MÉTODOS DE ORDENACIÓN

Consisten en clasificar los elementos según un orden determinado. La ordenación de cualquier lista de elementos está basada siempre en la comparación de dos elementos de esa lista e intercambiarlos si es necesario. Si los datos están ordenados es más fácil la localización de algún elemento. Existen diversos métodos de ordenación, dependiendo de si los datos están en memoria o se encuentran en un soporte.

Los métodos de ordenación interna deben gestionar la memoria de forma eficiente, para ello deben realizarse sobre las posiciones ocupadas, a excepción de alguna variable auxiliar para poder realizar el intercambio. Debemos tener en cuenta que no siempre los datos podrán están en memoria de forma completa para su ordenación.

Recordar: que los algoritmos son genéricos y sirven para cualquier lenguaje de programación. En función de cada lenguaje tendremos que cambiar o no el valor de inicio del array. Hay lenguajes con los arrays empiezan en 1 y otros empiezan en cero su primer elemento.

Existen tres grupos en cuanto a los métodos de ordenación:

1. **INTERCAMBIO.**
2. **SELECCIÓN.**
3. **INSERCION.**

ORDENACION POR INTERCAMBIO.

Consiste en comparar los elementos consecutivos e intercambiarlos, si no cumplen la condición deseada. De esta forma vamos arrastrando un valor hasta el final de la lista. El proceso se ira repitiendo, excluyendo los últimos elementos de la lista que ya están ordenados. Hasta que esta se encuentre ordenada. El método más sencillo y conocido es el **de la burbuja**, pero también es poco eficiente al realizar demasiadas operaciones para el ordenamiento, esto hace que sea más lento este método de ordenación.

N, tamaño de la tabla, nº entero. P, J, nº enteros.
tabla(N), array de nº enteros

```
Para P=1, N-1, 1
   Para J=1, N-P, 1
        Si  tabla[J] > tabla[J+1]
                intercambiar(tabla[J], tabla[J+1])
        Fin_Si
   Fin_Para
Fin_Para
```

El método de la burbuja puede mejorarse. Si en la estructura repetitiva interna no se produce ningún cambio, nuestro proceso debe detenerse al estar la lista ya ordenada.

N, tamaño de la tabla, nº entero. P, J, nº enteros.
tabla(N), array de nº enteros
sw, variable para detectar que se producen intercambios de elementos y debemos continuar con el proceso. Le asignamos el valor inicial de 1.

```
sw = 1      P = 0
Mientras  sw = 1
   sw = 0     P = P + 1
   Para J=1, N - P, 1
        Si  tabla[J] > tabla[J+1]
                intercambiar(tabla[J], tabla[J+1])
                sw =1
        Fin_Si
   Fin_Para
Fin_Mientras
```

ORDENACION POR SELECCION.

Este método consiste en:

a) Seleccionar el elemento más pequeño o grande del array. Para ello debe compararlo con todos los demás.
b) Colocarlo en la posición más baja o alta del array.
c) Repetir el proceso hasta ordenar la lista deseada.

Este método es más rápido que el de la burbuja, pero si la lista esta parcialmente ordenada podría ser más lento.

N, tamaño de la tabla, nº entero. P, J, nº enteros.
tabla(N), array de nº enteros

```
Para P=1, N-1, 1
   Para J=P+1, N, 1
       Si  tabla[P] > tabla[J]
             intercambiar(tabla[P], tabla[J])
       Fin_Si
   Fin_Para
Fin_Para
```

Mejora de este método, para realizar menos intercambios.

N, tamaño de la tabla, nº entero. P, J, nº enteros.
tabla(N), array de nº enteros
menor, entero, se almacena el menor valor de la lista y lo utilizamos para comparar con el resto de los valores de la lista.
posicion, entero, guardamos la posición del elemento menor para al final de un recorrido completo realizar el intercambio de valores.

```
Para P=1, N-1, 1
   menor = tabla[P]
   posicion = P
   Para J=P+1, N, 1
       Si  menor > tabla[J]
             menor = tabla[J]
             posicion = J
       Fin_Si
   Fin_Para
   Si posicion > P
       intercambiar(tabla[P], tabla[posicion])
   FinSi
Fin_Para
```

ORDENACION POR INSERCION.

