Métodos de Ordenamiento

Método Burbuja

Se basa en recorrer el array ("realizar una pasada") un cierto número de veces, comparando pares de valores que ocupan posiciones adyacentes (0-1,1-2, ...). Si ambos datos no están ordenados, se intercambian. Esta operación se repite n-1 veces, siendo n el tamaño del conjunto de datos de entrada. Al final de la última pasada el elemento mayor estará en la última posición; en la segunda, el segundo elemento llegará a la penúltima, y así sucesivamente.

```
public static void ordenarBurbuja(byte [] A){
    int i, j;
    byte aux;
    for(i=0;i<A.length-1;i++) {
         for(j=0;j<A.length-i-1;j++) {
              if(A[j+1]<A[j]){
                 aux=A[j+1];
                 A[j+1]=A[j];
                 A[j]=aux;
              }
         }
         mostrarVector(A);
         System.out.println();
    }
}
```

Muestra de ejecución

```
Mostramos el vector desordenado
 54  -7  50  -122  89  48  70  23  93  27

 -7  50  -122  54  48  70  23  89  27  93
 -7  -122  50  48  54  23  70  27  89  93
 -122  -7  48  50  23  54  27  70  89  93
 -122  -7  48  23  50  27  54  70  89  93
 -122  -7  23  48  27  50  54  70  89  93
 -122  -7  23  27  48  50  54  70  89  93
 -122  -7  23  27  48  50  54  70  89  93
 -122  -7  23  27  48  50  54  70  89  93
 -122  -7  23  27  48  50  54  70  89  93
Ha tardado I 1 milisegundos
```

Burbuja mejorada

Existe una forma muy obvia para mejorar el algoritmo de la burbuja. Basta con tener en cuenta la posibilidad de que el conjunto esté ordenado en algún paso intermedio. Si el bucle interno no necesita realizar ningún intercambio en alguna pasada, el conjunto estará ya ordenado.

```
private static void ordenarBurbujaV2(byte[] vector) {
    boolean bandera = true;
    for (int i = 0; i < vector.length-1 && bandera; i++) {
        bandera = false;
        for (int j = 0; j < vector.length-1-i; j++) {
            if(vector[j]>vector[j+1]) {
                byte aux = vector[j];
                vector[j] = vector[j+1];
                vector[j+1] = aux;
                bandera = true;
            }
        }
        mostrarVector(vector);
        System.out.println();
    }
}
```

```
Mostramos el vector desordenado
 54  -7  50  -122  89  48  70  23  93  27

 -7  50  -122  54  48  70  23  89  27  93
 -7  -122  50  48  54  23  70  27  89  93
 -122  -7  48  50  23  54  27  70  89  93
 -122  -7  48  23  50  27  54  70  89  93
 -122  -7  23  48  27  50  54  70  89  93
 -122  -7  23  27  48  50  54  70  89  93
 -122  -7  23  27  48  50  54  70  89  93
Ha tardado I 1 milisegundos
```

Método por intercambio

El algoritmo de ordenación tal vez más sencillo sea el denominado de intercambio, que ordena los elementos de una lista en orden ascendente. El algoritmo se basa en la lectura sucesiva de la lista a ordenar, comparando el elemento inferior de la lista con los restantes y efectuando un intercambio de posiciones cuando el orden resultante de la comparación no sea el correcto.

```
private static void ordenarPorIntercambio(byte[] vector) {
        byte aux;
        for (int i = 0; i < vector.length - 1; i++) {
            for (int j = (i+1); j < vector.length; j++) {
                if (vector[j] < vector[i]) {
                    aux = vector[i];
                    vector[i] = vector[j];
                    vector[j] = aux;
                }
            }
            mostrarVector(vector);
            System.out.println();
        }
}
```

```
Mostramos el vector desordenado
 54  -7  50  -122  89  48  70  23  93  27

 -122  54  50  -7  89  48  70  23  93  27
 -122  -7  54  50  89  48  70  23  93  27
 -122  -7  23  54  89  50  70  48  93  27
 -122  -7  23  27  89  54  70  50  93  48
 -122  -7  23  27  48  89  70  54  93  50
 -122  -7  23  27  48  50  89  70  93  54
 -122  -7  23  27  48  50  54  89  93  70
 -122  -7  23  27  48  50  54  70  93  89
 -122  -7  23  27  48  50  54  70  89  93
Ha tardado I 2 milisegundos
```

Método de selección

Este método considera que el array está formado por 2 partes: una parte ordenada (la izquierda) que estará vacía al principio y al final comprende todo el array; y una parte desordenada (la derecha) que al principio comprende todo el array y al final estará vacía. El algoritmo toma elementos de la parte derecha y los coloca en la parte izquierda.

Empieza por el menor elemento de la parte desordenada y lo intercambia con el que ocupa su posición en la parte ordenada. Así, en la la primera iteración se busca el menor elemento y se intercambia con el que ocupa la posición 0; en la segunda, se busca el menor elemento entre la posición 1 y el final y se intercambia con el elemento en la posición 1. De esta manera las dos primeras posiciones del array están ordenadas y contienen los dos elementos menores dentro del array. Este proceso continúa hasta ordenar todos los elementos del array.

```java
private static void ordenarSeleccion(byte[] vector) {

    int  menor, pos;
    byte aux;
    for (int i = 0; i < vector.length - 1; i++) {      // tomamos como menor el primero
          menor = vector[i];                           // de los elementos que quedan por ordenar
          pos = i;                                     // y guardamos su posición
          for (int j = i + 1; j < vector.length; j++){ // buscamos en el resto
                if (vector[j] < menor) {               // del array algún elemento
                    menor = vector[j];                 // menor que el actual
                    pos = j;
                }
          }
          if (pos != i){                       // si hay alguno menor se intercambia
              aux = vector[i];
              vector[i] = vector[pos];
              vector[pos] = aux;
          }
          mostrarVector(vector);
          System.out.println();
    }
}
```

```
Mostramos el vector desordenado
 54   -7   50   -122   89   48   70   23   93   27

 -122   -7   50   54   89   48   70   23   93   27
 -122   -7   50   54   89   48   70   23   93   27
 -122   -7   23   54   89   48   70   50   93   27
 -122   -7   23   27   89   48   70   50   93   54
 -122   -7   23   27   48   89   70   50   93   54
 -122   -7   23   27   48   50   70   89   93   54
 -122   -7   23   27   48   50   54   89   93   70
 -122   -7   23   27   48   50   54   70   93   89
 -122   -7   23   27   48   50   54   70   89   93
Ha tardado I 1 milisegundos
```

