether there is an $i \geq 0$ such that $B[1] = A[i+1]$,$B[2] = A[i+2]$,...,$B[m] = A[i+m]$. In other words, we want to determine if B is a substring of A. Show how to solve this problem in $O(n \log n)$ time with high probability.

2. (5 points) Let $a, b, c \in \mathbb{Z}^+$.

   (a) Prove that $\gcd(a, b) \cdot \operatorname{lcm}(a, b) = ab$.

   (b) Prove $\operatorname{lcm}(a, b, c) = \operatorname{lcm}(\operatorname{lcm}(a, b), c)$.

   (c) Prove $\gcd(a, b, c)\operatorname{lcm}(ab, ac, bc) = abc$.

3. (5 points) Describe an efficient algorithm to compute multiplicative inverses modulo a prime p. Does your algorithm work if the modulos is composite?

4. (10 points) Describe an efficient algorithm to compute $F_n \bmod m$, given integers n and m as input.

5. (10 points) Let n have the prime factorization $p_1^{k_1} p_2^{k_2} \cdots p_t^{k_t}$, where the primes p_i are distinct and have exponents $k_i > 0$. Prove that

$$\phi(n) = \prod_{i=1}^{t} p_i^{k_i - 1}(p_i - 1).$$

Conclude that $\phi(n)$ can be computed in polynomial time given the prime factorization of n.

6. (10 points) Suppose we want to compute the Fast Fourier Transform of an intger vector $P[0\,..\,n-1]$. We could choose an integer m larger than any coefficient $P[i]$, and then perform all arithmetic modulo m (or more formally, in the ring $\mathbb{Z}_m$). In order to make the FFT algorithm work, we need to find an integer that functions as a "primitive nth root of unity modulo m".

   For this problem, let's assume that $m = 2^{n/2} + 1$, where as usual n is a power of two.

   (a) Prove that $2^n \equiv 1 \pmod{m}$.

   (b) Prove that $\sum_{k=0}^{n-1} 2^k \equiv 0 \pmod{m}$. These two conditions imply that 2 is a primitive nth root of unity in $\mathbb{Z}_m$.

   (c) Given (a), (b), and (c), *briefly* argue that the "FFT modulo m" of P is well-defined and be computed in $O(n \log n)$ arithmetic operations.

   (d) Prove that n has a multiplicative inverse in $\mathbb{Z}_m$. *[Hint: n is a power of 2, and m is odd.]* We need this property to implement the inverse FFT modulo m.

   (e) What is the FFT of the sequence $[3, 1, 3, 3, 7, 3, 7, 3]$ modulo 17?

7. (10 points) *[This problem is required only for graduate students taking CS 373 for a full unit; anyone else can submit a solution for extra credit.]*

   (a) Prove that for any integer $n > 1$, if the n-th Fibonacci number F_n is prime then either n is prime or $n = 4$.

   (b) Prove that if a divides b, then F_a divides F_b.

   (c) Prove that $\gcd(F_a, F_b) = F_{\gcd(a,b)}$. This immediately implies parts (a) and (b), so if you solve this part, you don't have to solve the other two.

Practice Problems

1. Let $a, b, n \in Z \setminus \{0\}$. Assume $\gcd(a,b)|n$. Prove the entire set of solutions to the equation

$$n = ax + by$$

is given by:

$$\Gamma = \{x_o + \frac{tb}{\gcd(a,b)}, y_0 - \frac{ta}{\gcd(a,b)} : t \in Z\}.$$

2. Show that in the RSA cryptosystem the decryption exponent d can be chosen such that $de \equiv 1 \mod \operatorname{lcm}(p-1, q-1)$.

3. Let (n, e) be a public RSA key. For a plaintext $m \in \{0, 1, ..., n-1\}$, let $c = m^e \mod n$ be the corresponding ciphertext. Prove that there is a positive integer k such that

$$m^{e^k} \equiv m \mod n.$$

For such an integer k, prove that

$$c^{e^{k-1}} \equiv m \mod n.$$

Is this dangerous for RSA?

4. Prove that if Alice's RSA public exponent e is 3 and an adversary obtains Alice's secret exponent d, then the adversary can factor Alice's modulus n in time polynomial in the number of bits in n.

CS 373: Combinatorial Algorithms, Fall 2002

http://www-courses.cs.uiuc.edu/~cs373

Homework 6 (Do not hand in!)

