Facultad: Ingeniería
Escuela: Computación
Asignatura: Programación IV

Tema: Métodos de Ordenamiento. Parte 2.

Objetivos Específicos

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

Materiales y Equipo

- Guía Número 4.
- Computadora con programa Microsoft Visual C#.

Introducción Teórica

Método de ordenamiento ShellSort.

El método se denomina Shell en honor de su inventor Donald Shell.

El método ShellSort es una generalización del ordenamiento por inserción, teniendo en cuenta dos observaciones:

1) El ordenamiento por inserción es eficiente si la entrada está “casi ordenada”.
2) El ordenamiento por inserción es ineficiente, en general, porque mueve los valores sólo una posición cada vez.

El algoritmo ShellSort mejora el ordenamiento por inserción comparando elementos separados por un espacio de varias posiciones. Esto permite que un elemento haga “pasos más grandes” hacia su posición esperada.

Los pasos múltiples sobre los datos se hacen con tamaños de espacio cada vez más pequeños. El último paso del ShellSort es un simple ordenamiento por inserción, pero para entonces, ya está garantizado que los datos del vector están casi ordenados.

Algoritmo.

```
Inicio
inc = vector / 2
Repetir mientras inc > 0
    Hacer inc+1
    Para i = inc+1, condición i < (Número de elementos de lista), paso i+1
        Hacer j = i - inc
        Repetir mientras j > 0
            Hacer j+1
            Hacer
            Si lista [j] > lista [j+inc] entonces Hacer
                aux = lista [j]
                lista [j] = lista [j+inc]
                lista [j+inc] = aux
                j = j - inc
            Sino
                j = 0
        Fin del ciclo Mientras
    Fin del ciclo Para
    Hacer inc = inc / 2
Fin del ciclo Mientras
```

QuickSort, método de ordenamiento rápido.

El método de ordenamiento QuickSort es actualmente el más eficiente y veloz de los métodos de ordenación interna.

Este método es una mejora sustancial del método de intercambio directo y recibe el nombre de QuickSort por la velocidad con que ordena los elementos del arreglo.

Es un algoritmo basado en la técnica de divide y vencerás, que permite, en promedio, ordenar “n” elementos en un tiempo proporcional a “n Log n”.

Descripción del algoritmo.

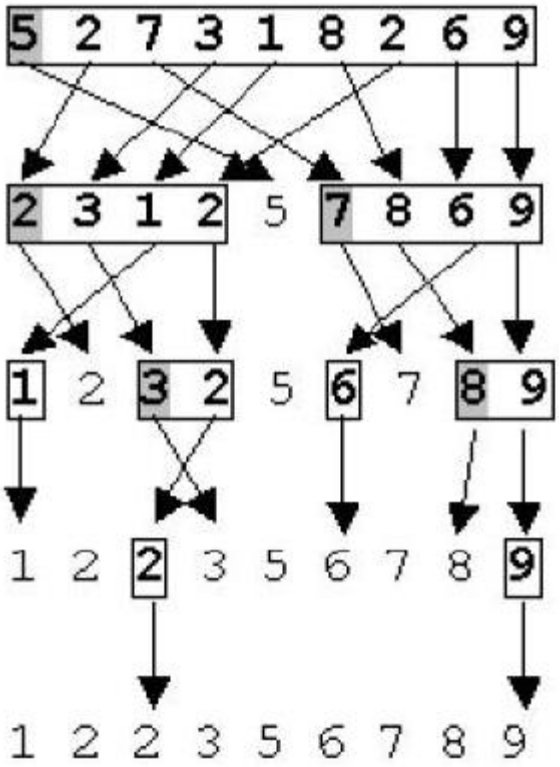
Elegir un elemento de la lista de elementos a ordenar, al que llamaremos “pivote”.

