Facultad: Ingeniería
Escuela: Computación
Asignatura: Programación IV

Tema: Métodos de Ordenamiento. Parte 1.

Objetivos Específicos

- Identificar la estructura de algunos algoritmos de ordenamiento.
- Interpretar el algoritmo de ordenamiento en sintaxis de C#.
- Aplicar los algoritmos de ordenamiento.

Materiales y Equipo

- Guía Número 3.
- Computadora con programa Microsoft Visual C#.

Introducción Teórica

1. ¿Qué es ordenamiento?

Es la operación de arreglar los elementos en algún orden secuencial de acuerdo a un criterio de ordenamiento.

El propósito principal de un ordenamiento es el de facilitar las búsquedas de los miembros del conjunto ordenado.

Ordenar un grupo de datos significa mover los datos o sus referencias para que queden en una secuencia por categorías y en forma ascendente o descendente.

2. ¿Cuándo conviene usar un método de ordenamiento?

Cuando se requiere hacer una cantidad considerable de búsquedas y es importante el factor tiempo.

3. Tipos de ordenamientos:

Los 2 tipos de ordenamientos que se pueden realizar son: los internos y los externos.

3.1 Ordenamientos internos:

Son aquellos en los que los valores a ordenar están en memoria principal, por lo que se asume que el tiempo que se requiere para acceder cualquier elemento sea el mismo (a [1], a [500], etc.).

3.2 Ordenamientos externos:

Son aquellos en los que los valores a ordenar están en memoria secundaria (disco, cinta, cilindro magnético, etc.), por lo que se asume que el tiempo que se requiere para acceder a cualquier elemento depende de la última posición accesada (posición 1, posición 500, etc.).

4. Algoritmos de ordenamiento.

4.1 Internos:

1. Inserción (InsertionSort).
2. Selección (SelectionSort).
3. Intercambio.
4. Ordenamiento de árbol.
5. QuickSort.
6. MergeSort.
7. RadixSort.

4.2 Externos:

1. Straight merging.
2. Natural merging.
3. Balanced multiway merging.
4. Polyphase sort.
5. Distribution of initial runs.

5. Clasificación de los algoritmos de ordenamiento de información.

El hecho de que la información está ordenada, nos sirve para poder encontrarla y acceder de manera más eficiente, ya que de lo contrario se tendría que hacer de manera secuencial.
A continuación se describirán cuatro grupos de algoritmos para ordenar información.

5.1 Algoritmos de inserción.

En este tipo de algoritmo los elementos que van a ser ordenados son considerados uno a la vez. Cada elemento es “insertado” en la posición apropiada con respecto al resto de los elementos ya ordenados. Entre estos algoritmos se encuentran el de *Inserción Directa, ShellSort, Inserción Binaria y Hashing*.

5.2 Algoritmos de intercambio.

En este tipo de algoritmos se toman los elementos de dos en dos, se comparan y se “intercambian” si no están en el orden adecuado. Este proceso se repite hasta que se ha analizado todo el conjunto de elementos y ya no hay intercambios. Entre estos algoritmos se encuentran *el Burbuja (BubbleSort) y QuickSort*.

5.3 Algoritmos de selección.

En este tipo de algoritmos se “selecciona” o se busca el elemento más pequeño (o más grande) de todo el conjunto de elementos y se coloca en su posición adecuada. Este proceso se repite para el resto de los elementos hasta que todos son analizados. Entre estos algoritmos se encuentra el de *Selección Directa*.

5.4 Algoritmos de enumeración.

En este tipo de algoritmos cada elemento es comparado contra los demás. En la comparación se cuenta cuántos elementos son más pequeños que el elemento que se está analizando, generando así una “enumeración”. El número generado para cada elemento indicará su posición.

6. Los métodos de ordenamiento se dividen en simples y complejos.

6.1 Los métodos simples: Inserción (o por inserción directa), Selección, Burbuja y ShellSort, en dónde el último es una extensión al método de inserción, siendo más rápido.