Name:		
Net ID:	Alias:	U $^3/_4$ 1

Name:		
Net ID:	Alias:	U $^3/_4$ 1

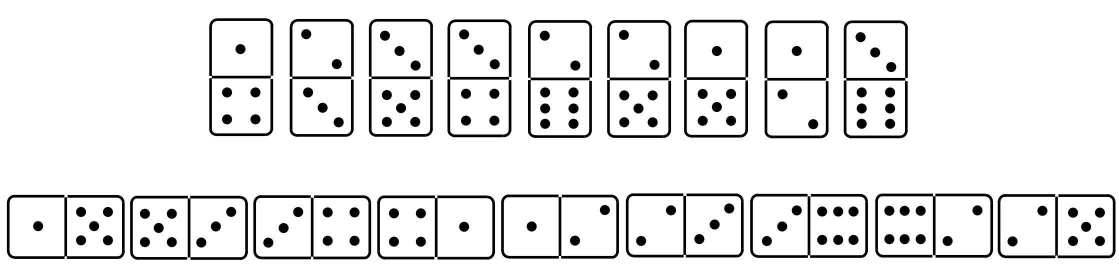
Name:		
Net ID:	Alias:	U $^3/_4$ 1

Neatly print your name(s), NetID(s), and the alias(es) you used for Homework 0 in the boxes above. Please also tell us whether you are an undergraduate, 3/4-unit grad student, or 1-unit grad student by circling U, $^3/_4$, or 1, respectively. Staple this sheet to the top of your homework.

Required Problems

1. (10 points) Prove that SAT is still a NP-complete problem even under the following constraints: each variable must show up once as a positive literal and once or twice as a negative literal in the whole expression. For instance, $(A\vee\overline{B})\wedge(\overline{A}\vee C\vee D)\wedge(\overline{A}\vee B\vee\overline{C}\vee\overline{D})$ satisfies the constraints, while $(A\vee\overline{B})\wedge(\overline{A}\vee C\vee D)\wedge(A\vee B\vee\overline{C}\vee\overline{D})$ does not, because positive literal A appears twice.

2. (10 points) A domino is 2×1 rectanble divided into two squares, with a certain number of pips(dots) in each square. In most domino games, the players lay down dominos at either end of a single chain. Adjacent dominos in the chain must have matching numbers. (See the figure below.)
   Describe and analyze an efficient algorithm, or prove that it is NP-complete, to determine wheter a given set of n dominos can be lined up in a single chain. For example, for the sets of dominos shown below, the correct output is TRUE.



Top: A set of nine dominos
Bottom:The entire set lined up in a single chain

3. (10 points) Prove that the following 2 problems are NP-complete. Given an undirected Graph $G = (V, E)$, a subset of vertices $V' \subseteq V$, and a positive integer k:

   (a) determine whether there is a spanning tree T of G whose leaves are the same as V'.

   (b) determine whether there is a spanning tree T of G whose degree of vertices are all less than k.

4. (10 points) An optimized version of Knapsack problem is defined as follows. Given a finite set of elements U where each element of the set $u \in U$ has its own size $s(u) > 0$ and the value $v(u) > 0$, maximize $A(U') = \sum_{u \in U'} v(u)$ under the condition $\sum_{u \in U'} s(u) \leq B$ and $U' \subseteq U$. This problem is NP-hard. Consider the following polynomial time approximation algorithm. Determine the worst case approximation ratio $R(U) = \max_{U} Opt(U)/Approx(U)$ and prove it.

```
APPROXIMATIONALGORITHM:
    A1 ← Greedy()
    A2 ← SingleElement()
    return max(A1, A2)
```

```
GREEDY:
    Put all the elements u∈U into an array A[i]
    Sort A[i] by v(u)/s(u) in a decreasing order
    S ← 0
    V ← 0
    for i ← 0 to NumOfElements
        if (S + s(u[i]) > B)
            break
        S ← S + s(u[i])
        V ← V + v(u[i])
    return V
```

```
SINGLEELEMENT:
    Put all the elements u∈U into an array A[i]
    V ← 0
    for i ← 0 to NumOfElements
        if (s(u[i]) ≤ B & V < v(u[i]))
            V ← v(u[i])
    return V
```

5. (10 points) The recursion fairy's distant cousin, the reduction genie, shows up one day with a magical gift for you: a box that determines in constant time whether or not a graph is 3-colorable.(A graph is 3-colorable if you can color each of the vertices red, green, or blue, so that every edge has do different colors.) The magic box does not tell you how to color the graph, just wheter or not it can be done. Devise and analyze an algorithm to 3-color any graph in **polynomial time** using the magic box.