La idea central de este algoritmo consiste en lo siguiente:

- Se toma un elemento “x” de una posición cualquiera del arreglo. Se trata de ubicar a “x” en la posición correcta del arreglo, de tal forma que todos los elementos que se encuentran a su izquierda sean menores o iguales a “x” y todos los elementos que se encuentren a su derecha sean mayores o iguales a “x”.
- Se repiten los pasos anteriores pero ahora para los conjuntos de datos que se encuentran a la izquierda y a la derecha de la posición correcta de “x” en el arreglo.
- Reubicar los demás elementos de la lista a cada lado del pivote, de manera que a un lado queden todos los menores que él, y al otro los mayores. En este momento, el pivote ocupa exactamente el lugar que le corresponderá en la lista ordenada.
- Repetir este proceso de forma recursiva para cada sublista mientras éstas contengan más de un elemento. Una vez terminado este proceso todos los elementos estarán ordenados.

Como se puede suponer, la eficiencia del algoritmo depende de la posición en la que termine el pivote elegido.

Ejemplo:



MergeSort, el ordenamiento por mezcla.

La estrategia de este algoritmo se basa en el principio de divide y vencerás:

a. Partir la lista por la mitad.
b. Ordenar la sublista de la izquierda.
c. Ordenar la sublista de la derecha.
d. Mezclar las dos sublistas ordenadas en una sola lista ordenada.

La fusión de vectores permite un método de ordenación rápida y potente conocido por el nombre de ordenación por fusión (MergeSort).
La idea es dividir el vector en dos mitades, a continuación ordenar cada mitad y después mezclar ambas mitades para obtener un nuevo vector ordenado.
El algoritmo admite una formulación recursiva:

- Dividir el vector en dos mitades.
- Ordenar la mitad izquierda.
- Ordenar la mitad derecha.
- Mezclar las dos mitades juntas.

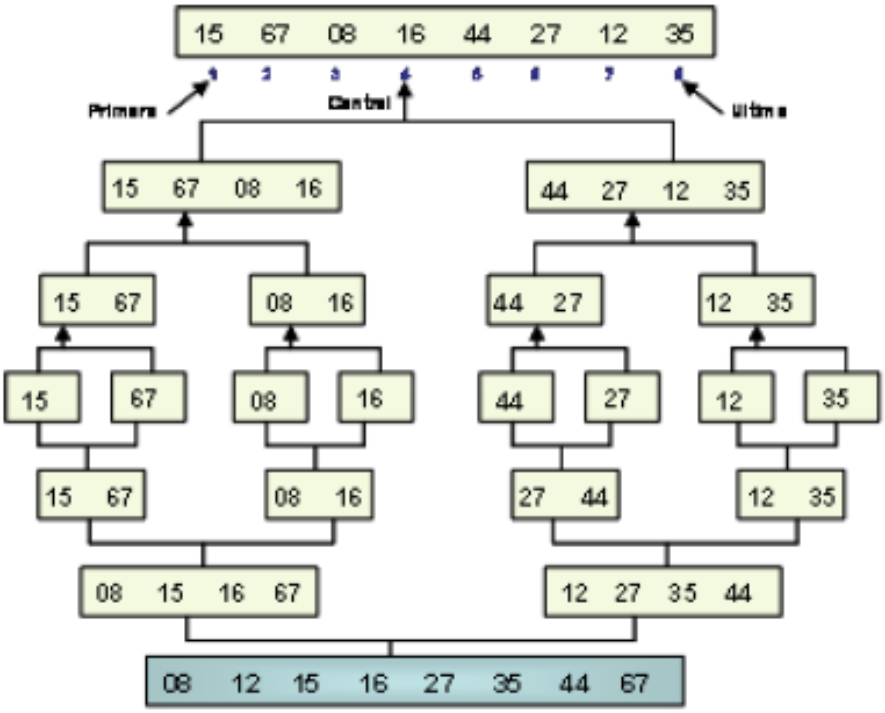
Procedimiento MergeSort (A, Primero, Ultimo).

