Metodos de Ordenamiento

Metodo de burbuja o metodo directo:

Funciona comenzando desde el inicio del vector, se compara cada par de elementos adyacentes. Si ambos no están ordenados (el segundo es menor que el primero), se intercambian sus posiciones. En cada iteración, un elemento menos necesita ser evaluados (el último), ya que no hay más elementos a su derecha que necesiten ser comparados, puesto que ya están ordenados.

Rendimiento:

n^2 comparaciones o intercambios de n elementos

Metodo de selección:

El algoritmo se desarrolla de la siguiente manera:

Se buscar en el vetor el mínimo elemento de la lista, luego cuando se encuentra el elemento se intercambia con el primero y se coloca en la primera posicion. A continuacion se busca el siguiente elemento mínimo en el resto de la lista y se lo intercambiar con la segunda posicion de la lista y asi sucesivamente hasta obtener todo el vector ordenado.

Rendimiento:

n^2 comparaciones o intercambios de n elementos

Metodo de insercion:

Este metodo consiste en ir insertando en la lista, el elemento que se evalua, según el peso del mismo, dentro de un conjunto de elementos menores a la cantidad total del vector. Entonces el procedimiento termina cuando se llego a examinar el ultimo elemento de la lista principal.

Rendimiento:

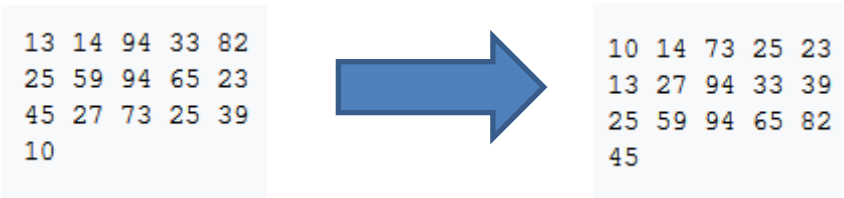
n^2 comparaciones o intercambios de n elementos.

Metodo shell:

comparando elementos separados por un espacio de varias posiciones. Esto permite que un elemento haga "pasos más grandes" hacia su posición esperada. Los pasos o incrementos múltiples sobre los datos se hacen con tamaños de espacio cada vez más pequeños. El último paso del Shell sort es un simple ordenamiento por inserción, pero para entonces, ya está garantizado que los datos del vector están casi ordenados.

Ejemplo:

13 14 94 33 82 25 59 94 65 23 45 27 73 25 39 10



10 14 73 25 23 13 27 94 33 39 25 59 94 65 82 45. A partir de aquí se resuelve como si se aplicara el metodo de insercion.

Rendimiento:

n^3/2 comparaciones o intercambios de n elementos

Metodo quicksort:

Permite, en promedio, ordenar *n* elementos en un tiempo proporcional a *n* log *n*. Se elege un elemento de la lista de elementos a ordenar, al que llamaremos pivote. Luego se colocan 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. Los elementos iguales al pivote pueden ser colocados tanto a su derecha como a su izquierda, dependiendo de la implementación deseada. En este momento, el pivote ocupa exactamente el lugar que le corresponderá en la lista ordenada. La lista queda separada en dos sublistas, una formada por los elementos a la izquierda del pivote, y otra por los elementos a su derecha. Se Repete 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.

Ejemplo:

En el siguiente ejemplo se marcan el pivote y los índices **i** y **j** con las letras **p**, **i** y **j** respectivamente.

Comenzamos con la lista completa. El elemento pivote será el 4:

```
5 - 3 - 7 - 6 - 2 - 1 - 4
                        p
```

Comparamos con el 5 por la izquierda y el 1 por la derecha.

```
5 - 3 - 7 - 6 - 2 - 1 - 4
i                   j   p
```

5 es mayor que 4 y 1 es menor. Intercambiamos:

```
1 - 3 - 7 - 6 - 2 - 5 - 4
i                   j   p
```

Avanzamos por la izquierda y la derecha:

```
1 - 3 - 7 - 6 - 2 - 5 - 4
    i           j       p
```

3 es menor que 4: avanzamos por la izquierda. 2 es menor que 4: nos mantenemos ahí.

```
1 - 3 - 7 - 6 - 2 - 5 - 4
        i       j       p
```

7 es mayor que 4 y 2 es menor: intercambiamos.

```
1 - 3 - 2 - 6 - 7 - 5 - 4
        i       j       p
```

Avanzamos por ambos lados:

```
1 - 3 - 2 - 6 - 7 - 5 - 4
          iyj           p
```

En este momento termina el ciclo principal, porque los índices se cruzaron. Ahora intercambiamos lista[i] con lista[p] (pasos 16-18):

```
1 - 3 - 2 - 4 - 7 - 5 - 6
          p
```

Aplicamos recursivamente a la sublista de la izquierda (índices 0 - 2). Tenemos lo siguiente:

```
1 - 3 - 2
```

1 es menor que 2: avanzamos por la izquierda. 3 es mayor: avanzamos por la derecha. Como se intercambiaron los índices termina el ciclo. Se intercambia lista[i] con lista[p]:

```
1 - 2 - 3
```

El mismo procedimiento se aplicará a la otra sublista. Al finalizar y unir todas las sublistas queda la lista inicial ordenada en forma ascendente.

```
1 - 2 - 3 - 4 - 5 - 6 - 7
```

Rendimiento:

n logn comparaciones o intercambios de n elementos