6.2 Los métodos complejos: el QuickSort (ordenación rápida) y el HeapSort.

Procedimiento

El entorno que se ocupará en C#, para esta guía será en modo "**Windows Forms**", por lo cual se seguirán los siguientes pasos:

1. Inicio -> Todos los programas -> Microsoft Visual Studio 2012 -> Microsoft Visual Studio 2012.
2. Archivo -> Nuevo -> Proyecto -> En la ventana emergente de Nuevo proyecto, seleccionar en Tipo de proyecto:(Visual C# -> Windows Forms Application), Nombre: Según se le indique en el ejemplo.

Realice los siguientes ejemplos interpretando el algoritmo de ordenamiento, en sintaxis de C#.

Ejemplo 1. Método de ordenamiento Burbuja.

El Ordenamiento de Burbuja (BubbleSort en inglés) es un sencillo algoritmo de ordenamiento. Funciona revisando cada elemento de la lista que va a ser ordenada con el siguiente, intercambiándolos de posición si están en el orden equivocado.

Es necesario revisar varias veces toda la lista hasta que no se necesiten más intercambios, lo cual significa que la lista está ordenada.

Este algoritmo obtiene su nombre de la forma con la que suben por la lista los elementos durante los intercambios, como si fueran pequeñas "burbujas". También es conocido como ***método del intercambio directo***.

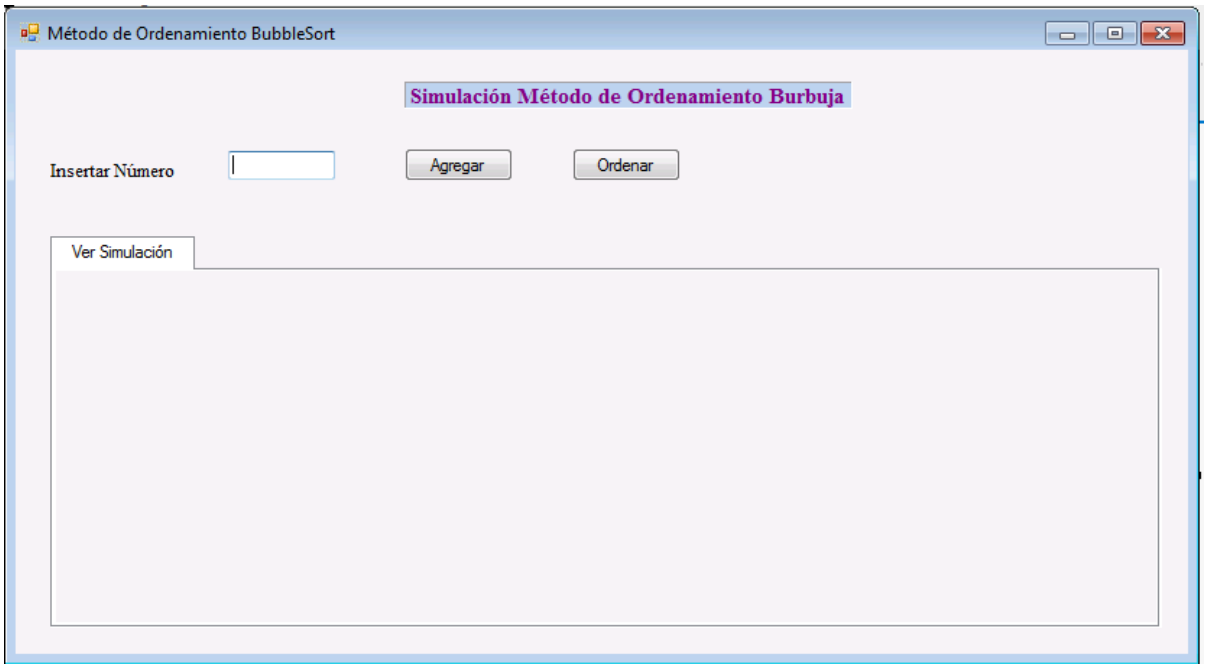
Algoritmo.

```
Para i = 0    condicion i < (Numero de elementos de a) paso i+1 Hacer
    Para j = 0  condición j <  (Numero de elementos de a) paso j+1  Hacer
        Si a [j] > a [j+1] entonces Hacer
            aux = a [j]
            a [j] = a [j+1]
            a [j+1] = aux
    Fin del ciclo Para
Fin del ciclo Para
```

Código en C#. (Guardar con el nombre Burbuja)

Lo primero que debemos hacer es renombrar el formulario con el nombre: **Burbuja**.