6. (10 points) The following is an NP-hard version of PARTITION problem.

> PARTITION(NP-HARD):
>
> Given a set of n positive integers $S = \{a_i | i = 0 \ldots n-1\}$,
>
> minimize $\max\left(\sum_{a_i \in T} a_i, \sum_{a_i \in S-T} a_i\right)$
>
> where T is a subset of S.

A polynomial time approximation algorithm is given in what follows. Determine the worst case approximation ratio $\min_{S} Approx(S)/Opt(S)$ and prove it.

```
APPROXIMATIONALGORITHM:
    Sort S in an increasing order
    s1 ← 0
    s2 ← 0
    for i ← 0 to n
        if s1 ≤ s2
            s1 ← s1 + a_i
        else
            s2 ← s2 + a_i
    result ← max(s1, s2)
```

Practice Problems

1. Construct a linear time algorithm for 2 SAT problem.

2. Assume that $P \neq NP$. Prove that there is no polynomial time approximation algorithm for an optimized version of Knapsack problem, which outputs $A(I)$ s.t. $|Opt(I) - A(I)| \leq K$ for any instance I, where K is a constant.

3. Your friend Toidi is planning to hold a party for the coming Christmas. He wants to take a picture of all the participants including himself, but he is quite **shy** and thus cannot take a picture of a person whom he does not know very well. Since he has only **shy** friends, every participant coming to the party is also **shy**. After a long struggle of thought he came up with a seemingly good idea:

   - At the beginning, he has a camera.
   - A person, holding a camera, is able to take a picture of another participant whom the person knows very well, and pass a camera to that participant.
   - Since he does not want to waste films, everyone has to be taken a picture exactly once.

   Although there can be some people whom he does not know very well, he knows completely who knows whom well. Therefore, in theory, given a list of all the participants, he can determine if it is possible to take all the pictures using this idea. Since it takes only linear time to take all the pictures if he is brave enough (say “Say cheese!” N times, where N is the number of people), as a student taking CS373, you are highly expected to give him an advice:

   - show him an efficient algorithm to determine if it is possible to take pictures of all the participants using his idea, given a list of people coming to the party.
   - or prove that his idea is essentially facing a NP-complete problem, make him give up his idea, and give him an efficient algorithm to practice saying “Say cheese!”:

     e.g., $\boxed{\begin{array}{l}\text{for } i \leftarrow 0 \text{ to } N \\ \quad \text{Make him say “Say cheese!” } 2^i \text{ times}\end{array}}$ oops, it takes exponential time...

4. Show, given a set of numbers, that you can decide wheter it has a subset of size 3 that adds to zero in polynomial time.

5. Given a CNF-normalized form that has at most one negative literal in each clause, construct an efficient algorithm to solve the satisfiability problem for these clauses. For instance,

$$(A \vee B \vee \overline{C}) \wedge (B \vee \overline{A}),$$
$$(A \vee \overline{B} \vee C) \wedge (B \vee \overline{A} \vee D) \wedge (A \vee D),$$
$$(\overline{A} \vee B) \wedge (B \vee \overline{A} \vee C) \wedge (C \vee D)$$

satisfy the condition, while

$$(\overline{A} \vee B \vee \overline{C}) \wedge (B \vee \overline{A}),$$
$$(A \vee \overline{B} \vee C) \wedge (B \vee \overline{A} \vee \overline{D}) \wedge (A \vee D),$$
$$(\overline{A} \vee B) \wedge (B \vee \overline{A} \vee C) \wedge (\overline{C} \vee \overline{D})$$

do not.

6. The ExactCoverByThrees problem is defined as follows: given a finite set X and a collection C of 3-element subsets of X, does C contain an exact cover for X, that is, a sub-collection $C' \subseteq C$ where every element of X occurs in exactly one member of C'? Given that ExactCoverByThrees is NP-complete, show that the similar problem ExactCoverByFours is also NP-complete.

7. The *LongestSimpleCycle* problem is the problem of finding a simple cycle of maximum length in a graph. Convert this to a formal definition of a decision problem and show that it is NP-complete.

