Algoritmos http://www.lfcia.org/alg/ 2007-2008

Ingeniería Informática

Ingeniería Técnica de Informática de Gestión

Práctica 2

Ordenación por inserción

El problema consiste en ordenar ascendentemente un vector de n números enteros. Como algoritmo de ordenación se utilizará la *ordenación por inserción*:

```
procedimiento Ordenación por inserción (var a[1..n])
  para i := 2 hasta n hacer
    x := a[i] ;
    j := i-1 ;
    mientras j > 0 y a[j] > x hacer
      a[j+1] := a[j] ;
      j := j-1
    fin mientras ;
    a[j+1] := x
  fin para
fin procedimiento
```

1. Implemente la interfaz `Comparable` y la clase `Entero` cuyas instancias se utilizarán como datos para la ordenación (figuras 1 y 2).

```
public interface Comparable {
    public boolean esIgualA(Comparable x);
    public boolean esMayorQue(Comparable x);
    public boolean esMenorQue(Comparable x);
}
```

Figura 1: La interfaz `Comparable`

2. Implemente el algoritmo de ordenación por inserción (figuras 3 y 4)
3. Valide el correcto funcionamiento de este método de ordenación (figura 5).
4. Determine los tiempos de ejecución para distintos valores de n $(500, 1000, 2000, 4000, \ldots)$ y para tres diferentes situaciones iniciales: (a) el vector ya está ordenado en orden ascendente, (b) el vector ya está ordenado en orden descendente, y (c) el vector está inicialmente desordenado (figuras 6 y 7).
5. Calcule empíricamente la complejidad de ambos algoritmos para cada una de las diferentes situaciones iniciales del vector (figura 8).

```
public class Entero implements Comparable {
    private int valor;
    public Entero(int i) {
        valor = i;
    }
    public boolean esIgualA(Comparable x) {
        Entero e = (Entero) x;
        return (valor == e.valor);
    }
    // Definición de esMayorQue y esMenorQue ...
    public String toString() {
        return (new Integer(valor)).toString();
    }
}
```

Figura 2: La clase `Entero`

```
public interface Ordenacion {
    public void ordenar(Comparable [] v);
}
```

Figura 3: La interfaz `Ordenacion`

```
public class OrdenacionPorInsercion implements Ordenacion {
    public void ordenar(Comparable [] v) {
        int i, j;
        Comparable x;
        for (i=1; i<v.length; i++) {
            j = i;
            x = v[i];
            while (j>0 && v[j-1].mayorQue(x)) {
                v[j] = v[j-1];;
                j--;
            }
            v[j] = x;
        }
    }
    public String toString() {
        return "Ordenacion por Insercion";
    }
}
```

Figura 4: La clase `OrdenacionPorInsercion`

```
** Ordenacion por Insercion
* Inicializacion aleatoria
antes: 99968702 1324958530 -1910779118 -577952317 1162223988
1724350983 1002321388 -394956363 -2053810004 -1235697473 1051462807
-552159116 -1689891144 -882630870 1756383891
despues: -2053810004 -1910779118 -1689891144 -1235697473 -882630870
-577952317 -552159116 -394956363 99968702 1002321388 1051462807
1162223988 1324958530 1724350983 1756383891
ordenado? true
* Inicializacion descendente
antes: 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1
despues: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
ordenado? true
```

Figura 5: Test

```
public interface InicializacionDeVectores {
    void inicializar(Entero [] v);
}
```

Figura 6: La interfaz `InicializacionDeVectores`

```
import java.util.*;
public class Aleatoria implements InicializacionDeVectores {
    Random rand = new Random();
    public void inicializar(Entero [] v) {
        for (int i=0; i<v.length; i++)
            v[i] = new Entero(rand.nextInt());
    }
    public String toString() {
        return "Inicializacion aleatoria";
    }
}
```

Figura 7: la clase `Aleatoria`

```
Inicializacion descendente, Ordenacion por Insercion

     n      T(n)     T(n)/n^1,5     T(n)/n^1,8     T(n)/n^2,0     T(n)/n^2,2     T(n)/n^2,5
   500     2,782(*)  0,00024883     0,00003857     0,00001113     0,00000321     0,00000050
  1000    11,000     0,00034785     0,00004379     0,00001100     0,00000276     0,00000035
  2000    46,000     0,00051430     0,00005259     0,00001150     0,00000251     0,00000026
  4000   182,000     0,00071942     0,00005975     0,00001138     0,00000217     0,00000018
  8000   733,000     0,00102440     0,00006911     0,00001145     0,00000190     0,00000013
 16000 2.922,000     0,00144378     0,00007912     0,00001141     0,00000165     0,00000009
```

Figura 8: Parte de la posible salida por pantalla de la ejecución del programa principal