Método por inserción

Se utiliza un método similar al anterior, tomando un elemento de la parte no ordenada para colocarlo en su lugar en la parte ordenada. El primer elemento del array (A[0]) se considerado ordenado (la lista inicial consta de un elemento). A continuación se inserta el segundo elemento (A[1]) en la posición correcta (delante o detrás de A[0]) dependiendo de que sea menor o mayor que A[0]. Repetimos esta operación sucesivamente de tal modo que se va colocando cada elemento en la posición correcta. El proceso se repetirá TAM-1 veces.

```
public static void ordenarInsercion (byte [] A) {
    int j;
    byte aux;
    for (int i = 1; i < A.length; i++){ // desde el segundo elemento hasta
              aux = A[i];               // el final, guardamos el elemento y
              j = i - 1;                // empezamos a comprobar con el anterior
              while ((j >= 0) && (aux < A[j])){ // mientras queden posiciones y el
                                                // valor de aux sea menor que los
                            A[j + 1] = A[j];    // de la izquierda, se desplaza a
                            j--;                // la derecha
              }
              A[j + 1] = aux;       // colocamos aux en su sitio
    }
}
```

```
Mostramos el vector desordenado
 54   -7   50   -122   89   48   70   23   93   27

 -7   54   50   -122   89   48   70   23   93   27
 -7   50   54   -122   89   48   70   23   93   27
 -122   -7   50   54   89   48   70   23   93   27
 -122   -7   50   54   89   48   70   23   93   27
 -122   -7   48   50   54   89   70   23   93   27
 -122   -7   48   50   54   70   89   23   93   27
 -122   -7   23   48   50   54   70   89   93   27
 -122   -7   23   48   50   54   70   89   93   27
 -122   -7   23   27   48   50   54   70   89   93
Ha tardado I 3 milisegundos
```

QuickSort

El método de ordenación Quicksort fue desarrollado por Hoare en el año 1960.
Es el algoritmo de ordenación más rápido.

Se basa en la técnica **divide y vencerás**, que consiste en ir subdividiendo el array en arrays más pequeños, y ordenar éstos. Para hacer esta división, se toma un valor del array como **pivote**, y se mueven todos los elementos menores que este pivote a su izquierda, y los mayores a su derecha. A continuación se aplica el mismo método a cada una de las dos partes en las que queda dividido el array.

Después de elegir el pivote se realizan dos búsquedas:

Una de izquierda a derecha, buscando un elemento mayor que el pivote

Otra de derecha a izquierda, buscando un elemento menor que el pivote.

Cuando se han encontrado los dos elementos anteriores, se intercambian, y se sigue realizando la búsqueda hasta que las dos búsquedas se encuentran.

La implementación del método de ordenación Quicksort es claramente recursiva.

Suponiendo que tomamos como pivote el primer elemento, el método Java Quicksort que implementa este algoritmo de ordenación para ordenar un array de enteros se presenta a continuación. Los parámetros izq y der son el primer y último elemento del array a tratar en cada momento.

El método ordena un array A de enteros desde la posición *izq* hasta la posición *der.* En la primera llamada recibirá los valores izq = 0, der = ELEMENTOS-1.

```
public static void quickSort(byte A[], int izq, int der) {
      byte pivote=A[izq]; // tomamos primer elemento como pivote
      int i=izq;          // i realiza la búsqueda de izquierda a derecha
      int j=der;          // j realiza la búsqueda de derecha a izquierda
      byte aux;

      while(i < j){                          // mientras no se crucen las búsquedas
         while(A[i] <= pivote && i < j) i++; // busca elemento mayor que pivote
         while(A[j] > pivote) j--;           // busca elemento menor que pivote
         if (i
             byte[] A - ordenarvector.MainOrdenarVector.quickSort(byte[], int, int)
             au                                                    rcambia
             A[
             A[
         }
       }

       A[izq]=A[j];      // se coloca el pivote en su lugar de forma que tendremos
       A[j]=pivote;      // los menores a su izquierda y los mayores a su derecha

       mostrarVector(A);
       System.out.println();

       if(izq < j-1)
          quickSort(A,izq,j-1);          // ordenamos subarray izquierdo
       if(j+1 < der)
          quickSort(A,j+1,der);          // ordenamos subarray derecho
   }
```

```
Mostramos el vector desordenado
 54  -7  50  -122  89  48  70  23  93  27

 23  -7  50  -122  27  48  54  70  93  89
 -122  -7  23  50  27  48  54  70  93  89
 -122  -7  23  50  27  48  54  70  93  89
 -122  -7  23  48  27  50  54  70  93  89
 -122  -7  23  27  48  50  54  70  93  89
 -122  -7  23  27  48  50  54  70  93  89
 -122  -7  23  27  48  50  54  70  89  93
Ha tardado I 2 milisegundos
```