Write your answers in the separate answer booklet.

1. **Multiple Choice:** Each question below has one of the following answers.

   A: $\Theta(1)$ B: $\Theta(\log n)$ C: $\Theta(n)$ D: $\Theta(n \log n)$ E: $\Theta(n^2)$ X: I don't know.

   For each question, write the letter that corresponds to your answer. You do not need to justify your answers. Each correct answer earns you 1 point. Each X earns you $\frac{1}{4}$ point. **Each incorrect answer *costs* you $\frac{1}{2}$ point.** Your total score will be rounded **down** to an integer. Negative scores will be rounded up to zero.

   (a) What is $\sum_{i=1}^{n} \frac{i}{n}$?

   (b) What is $\sum_{i=1}^{n} \frac{n}{i}$?

   (c) How many bits do you need to write 10^n in binary?

   (d) What is the solution of the recurrence $T(n) = 9T(n/3) + n$?

   (e) What is the solution of the recurrence $T(n) = T(n-2) + \frac{3}{n}$?

   (f) What is the solution of the recurrence $T(n) = 5T\left(\left\lceil \frac{n-17}{25} \right\rceil - \lg \lg n\right) + \pi n + 2^{\sqrt{\log^* n}} - 6$?

   (g) What is the worst-case running time of randomized quicksort?

   (h) The expected time for inserting one item into a randomized treap is $O(\log n)$. What is the worst-case time for a sequence of n insertions into an initially empty treap?

   (i) Suppose StupidAlgorithm produces the correct answer to some problem with probability $1/n$. How many times do we have to run StupidAlgorithm to get the correct answer with high probability?

   (j) Suppose you correctly identify three of the possible answers to this question as obviously wrong. If you choose one of the three remaining answers at random, each with equal probability, what is your expected score for this question?

2. Consider the following algorithm for finding the smallest element in an unsorted array:

```
RandomMin(A[1..n]):
  min ← ∞
  for i ← 1 to n in random order
      if A[i] < min
          min ← A[i]    (⋆)
  return min
```

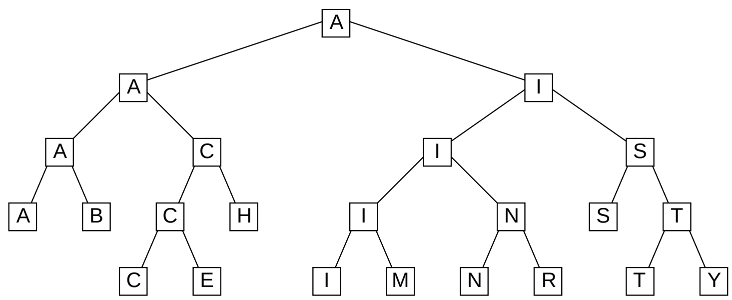
   (a) **[1 point]** In the worst case, how many times does RandomMin execute line ($\star$)?

   (b) **[3 points]** What is the probability that line ($\star$) is executed during the nth iteration of the for loop?

   (c) **[6 points]** What is the exact expected number of executions of line ($\star$)? (A correct $\Theta()$ bound is worth 4 points.)

3. Algorithms and data structures were developed millions of years ago by the Martians, but not quite in the same way as the recent development here on Earth. Intelligent life evolved independently on Mars' two moons, Phobos and Deimos.[1] When the two races finally met on the surface of Mars, after thousands of Phobos-orbits[2] of separate philosophical, cultural, religious, and scientific development, their disagreements over the proper structure of binary search trees led to a bloody (or more accurately, ichorous) war, ultimately leading to the destruction of all Martian life.

   A *Phobian* binary search tree is a full binary tree that stores a set X of search keys. The root of the tree stores the *smallest* element in X. If X has more than one element, then the left subtree stores all the elements less than some pivot value p, and the right subtree stores everything else. Both subtrees are *nonempty* Phobian binary search trees. The actual pivot value p is *never* stored in the tree.



A Phobian binary search tree for the set $\{M, A, R, T, I, N, B, Y, S, C, H, E\}$.

   (a) **[2 points]** Describe and analyze an algorithm $\text{Find}(x, T)$ that returns True if x is stored in the Phobian binary search tree T, and False otherwise.

   (b) **[2 points]** Show how to perform a rotation in a Phobian binary search tree in $O(1)$ time.