Luego insertamos los objetos necesarios, de tal manera que el formulario luzca similar al mostrado en la figura siguiente:



Se han agregado al formulario dos Labels, un TextBox, dos Button y un TabControl.

Al objeto TextBox le cambiamos la propiedad “Name” a “**txtNumero**”.

Al objeto Button le cambiamos la propiedad “Name” a “**btnAgregar**”.

Al otro objeto Button le cambiamos la propiedad “Name” a “**btnOrdenar**”.

Al objeto TabControl le cambiamos la propiedad “TabPages” para que solo contenga una ficha; la propiedad “SizeMode” a “**Fixed**”.

Al tabPage1 (dentro del TabControl) le cambiamos la propiedad “AutoScroll” a “**True**”.

A continuación declaramos las variables a utilizar:

```
bool estado = false;    // Nos servirá como variable de control para saber el estado de la
                        // simulación
int[] Arreglo_numeros; // Definimos un arreglo de enteros, que contendrá los datos a
                        // ordenar
Button[] Arreglo;       // Definimos un arreglo de botones, que nos ayudará para la
                        // simulación
```

Definimos una clase de nombre “Numeros” que nos servirá lógicamente para el manejo de los datos a ordenar.

```
class Numeros
{
     private int longitud;                    // cantidad de datos a ordenar
     private int[] arreglo = new int[1];
     private Button[] arreglo_botones = new Button[1]; // definimos un nuevo
                                                       // arreglo de botones

public Numeros()                                    // Constructor de la clase
{
      int a = 0;                                  // variable auxiliar
      arreglo[0] = a;                             // el texto que desplegará el botón
      arreglo_botones[0] = new Button();
      arreglo_botones[0].Width = 40;                    // definimos el ancho del botón
      arreglo_botones[0].Height = 40;                   // definimos el alto del botón
      arreglo_botones[0].BackColor = GreenYellow;       // definimos el color del botón
      arreglo_botones[0].Text = a.ToString();
      Calcular_Longitud();
}

public void Calcular_Longitud()       // Con esta función determinamos cuántos datos
{                                     // se van a ordenar
      longitud = arreglo.Length;
}

public int Obtener_Longitud()
{
      return longitud;
}

public int[] Obtener_Arreglo()
{
      return arreglo;
}

public void Insertar_Dato(int dato) // Esta función la utilizamos para insertar el
{                                   // valor digitado por el usuario como texto en el
                                    // botón
      Array.Resize<int>(ref arreglo, longitud + 1);  //Se incrementará cada vez en 1
      arreglo[longitud] = dato;
      Array.Resize<Button>(ref arreglo_botones, longitud + 1);
      arreglo[longitud] = dato;
      arreglo_botones[longitud] = new Button();
      arreglo_botones[longitud].Width = 50;
      arreglo_botones[longitud].Height = 50;
      arreglo_botones[longitud].BackColor = Color. GreenYellow;
      arreglo_botones[longitud].Text = dato.ToString();
      Calcular_Longitud();
}

public Button[] Arreglo_Botones()
{
      return arreglo_botones;
}
}  // Final Clase Numeros
```

A continuación instanciamos la clase Enteros:

```
Numeros Datos = new Numeros();   // Instanciamos la clase Numeros
```

Agregamos la codificación del método “Burbuja”:

```
// Método de Ordenamiento Burbuja
public void BubbleSort(ref int[] arreglo, ref Button[] Arreglo_Numeros)
{
    for (int i = 0; i < arreglo.Length; i++)
    {
        for (int j = 0; j < arreglo.Length - 1; j++)
        {
            if (arreglo[j] > arreglo[j + 1])
            {
               int aux = arreglo[j];
               arreglo[j] = arreglo[j + 1];
               arreglo[j + 1] = aux;
            }
        }
    }
}
```

