

MÉTODOS DE ORDENAMIENTO





Ordenamiento

Es la operación de arreglar los registros de un vector en algún orden secuencial de acuerdo a un criterio de ordenamiento. El ordenamiento se efectúa con base en el valor de algún campo en un registro, en nuestro caso trabajaremos con elementos sencillos, vectores numéricos. El propósito principal de un ordenamiento es el de facilitar las búsquedas de los componentes del conjunto ordenado.

El ordenar un grupo de datos significa mover los datos o sus referencias para que queden en una secuencia tal que represente un orden, el cual puede ser numérico, alfabético o incluso alfanumérico, ascendente o descendente.



Ordenamiento Burbuja (Bubblesort)

Este es el algoritmo más sencillo probablemente. Ideal para empezar.

Consiste en ciclar repetidamente a través de la lista, comparando elementos adyacentes de dos en dos. Si un elemento es mayor que el que está en la siguiente posición se intercambian.



Un ejemplo

Esta es nuestra lista:

4 - 3 - 5 - 2 - 1

Tenemos 5 elementos. Comenzamos comparando el primero con el segundo elemento. 4 es mayor que 3, así que intercambiamos. Ahora tenemos:

3 - 4 - 5 - 2 - 1

Ahora comparamos el segundo con el tercero: 4 es menor que 5, así que no hacemos nada. Continuamos con el tercero y el cuarto: 5 es mayor que 2. Intercambiamos y obtenemos:

3 - 4 - **2 - 5** - 1



Comparamos el cuarto y el quinto: 5 es mayor que 1.
Intercambiamos nuevamente:

3 - 4 - 2 - **1** - **5**

Repitiendo este proceso vamos obteniendo los siguientes resultados:

3 - 2 - 1 - 4 - 5

2 - 1 - 3 - 4 - 5

1 - 2 - 3 - 4 - 5

Análisis del algoritmo

Éste es el análisis para la versión no optimizada del algoritmo:

Estabilidad: Este algoritmo nunca intercambia registros con claves iguales. Por lo tanto es *estable*.

Requerimientos de Memoria: Este algoritmo sólo requiere de una variable adicional para realizar los intercambios.

Tiempo de Ejecución: El ciclo interno se ejecuta **n** veces para una lista de **n** elementos. El ciclo externo también se ejecuta **n** veces. Es decir, la complejidad es $n * n = O(n^2)$. El comportamiento del caso promedio depende del orden de entrada de los datos, pero es sólo un poco mejor que el del peor caso, y sigue siendo $O(n^2)$.



Ventajas:

Fácil implementación.

No requiere memoria adicional.

Desventajas:

Muy lento.

Realiza numerosas comparaciones.

Realiza numerosos intercambios.

Este algoritmo es uno de los más pobres en rendimiento. No es recomendable usarlo. Se tiene en cuenta por cuestiones académicas, y porque su sencillez lo hace bueno para empezar.



Ordenamiento por Selección

Este algoritmo también es sencillo. Consiste en lo siguiente:

Buscas el elemento más pequeño del vector.

Lo intercambias con el elemento ubicado en la primera posición del vector.

Buscas el segundo elemento más pequeño del vector.

Lo intercambias con el elemento que ocupa la segunda posición en el vector.

Repites este proceso hasta que hayas ordenado todo el vector.






Un ejemplo

Vamos a ordenar la siguiente lista (la misma del ejemplo anterior) :

4 - 3 - 5 - 2 - 1

Comenzamos buscando el elemento menor entre la primera y última posición. Es el 1. Lo intercambiamos con el 4 y la lista queda así:

1 - 3 - 5 - 2 - 4



Ahora buscamos el menor elemento entre la segunda y la última posición. Es el 2. Lo intercambiamos con el elemento en la segunda posición, es decir el 3. La lista queda así:

1 - **2** - 5 - **3** - 4

Buscamos el menor elemento entre la tercera posición y la última. Es el 3, que intercambiamos con el 5:

1 - 2 - **3** - **5** - 4

El menor elemento entre la cuarta y quinta posición es el 4, que intercambiamos con el 5:

1 - 2 - 3 - **4** - **5**



Análisis del algoritmo

Requerimientos de Memoria: Al igual que el ordenamiento burbuja, este algoritmo sólo necesita una variable adicional para realizar los intercambios.