   (c) **[6 points]** A *Deimoid* binary search tree is almost exactly the same as its Phobian counterpart, except that the *largest* element is stored at the root, and both subtrees are Deimoid binary search trees. Describe and analyze an algorithm to transform an n-node Phobian binary search tree into a Deimoid binary search tree in $O(n)$ time, using as little additional space as possible.

4. Suppose we are given an array $A[1..n]$ with the special property that $A[1] \geq A[2]$ and $A[n-1] \leq A[n]$. We say that an element $A[x]$ is a *local minimum* if it is less than or equal to both its neighbors, or more formally, if $A[x-1] \geq A[x]$ and $A[x] \leq A[x+1]$. For example, there are five local minima in the following array:

9	7	7	2	**1**	3	7	5	**4**	7	**3**	**3**	4	8	**6**	9

   We can obviously find a local minimum in $O(n)$ time by scanning through the array. Describe and analyze an algorithm that finds a local minimum in $O(\log n)$ time. *[Hint: With the given boundary conditions, the array **must** have at least one local minimum. Why?]*

[1] Greek for "fear" and "panic", respectively. Doesn't that make you feel better?
[2] 1000 Phobos orbits $\approx$ 1 Earth year

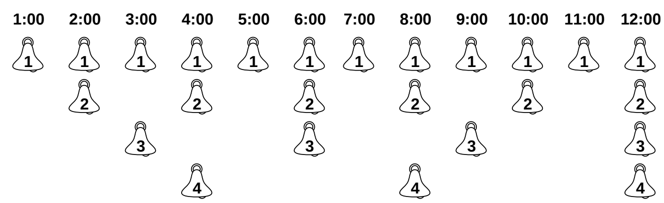
5. *[Graduate students must answer this question.]*

   A *common supersequence* of two strings A and B is a string of minimum total length that includes both the characters of A in order and the characters of B in order. Design and analyze and algorithm to compute the length of the *shortest* common supersequence of two strings $A[1..m]$ and $B[1..n]$. For example, if the input strings are `ANTHROHOPOBIOLOGICAL` and `PRETERDIPLOMATICALLY`, your algorithm should output 31, since a shortest common supersequence of those two strings is `PREANTHEROHODPOBIOPLOMATGICALLY`. You do not need to compute an actual supersequence, just its length. For full credit, your algorithm must run in $\Theta(nm)$ time.

Write your answers in the separate answer booklet.
This is a 90-minute exam. The clock started when you got the questions.

1. Professor Quasimodo has built a device that automatically rings the bells in the tower of the Cathédrale de Notre Dame de Paris so he can finally visit his true love Esmerelda. Every hour exactly on the hour (when the minute hand is pointing at the 12), the device rings at least one of the n bells in the tower. Specifically, the ith bell is rung once every i hours.

   For example, suppose $n = 4$. If Quasimodo starts his device just after midnight, then his device rings the bells according to the following twelve-hour schedule:



   What is the *amortized* number of bells rung per hour, as a function of n? For full credit, give an exact closed-form solution; a correct $\Theta()$ bound is worth 5 points.

2. Let G be a directed graph, where every edge $u \to v$ has a weight $w(u \to v)$. To compute the shortest paths from a start vertex s to every other node in the graph, the generic single-source shortest path algorithm calls InitSSSP once and then repeatedly calls Relax until there are no more tense edges.

```
InitSSSP(s):
  dist(s) ← 0
  pred(s) ← Null
  for all vertices v ≠ s
      dist(v) ← ∞
      pred(v) ← Null
```

```
Relax(u → v):
  if dist(v) > dist(u) + w(u → v)
      dist(v) ← dist(u) + w(u → v)
      pred(v) ← u
```

   Suppose the input graph has no negative cycles. Let v be an arbitrary vertex in the input graph. **Prove** that after every call to Relax, if $\text{dist}(v) \neq \infty$, then $\text{dist}(v)$ is the total weight of some path from s to v.

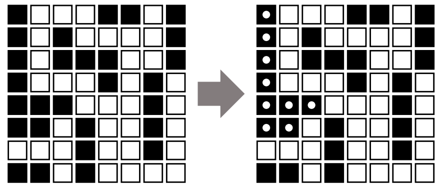
3. Suppose we want to maintain a dynamic set of values, subject to the following operations:

   - Insert(x): Add x to the set (if it isn't already there).
   - Print&DeleteRange(a, b): Print and delete every element x in the range $a \leq x \leq b$. For example, if the current set is $\{1, 5, 3, 4, 8\}$, then Print&DeleteRange$(4, 6)$ prints the numbers 4 and 5 and changes the set to $\{1, 3, 8\}$.