Vamos a codificar el evento “Click” del botón “Agregar”:

```
private void btnAgregar_Click(object sender, EventArgs e)
{
    try
    {
        int num = Convert.ToInt32(txtNumero.Text);
        Datos.Insertar_Dato(num);                    // Se agrega al objeto "Datos"
        Arreglo_numeros = Datos.Obtener_Arreglo();  // Se sacan los arreglos del
                                                    // objeto "Datos"
        Arreglo = Datos.Arreglo_Botones();
    }
    catch
    {
        MessageBox.Show("Solo se admiten números enteros");
    }

    estado = true;  // Cambiamos el valor de la variable de control para la simulación
    tabPage1.Refresh();
}
```

Puedes probar la funcionalidad que tiene la aplicación. Al dar clic al botón “**Agregar**” no ocurre nada, pues no le hemos indicado al objeto “tabPage” ninguna acción. Por lo que a continuación modificaremos el evento “**Paint**” de este objeto. Este evento es el que permitirá dibujar los botones para cada dato insertado por el usuario.

```
private void tabPage1_Paint(object sender, PaintEventArgs e)
{
    if (estado)
    {
        Point xy = new Point(50, 70);    // método de la librería Drawing

        try
        {        // Función que dibujará los datos en la simulación
            Dibujar_Arreglo(ref Arreglo, xy, ref tabPage1);

        }

        catch
        {
        }

        estado = false;
    }
}
```

La función "**Dibujar_Arreglo**" la utilizamos para dibujar el arreglo de botones que representa los datos a ordenar:

```
public void Dibujar_Arreglo(ref Button[] Arreglo, Point xy, ref TabPage t)
{
    for (int i = 1; i < Arreglo.Length; i++)
    {
        Arreglo[i].Location = xy;
        t.Controls.Add(Arreglo[i]);
        xy += new Size(70, 0);
    }
}
```

Ahora programaremos el evento "Click" del botón "Ordenar":

```
private void btnOrdenar_Click(object sender, EventArgs e)
{
    this.Cursor = Cursors.WaitCursor;   // cambiamos la apariencia del cursor al modo
                                        // espera
    btnOrdenar.Enabled = false;
    txtNumero.Enabled = false;
    btnAgregar.Enabled = false;

    BubbleSort(ref Arreglo_numeros, ref  Arreglo);

    this.Cursor = Cursors.Default;      // retornamos la apariencia del cursor al modo
                                        // por defecto
    btnOrdenar.Enabled = true;
    txtNumero.Enabled = true;
    btnAgregar.Enabled = true;
}
```

Lo que nos hace falta hacer es programar la simulación del ordenamiento que queremos observar en el objeto “TabControl”. Para ello definimos un método de nombre “**Intercambio**”. Para la correcta funcionalidad de este método debemos agregar la siguiente librería de clase:

```
using System.Threading;
```

Esto nos permitirá controlar los distintos “hilos” de ejecución de la animación (con el evento “**Sleep**”).

```
public void Intercambio(ref Button[] boton, int a, int b)
{
    string temp = boton[a].Text;    // variable temporal

    Point pa = boton[a].Location;   // definimos posición inicial
    Point pb = boton[b].Location;   // definimos posición final
    int diferencia = pa.X - pb.X;   // distancia a moverse en forma horizontal
    int x = 10;
    int y = 10;
    int t = 70;

    while (y != 70)          // La cantidad de movimientos que realizará cada botón en
    {                        // forma vertical
        Thread.Sleep(t);     // Para agregar pausas al hilo de la ejecución
        boton[a].Location += new Size(0, 10);
        boton[b].Location += new Size(0, -10);
        y += 10;
    }

    while (x != diferencia + 10)                // Movimiento horizontal
    {
        Thread.Sleep(t);
        boton[a].Location += new Size(-10, 0);
        boton[b].Location += new Size(10, 0);
        x += 10;
    }

    y = 0;

    while (y != -60)                            // Movimiento vertical
    {
        Thread.Sleep(t);
        boton[a].Location += new Size(0, -10);
        boton[b].Location += new Size(0, +10);
        y -= 10;
    }

    boton[a].Text = boton[b].Text;
    boton[b].Text = temp;
    boton[b].Location = pb;
    boton[a].Location = pa;
    estado = true;
    tabPage1.Refresh();
}
```

Probamos la funcionalidad de la aplicación, ¿qué ocurre?