A: arreglo lineal
Primero: índice al primer elemento
Ultimo: índice al último elemento

```
Si A [Primero] < A [Ultimo] entonces
        Central = (Primero + Ultimo) / 2
        MergeSort (A, Primero, Central)
        MergeSort (A, Central + 1, Ultimo)
        Mezcla A [Primero… central] con A [central+1…Ultimo]
Fin Si
```

Donde los parámetros del algoritmo Mezcla han sido modificados convenientemente.

Ejemplo:



Procedimiento

Ejemplo 1. Método de ordenamiento ShellSort.

Guardar con el nombre “Shell”.

Crear una función llamada Sort dentro de la clase llamada ShellSorter en medio de

```
using System;
using System.Collections.Generic;
using System.Text;
```

y

```
namespace shell
{ ..... }
```

Digitar el siguiente código:

```
public class ShellSorter
{       public void Sort (int [ ] list)
        {   int j, inc;
            inc = list.Length / 2;
            while (inc >= 1)
```

```
        {   for (int i = inc; i < list.Length; i++)
            {    int v = list [ i ];
                 j = i - inc;
                 while (j >= 0 && list [ j ] > v)
                 {     list [j + inc] = list [ j ];
                       j = j - inc;
                 }
                 list [j + inc] = v;
            }
            inc = inc / 2;
        }
    }
}
```

Adentro de las llaves de la siguiente sintaxis.

```
static void Main (string [ ] args) {
}
```

Digitar el siguiente código:

```
int [ ] iArrary = new int [ ] {22, 15, 3, 16, 30, 45, 9, 27, 7, 12, 44, 85, 24, 47, 33, 67, 55, 1, 93,
16};
System.Console.WriteLine ("Metodo de Ordenamiento: Shell ");
ShellSorter sh = new ShellSorter ( );
sh.Sort (iArrary);
System.Console.WriteLine (" Se imprimen: ");
for (int m = 0; m <= 19; m++)
        Console.WriteLine ("Valor {0}: {1}", m + 1, iArrary [m]);
Console.Read ( );
```

Ejemplo 2. Método de ordenamiento QuickSort.

Escribir el siguiente código:

```
using System;
namespace quicksort
{       class Class1
        {       static void Main ( )
                {       int n;
                        Console.WriteLine ("Ingrese longitud del arreglo n: ");
                        n = Int32.Parse (Console.ReadLine ( ));
                        Cllenar b = new Cllenar (n);
                }
        }

        class Cllenar
        {       int h;
                int [ ] vector;
                public Cllenar (int n)
                {       h = n;
                        vector = new int [h];
                        for (int i = 0; i < h; i++)
                        {       Console.WriteLine ("Ingrese valor {0}:", i + 1);
                                vector [i] = Int32.Parse (Console.ReadLine ( ));
                        }
                        quicksort (vector, 0, h - 1);
                        mostrar ( );
                }

                private void quicksort (int [ ] vector, int primero, int ultimo)
                {       int i, j, central;
                        double pivote;
                        central = (primero + ultimo) / 2;
                        pivote = vector [central];
                        i = primero;
                        j = ultimo;
```

```
                    do
                    {       while (vector [ i ] < pivote)    i++;
                            while (vector [ j ] > pivote)     j--;
                            if (i <= j)
                            {       int temp;
                                    temp = vector [ i ];
                                    vector [ i ] = vector [ j ];
                                    vector [ j ] = temp;
                                    i ++;
                                    j --;
                            }
                    }
                    while (i <= j);

                    if (primero < j)
                    {       quicksort (vector, primero, j);
                    }
                    if (i < ultimo)
                    {       quicksort (vector, i, ultimo);
                    }
            }

            private void mostrar ( )
            {       for (int i = 0; i < h; i++)
                    {       Console.Write ("{0} ", vector [ i ]);
                    }
                    Console.ReadLine ( );
            }
        }
} // Fin del namespace
```

Ejemplo 3. Método de ordenamiento por mezcla. MergeSort.