   Describe and analyze a data structure that supports these operations, each with amortized cost $O(\log n)$.

4. (a) **[4 pts]** Describe and analyze an algorithm to compute the size of the largest connected component of black pixels in an $n \times n$ bitmap $B[1\,..\,n, 1\,..\,n]$.

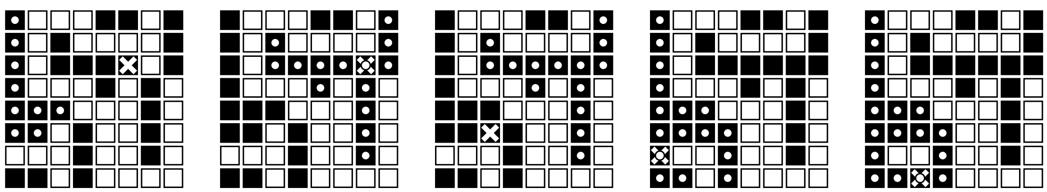
   For example, given the bitmap below as input, your algorithm should return the number 9, because the largest conected black component (marked with white dots on the right) contains nine pixels.

   9

   (b) **[4 pts]** Design and analyze an algorithm Blacken(i, j) that colors the pixel $B[i, j]$ black and returns the size of the largest black component in the bitmap. For full credit, the *amortized* running time of your algorithm (starting with an all-white bitmap) must be as small as possible.

   For example, at each step in the sequence below, we blacken the pixel marked with an X. The largest black component is marked with white dots; the number underneath shows the correct output of the Blacken algorithm.

   9 14 14 16 17

   (c) **[2 pts]** What is the *worst-case* running time of your Blacken algorithm?

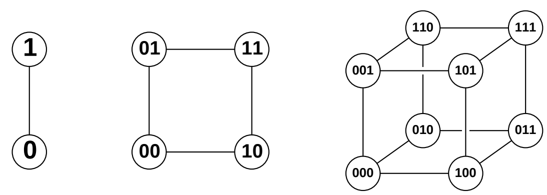
5. *[Graduate students must answer this question.]*

   After a grueling 373 midterm, you decide to take the bus home. Since you planned ahead, you have a schedule that lists the times and locations of every stop of every bus in Champaign-Urbana. Unfortunately, there isn't a single bus that visits both your exam building and your home; you must transfer between bus lines at least once.

   Describe and analyze an algorithm to determine the sequence of bus rides that will get you home as early as possible, assuming there are b different bus lines, and each bus stops n times per day. Your goal is to minimize your *arrival time*, not the time you actually spend travelling. Assume that the buses run exactly on schedule, that you have an accurate watch, and that you are too tired to walk between bus stops.

Write your answers in the separate answer booklet.
This is a 180-minute exam. The clock started when you got the questions.

1. The *d-dimensional hypercube* is the graph defined as follows. There are 2^d vertices, each labeled with a different string of d bits. Two vertices are joined by an edge if their labels differ in exactly one bit.

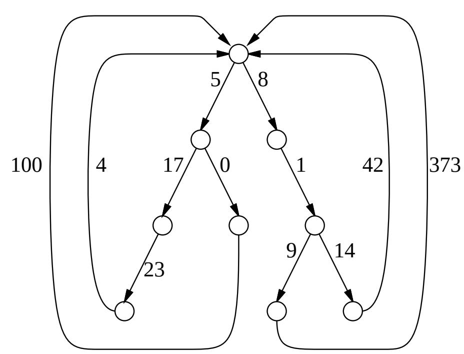


The 1-dimensional, 2-dimensional, and 3-dimensional hypercubes.

   (a) **[8 pts]** Recall that a Hamiltonian cycle passes through every vertex in a graph exactly once. ***Prove*** that for all $d \geq 2$, the d-dimensional hypercube has a Hamiltonian cycle.

   (b) **[2 pts]** Which hypercubes have an Eulerian circuit (a closed walk that visits every edge exactly once)? *[Hint: This is very easy.]*

2. A *looped tree* is a weighted, directed graph built from a binary tree by adding an edge from every leaf back to the root. Every edge has a non-negative weight. The number of nodes in the graph is n.



