Algoritmos elementales para ordenar un vector

1. Introducción

Una de las operaciones más importantes (y demandantes) en computación es *ordenar* un cierto conjunto de datos. Para nuestros propósitos de cálculo científico, tal colección de datos es simplemente un conjunto de N valores numéricos almacenados en un *arreglo unidimensional* (*vector)* A.
Dicho vector estará en *orden ascendente* si

$$i < j \text{ implica que } A(i) <= A(j)$$

para todos los elementos del vector.

En este apunte presentamos tres métodos elementales para ordenar un vector en orden ascendente, a saber, el *método de burbuja*, el *método de selección* y el *método de inserción*. Pero antes de presentarlos tenemos que enfatizar que éstos métodos *no* son los adecuados para uso general, sobre todo si el vector consta de una gran número de elementos (digamos, N > 50).

Para aclarar esto tenemos que hablar de la *eficiencia* de los métodos de ordenación. La mejor manera de medir el rendimiento de un algoritmo de ordenación consiste en contar el número de comparaciones entre elementos utilizados para ordenar un vector de N elementos. Un algoritmo de ordenación será más eficiente cuanto menor sea tal número de comparaciones. En el caso de los algoritmos mencionados puede demostrarse que tal número de comparaciones es proporcional a N2. Así, si se duplica el tamaño del vector, el número de comparaciones se cuadriplica. Por otra parte, los mejores algoritmos de ordenación tienen un número de comparaciones proporcional a N logN. Claramente estos algoritmos son computacionalmente más eficientes que los algoritmos elementales presentados aquí.

2. Método de burbuja

El algoritmo de ordenación de burbuja ("*bubble sort*") se basa en comparar los elementos adyacentes del vector e intercambiar los mismos si están desordenados. Se comienza comparando el primer elemento con el segundo, si están desordenados se intercambian. Luego se compara el segundo con el tercero, intercambiándolos si están desordenados. Este proceso que se realiza sobre todos los elementos constituye una *pasada* sobre el vector. Al terminar esta pasada el mayor elemento se encuentra al final del vector y algunos de los elementos más pequeños se han movido hacia las primeras posiciones (es decir, los elementos más pequeños han "burbujeado" hacia arriba, mientras que los más grandes se han "hundido", de aquí el nombre del método). Ahora se vuelve a repetir el procedimiento, pero sin comparar el mayor elemento que ya se encuentra en su posición correcta. Al terminar esta segunda pasada se habrá logrado poner el segundo mayor elemento en la posición correcta. El procedimiento se vuelve a repetir hasta obtener el vector ordenado, lo que ocurrirá cuando se hayan realizado N – 1 pasadas.

```
Algoritmo ordenamientoBurbujeo
   Definir vec, x, aux Como Entero
   Dimension vec[10]
   Definir flag Como Logico

   flag <- Falso

   Para x <- 0 Hasta 9 Hacer
         vec[x] <- azar(100) + 1
   FinPara

   Mientras flag = Falso Hacer
         flag <- Verdadero

         Para x <- 0 Hasta 8 Hacer
               Si vec[x] > vec[x+1] Entonces
                     aux <- vec[x]
                     vec[x] <- vec[x+1]
                     vec[x+1] <- aux
                     flag <- Falso
               FinSi
         FinPara
   FinMientras

   Para x <- 0 Hasta 9 Hacer
         Escribir vec[x]
   FinPara
FinAlgoritmo
```

3. Método de selección

El algoritmo de ordenación por selección procede encontrando el mayor elemento del vector e intercambiando su posición con el último elemento. A continuación se repite el procedimiento sobre los elementos 1... N-1, y así sucesivamente.

```
Algoritmo ordenamientoSeleccion
   Dimension vec[10]
   Definir vec, x, i, p, aux, j Como Entero

   Para x <- 0 Hasta 9 Hacer
         vec[x] <- azar(100) + 1
   FinPara

   Para i<-1 Hasta 9 Hacer
         aux <- vec[i]
         p <- i
         Para j <- i Hasta 9 Hacer
               SI vec[j] < aux Entonces
                     aux <- vec[j]
                     p <- j
               FinSi
         FinPara

         vec[p] <- vec[i]
         vec[i] <- aux

   FinPara

   Para x <- 0 Hasta 9 Hacer
         Escribir vec[x]
   FinPara

FinAlgoritmo
```

4. Método de inserción

El algoritmo de ordenación por inserción procede sobre cada elemento insertándolo en el lugar que le corresponde a la vez que desplaza los siguientes. Este es el método usual en que un jugador de cartas ordena las mismas.

```
Algoritmo ordenamientoInsercion
    Definir vec, x, p, aux, i Como Entero
    Dimension vec[10]

    Para x <- 0 Hasta 9 Hacer
        vec[x] <- azar(100) + 1
    FinPara

    Para i <- 1 Hasta 9 Hacer
        aux <- vec[i]
        p <- i - 1

        Mientras aux < vec[p] Y p >= 1 Hacer
            vec[p+1] <- vec[p]
            p <- p - 1
        FinMientras

        Si vec[p] <= aux Entonces
            vec[p+1] <- aux
        SiNo
            vec[p+1] <- vec[p]
            vec[p] <- aux
        FinSi
    FinPara

    Para x <- 0 Hasta 9 Hacer
        Escribir vec[x]
    FinPara

FinAlgoritmo
```