No se visualiza nada, pues tenemos que llamar a este método en la función donde implementamos el método de burbuja.

Agregamos al método de nombre “BubbleSort” justo después de las líneas del swap (intercambio), la siguiente línea de código:

```
// El valor contenido en índice j+1, se intercambia con el de índice j
Intercambio(ref Arreglo_Numeros, j+1, j);
```

Ahora sí, probemos nuestra aplicación y podremos observar su correcto funcionamiento.

Ejemplo 2. Método de ordenamiento InsertionSort.

Este es uno de los métodos más sencillos. Consiste en tomar uno por uno los elementos de un arreglo y recorrerlo hacia su posición con respecto a los anteriormente ordenados. Así empieza con el segundo elemento y lo ordena con respecto al primero. Luego sigue con el tercero y lo coloca en su posición ordenada con respecto a los dos anteriores, así sucesivamente hasta recorrer todas las posiciones del arreglo.

Este método se emplea con frecuencia para ordenar cartas en juegos de mano: se considera un elemento a la vez insertándolo en su lugar correspondiente entre aquellos que ya han sido considerados (manteniéndolos ordenados). El elemento se inserta moviendo los elementos superiores una posición a la derecha y ocupando la posición vacante.

Para su implementación, debemos agregar la siguiente librería de clase:

```
using System.Diagnostics;
```

Esta librería es necesaria para el uso de la clase “**Stopwatch**”, clase para calcular tiempos.

Una instancia de Stopwatch puede medir el tiempo transcurrido para un intervalo o el total de tiempo transcurrido entre varios intervalos.

En un escenario de Stopwatch habitual, se llama al método “Start”, en otro momento se llama al método “Stop” y, por último, se comprueba el tiempo transcurrido mediante la propiedad “Elapsed”.

La codificación del algoritmo InsertionSort es la siguiente:

```
static void Ordenamiento_Insercion (int [ ] array)
{      for (int i = 0; i < array.length; i++)
       {   int temp = array [i];
           int j = i - 1;

           while ((j >= 0) && (array [j] > temp))
           {    array [j + 1] = array [j];
                j--;
           }
          array [j + 1] = temp;
       }
}
```

Análisis de resultados

Ejercicio No. 1

Implementar el simulador del método de ordenamiento Inserción, de tal manera que la solución permita recibir los datos y ordenarlos a partir de una interfaz gráfica de formulario (Windows Forms). Debe tomarse como referencia el ejemplo No.1 desarrollado en esta guía.

Ejercicio No. 2

Modificar el programa del ejemplo No. 1 y el programa del ejercicio No. 1, de tal manera que los simuladores muestren al usuario el tiempo que se demora el ordenamiento de los datos ingresados.

Investigación Complementaria

Para la siguiente semana:

Elaborar un programa en entorno gráfico (Windows Forms) para ordenar datos numéricos a partir del algoritmo de **Selección (SelectionSort)**. Debe implementarse como un simulador del método, es decir debe considerarse que debe visualizarse el ordenamiento. Tomar como referencia el algoritmo visto en clase y el ejemplo No.1 desarrollado en esta Guía.

Guía 3: Métodos de Ordenamiento. Parte 1.

Hoja de cotejo: 3

Alumno: Máquina No:

Docente: GL: Fecha:

EVALUACIÓN					
	%	1-4	5-7	8-10	Nota
CONOCIMIENTO	Del 20 al 30%	Conocimiento deficiente de los fundamentos teóricos	Conocimiento y explicación incompleta de los fundamentos teóricos	Conocimiento completo y explicación clara de los fundamentos teóricos	
APLICACIÓN DEL CONOCIMIENTO	Del 40% al 60%				
ACTITUD	Del 15% al 30%	No tiene actitud proactiva.	Actitud propositiva y con propuestas no aplicables al contenido de la guía.	Tiene actitud proactiva y sus propuestas son concretas.	
TOTAL	100%				