   (a) How long would it take Dijkstra's algorithm to compute the shortest path between two vertices u and v in a looped tree?

   (b) Describe and analyze a faster algorithm.

3. Prove that $(x + y)^p \equiv x^p + y^p \pmod{p}$ for any prime number p.

4. A *palindrome* is a string that reads the same forwards and backwards, like X, 373, noon, redivider, or amanaplanacatahamayakayamahatacanalpanama. *Any* string can be written as a sequence of palindromes. For example, the string bubbaseesabanana ('Bubba sees a banana.') can be decomposed in several ways; for example:

   bub + baseesab + anana
   b + u + bb + a + sees + aba + nan + a
   b + u + bb + a + sees + a + b + anana
   b + u + b + b + a + s + e + e + s + a + b + a + n + a + n + a

   Describe an efficient algorithm to find the *minimum* number of palindromes that make up a given input string. For example, given the input string bubbaseesabanana, your algorithm would return the number 3.

5. Your boss wants you to find a *perfect* hash function for mapping a known set of n items into a table of size m. A hash function is perfect if there are *no* collisions; each of the n items is mapped to a different slot in the hash table. Of course, this requires that $m \geq n$.

   After cursing your 373 instructor for not teaching you about perfect hashing, you decide to try something simple: repeatedly pick *random* hash functions until you find one that happens to be perfect. A random hash function h satisfies two properties:

   - $\Pr\big[h(x) = h(y)\big] = \frac{1}{m}$ for any pair of items $x \neq y$.
   - $\Pr\big[h(x) = i\big] = \frac{1}{m}$ for any item x and any integer $1 \leq i \leq m$.

   (a) **[2 pts]** Suppose you pick a random hash function h. What is the *exact* expected number of collisions, as a function of n (the number of items) and m (the size of the table)? Don't worry about how to *resolve* collisions; just count them.

   (b) **[2 pts]** What is the *exact* probability that a random hash function is perfect?

   (c) **[2 pts]** What is the *exact* expected number of different random hash functions you have to test before you find a perfect hash function?

   (d) **[2 pts]** What is the *exact* probability that none of the first N random hash functions you try is perfect?

   (e) **[2 pts]** How many random hash functions do you have to test to find a perfect hash function *with high probability*?

   To get full credit for parts (a)–(d), give *exact* closed-form solutions; correct $\Theta(\cdot)$ bounds are worth significant partial credit. Part (e) requires only a $\Theta(\cdot)$ bound; an exact answer is worth extra credit.

6. Your friend Toidi is planning to hold a Christmas party. He wants to take a picture of all the participants, including himself, but he is quite shy and thus cannot take a picture of a person whom he does not know very well. Since he has only shy friends[1], everyone at the party is also shy. After thinking hard for a long time, he came up with a seemingly good idea:

   - Toidi brings a disposable camera to the party.
   - Anyone holding the camera can take a picture of someone they know very well, and then pass the camera to that person.
   - In order not to waste any film, every person must have their picture taken exactly once.

   Although there can be some people Toidi does not know very well, he knows completely who knows whom well. Thus, *in principle*, given a list of all the participants, he can determine whether it is possible to take all the pictures using this idea. But how quickly?

   Either describe an efficient algorithm to solve Toidi's problem, or show that the problem is NP-complete.

7. The recursion fairy's cousin, the reduction genie, shows up one day with a magical gift for you: a box that can solve the NP-complete PARTITION problem in constant time! Given a set of positive integers as input, the magic box can tell you in constant time it can be split into two subsets whose total weights are equal.

   For example, given the set $\{1, 4, 5, 7, 9\}$ as input, the magic box cheerily yells "YES!", because that set can be split into $\{1, 5, 7\}$ and $\{4, 9\}$, which both add up to 13. Given the set $\{1, 4, 5, 7, 8\}$, however, the magic box mutters a sad "Sorry, no."

   The magic box does not tell you *how* to partition the set, only whether or not it can be done. Describe an algorithm to actually split a set of numbers into two subsets whose sums are equal, **in polynomial time**, using this magic box.[2]

[1] Except you, of course. Unfortunately, you can't go to the party because you're taking a final exam. Sorry!

[2] Your solution to problem 4 in homework 1 does *not* solve this problem in polynomial time.