Escribir el siguiente código:

```
using System;
using System.Collections.Generic;
using System.Text;

namespace mergesort
{       class Class1
        {       static void Main ( )
                {       int n;
                        Console.WriteLine (" Ingrese longitud del arreglo n: ");
                        n = Int32.Parse (Console.ReadLine ( ));
                        Cllenar b = new Cllenar (n);
                }
        }

        class Cllenar
        {       int h;
                int [ ] vectorA;
                int [ ] vectorB;

                public Cllenar (int n)
                {       h = n;
                        vectorA = new int [h];
                        vectorB = new int [h];
                        for (int i = 0; i < h; i++)
                        {       Console.WriteLine ("Ingrese valor {0}:", i + 1);
                                vectorA [ i ] = Int32.Parse (Console.ReadLine ( ));
                        }
                        merge_sort (0, h - 1, vectorA, vectorB);
                        mostrar ( );
                }

                public void merge_sort (int left, int right, int [ ] vectorA, int [ ] vectorB)
                {       int mid;
```

```
        if (right > left)
        {       mid = (right + left) / 2;
                merge_sort (left, mid, vectorA, vectorB);
                merge_sort (mid + 1, right, vectorA, vectorB);
                merge (left, mid + 1, right);
        }
}

public void merge (int left, int mid, int right)
{       int i, left_end, num_elements, tmp_pos;
        left_end = mid - 1;
        tmp_pos = left;
        num_elements = right - left + 1;
        while ((left <= left_end) && (mid <= right))
        {       if (vectorA [left] <= vectorA [mid])
                {       vectorB [tmp_pos] = vectorA [left];
                        tmp_pos = tmp_pos + 1;
                        left = left + 1;
                }
                else
                {       vectorB [tmp_pos] = vectorA [mid];
                        tmp_pos = tmp_pos + 1;
                        mid = mid + 1;
                }
        }
        while (left <= left_end)
        {       vectorB [tmp_pos] = vectorA [left];
                left = left + 1;
                tmp_pos = tmp_pos + 1;
        }
        while (mid <= right)
        {       vectorB [tmp_pos] = vectorA [mid];
                mid = mid + 1;
                tmp_pos = tmp_pos + 1;
        }
```

```
                    for (i = 0; i < num_elements; i++)
                    {       vectorA [right] = vectorB [right];
                            right = right - 1;
                    }
            }
            private void mostrar ( )
            {       for (int i = 0; i < h; i++)
                    {       Console.Write ("{0} ", vectorA [ i ]);
                    }
                    Console.ReadLine ( );
            }
        }
    }   // Fin del namespace mergesort
```

Análisis de resultados

Ejercicio No. 1

Modificar el programa del ejemplo No. 1, de tal manera que la solución se implemente a través de un simulador en una interfaz gráfica de formulario (Windows Forms). El simulador debe mostrar también el tiempo que se demora el ordenamiento de los datos.

Ejercicio No. 2

Modificar el programa del ejemplo No. 2, de tal manera que la solución se implemente a través de un simulador en una interfaz gráfica de formulario (Windows Forms). El simulador debe mostrar también el tiempo que se demora el ordenamiento de los datos.

Ejercicio No. 3

Modificar el programa del ejemplo No. 3, de tal manera que la solución se implemente a través de un simulador en una interfaz gráfica de formulario (Windows Forms). El simulador debe mostrar también el tiempo que se demora el ordenamiento de los datos.

Investigación Complementaria

Para la siguiente semana:

1. Realice las modificaciones necesarias a los tres métodos estudiados, para que funcionen en el ordenamiento de arreglos tanto de números como de letras.
   La solución debe elaborarse como un simulador de cada método de ordenamiento en una interfaz gráfica de formulario (Windows Forms).

2. Investigar sobre el uso del objeto **Timer**. Determinar e indicar cómo podrían modificarse los simuladores elaborados en las guías Nos. 2 y 3 haciendo uso de este objeto.

Guía 4: Métodos de Ordenamiento. Parte 2.

Hoja de cotejo: 4

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%				