Tiempo de Ejecución: El ciclo externo se ejecuta **n** veces para una lista de **n** elementos. Cada búsqueda requiere comparar todos los elementos no clasificados. Luego la complejidad es **$O(n^2)$** . Este algoritmo presenta un comportamiento constante independiente del orden de los datos. Luego la complejidad promedio es también **$O(n^2)$** .



Ventajas:

Fácil implementación.

No requiere memoria adicional.

Realiza pocos intercambios.

Rendimiento constante: poca diferencia entre el peor y el mejor caso.

Desventajas:

Lento.

Realiza numerosas comparaciones.

Este es un algoritmo lento. No obstante, ya que sólo realiza un intercambio en cada ejecución del ciclo externo, puede ser una buena opción para listas con registros grandes y claves pequeñas.

Ordenamiento por Inserción

Este algoritmo también es bastante sencillo. ¿Has jugado cartas?. ¿Cómo las vas ordenando cuando las recibes? Yo lo hago de esta manera: tomo la primera y la coloco en mi mano. Luego tomo la segunda y la comparo con la que tengo: si es mayor, la pongo a la derecha, y si es menor a la izquierda (también me fijo en el color, pero omitiré esa parte para concentrarme en la idea principal). Después tomo la tercera y la comparo con las que tengo en la mano, desplazándola hasta que quede en su posición final. Continúo haciendo esto, *insertando* cada carta en la posición que le corresponde, hasta que las tengo todas en orden.

Para simular esto en un programa necesitamos tener en cuenta algo: no podemos desplazar los elementos así como así o se perderá un elemento. Lo que hacemos es guardar una copia del elemento actual (que sería como la carta que tomamos) y desplazar todos los elementos mayores hacia la derecha. Luego copiamos el elemento guardado en la posición del último elemento que se desplazó.



Un ejemplo

4 - 3 - 5 - 2 - 1

La variable auxiliar toma el valor del segundo elemento, 3. La *primera carta* es el 4. Ahora comparamos: 3 es menor que 4. Luego desplazamos el 4 una posición a la derecha y después copiamos el 3 en su lugar.


4 - **4** - 5 - 2 - 1

3 - 4 - 5 - 2 - 1

El siguiente elemento es 5. Comparamos con 4. Es mayor que 4, así que no ocurren intercambios.

Continuamos con el 2. Es menor que cinco: desplazamos el 5 una posición a la derecha:

3 - 4 - 5 - **5** - 1



Comparamos con 4: es menor, así que desplazamos el 4 una posición a la derecha:

3 - 4 - **4** - 5 - 1

Comparamos con 3. Desplazamos el 3 una posición a la derecha:

3 - **3** - 4 - 5 - 1

Finalmente copiamos el 2 en su posición final:

2 - 3 - 4 - 5 - 1

El último elemento a ordenar es el 1. Cinco es menor que 1, así que lo desplazamos una posición a la derecha:

2 - 3 - 4 - 5 - **5**



Continuando con el procedimiento la lista va quedando así:

2 - 3 - 4 - **4** - 5

2 - 3 - **3** - 4 - 5

2 - **2** - 3 - 4 - 5

1 - 2 - 3 - 4 - 5



Análisis del algoritmo

Estabilidad: Este algoritmo nunca intercambia registros con claves iguales. Por lo tanto es *estable*.

Requerimientos de Memoria: Una variable adicional para realizar los intercambios.

Tiempo de Ejecución: Para una lista de **n** elementos el ciclo externo se ejecuta **n-1** veces. El ciclo interno se ejecuta como máximo una vez en la primera iteración, 2 veces en la segunda, 3 veces en la tercera, etc. Esto produce una complejidad $O(n^2)$.



Ventajas:

Fácil implementación.

Requerimientos mínimos de memoria.

Desventajas:

Lento.

Realiza numerosas comparaciones.

Este también es un algoritmo lento, pero puede ser de utilidad para listas que están ordenadas o semiordenadas, porque en ese caso realiza muy pocos desplazamientos