Este método insertar los elementos no ordenados del array en subarrays del mismo que ya este ordenados. Existen diversos métodos para ello:

a) **INSERCIÓN DIRECTA.**

b) **MÉTODO SHELL.**

c) **ORDENACIÓN RAPIDA Ó QUICK SORT.**

INSERCION DIRECTA

Este método inserta los elementos no ordenados de la tabla en subtablas de la misma que ya están ordenadas. Existen diversas maneras de escribirlos

N, tamaño de la tabla, nº entero. P, J, nº enteros.
tabla(N), array de nº enteros
aux, valor tomado de la tabla sobre el que vamos a comparar.

```
Para P=2, N, 1
   aux = tabla[P]
   J = P - 1
   Mientras (tabla[J] > aux )  AND J > 1
         tabla[J+1] = tabla[J]
         J = J -1
   Fin_Mientras
   Si tabla[J] > aux
        tabla[J+1] = tabla[J]
        tabla[J] = aux
     sino
        tabla[J+1] = aux
    Fin_si
Fin_Para
```

MÉTODO SHELL

Este procedimiento es mucho más rápido que los vistos hasta ahora. En el se comparan entre sí los elementos que están separados a cierta distancia dentro de la lista. Cuando la comparación se termina, esa distancia se reduce a la mitad y se repite la comparación; el proceso continúa hasta que esa distancia queda reducida a cero. Se puede tomar un valor cualquiera como primer valor para la distancia entre los elementos que se comparan.

N, tamaño de la tabla, nº entero. P, J, nº enteros.
tabla(N), array de nº enteros
HUECO, K, SW enteros
HUECO, es la distancia que existe entre los valores para comparar. Aquí estamos tomando la mitad de los elementos.

```
HUECO = N/ 2
MIENTRAS   HUECO > 0
    PARA P= HUECO+1,   N,   1          *En Java, cambiar  HUECO+ 1  por HUECO.
    J = P – HUECO
      MIENTRAS   J > 0                 *En Java, J >= 0.
       K = J + HUECO
          SI  tabla[J]  <=  tabla[K]
             J = 0                     *En Java, J = - 1.
          SINO
             SW = tabla[J]
             tabla[J] = tabla[K]
             tabla[K] = SW
              J = J – HUECO
          FINSI
      FINMIENTRAS
    FINPARA
  HUECO = HUECO / 2
FINMIENTRAS
```

MÉTODO ORDENACIÓN RAPIDA (QUICK SORT)

Este procedimiento divide la lista que hay que ordenar en dos mitades, en la primera coloca todos los elementos que son menores que otro dado, en la segunda mitad, todos los mayores a él; después aplica el mismo procedimiento a las dos listas que se han formado, y así sucesivamente. Este procedimiento es recursivo.

Comienza el procedimiento eligiendo el elemento central, que se va a servir para colocar a un lado los que son menores que él y al otro los que son mayores. la forma más sencilla, aunque casi nunca es la que da mayor velocidad, consiste en elegir el que ocupa el lugar central de la lista. Las variables P y J son los índices que se desplazan por la lista que se va a ordenar, el primero desde el primer elemento hacia abajo y el segundo desde el último hacia arriba. Cada índice se detiene cuando encuentra un elemento que no está situado en la mitad que le corresponde. Los elementos marcados por esos índices son intercambiados. El proceso continúa hasta que los índices se cruzan; en ese momento la lista está dividida y puede aplicarse el procedimiento a cada una de las dos mitades.

MÉTODO ORDENACIÓN RAPIDA (QUICK SORT)

N, tamaño de la tabla, nº entero.
tabla(N), array de nº enteros
primero, ultimo, son nº enteros.

```
FUNCION ORDENAR (tabla, primero, ultimo)
    centro, P, J, son nº enteros.
    SI (ultimo - primero) > 1                    * En Java, preguntaremos por > 0.
        P = ultimo
        J = primero
        centro = tabla[(primero + ultimo)/2]
        MIENTRAS J <= P
          MIENTRAS tabla[J] < centro
                J = J + 1
          FIN_MIENTRAS
          MIENTRAS  tabla[P] > centro
                P = P – 1
          FIN_MIENTRAS
          SI J <=  P
              intercambiar (tabla[J], tabla[P])
              J = J + 1
              P = P – 1

           FIN_SI
         FIN_MIENTRAS
        ORDENAR(tabla, primero, P)
        ORDENAR(tabla, J, ultimo)
      SINO
        SI (ultimo – primero) = 1
            SI tabla[primero] > tabla[ultimo]
                  intercambiar ( tabla[primero], tabla[ultimo])
            FIN_SI
         FIN_SI
    FIN_SI
FIN_FUNCION
```