



Práctica 2

Ordenación Rápida

Resolución

[OrdenacionRapida.java](#)

[OrdenacionRapidaBarajada.java](#)

[BuscaElem.java](#)

[OrdenacionJava.java](#)

Antonio J. Galán Herrera

Ingeniería Informática (A)

Nota alcanzada en SIETTE: 100 / 100.

Ordenación Rápida

Los algoritmos de ordenación rápida aplican el procedimiento de *divide y vencerás* para, en este caso, ordenar una serie de números de menor a mayor.

Resolución

En todos los casos siguientes, cuando se dice que un vector está ordenado, quiere decir de menor a mayor.

OrdenacionRapida.java

1

Ordenar los elementos de un vector V .

```
public static <T extends Comparable<? super T>> void ordenar(T[] V) {
    ordRapidaRec(V, 0, V.length-1);
}
```

Inicia el proceso de ordenación de un vector V .

1. Aplicando el criterio de *divide y vencerás*, este método actúa como **paso inicial** a una llamada recursiva.
 - El primer paso inicia ordenando dentro de $[V_0, V_{L-1}]$, siendo L el tamaño de V .

```
public static <T extends Comparable <? super T>> void ordRapidaRec(T[] V, int inf, int sup) {
    // Caso intermedio
    if(inf < sup) {
        // Se ordena el sub-vector entre los límites inferior y superior
        // en base al pivote, que será el primer elemento del sub-vector
        int piv = partir(V, V[inf], inf, sup);

        ordRapidaRec(V, inf, piv-1);
        ordRapidaRec(V, piv+1, sup);
    }
}
```

Devuelve V ordenado o bien, lo sigue ordenando en la mitad inferior o superior.

1. **Se comprueba el caso base (implícito):** los extremos de búsqueda se han invertido.
 - Esto solo sucede cuando ya se ha ordenado V , por lo que no haría el caso intermedio.
2. **Se comprueba el caso intermedio:** los extremos de búsqueda no se han invertido.
 - Se parte el vector V en 2 sub-vectores, determinados en función del límite inferior actual.
 - Este método devuelve un pivote p .
 - Se ordena V entre el límite inferior y el anterior a p .
 - Se ordena V entre entre el siguiente a p y el límite superior.

```
public static <T extends Comparable <? super T>> int partir(T[] V, T piv, int inf, int sup) {
    int izq = inf, der = sup;

    while (izq < der) {
        while (0 < V[der].compareTo(piv) && 0 < der) {
            der--;
        }

        while (V[izq].compareTo(piv) <= 0 && izq < V.length-1) {
            izq++;
        }

        if (izq < der) {
            intercambiar(V, izq, der);
        }
    }

    if (0 < piv.compareTo(V[der])) {
        intercambiar(V, inf, der);
    }


    return der;
}
```

Ordena un intervalo de un vector V en base a un pivote p (siendo p un elemento de V), de forma que todos los elementos menores que él quedan a su izquierda.

1. Se crean `izq` y `der`, copias de los límites para conservar los originales (`inf` y `sup`).
2. Mientras que los extremos no se hayan cruzado:
 1. Mueve el puntero `der` hacia la izquierda, mientras que $p < V[\text{der}]$.
 2. Mueve el puntero `izq` hacia la derecha, mientras que $V[\text{izq}] \leq p$.
 3. Si los punteros no se han invertido, se intercambian $V[\text{izq}]$ y $V[\text{der}]$.
 - Esto sucede porque en ese punto, $V[\text{izq}] > V[\text{der}]$.
3. Si $V[\text{der}] < p$, se intercambian se intercambian $V[\text{inf}]$ y $V[\text{sup}]$.
 - Esto sucede porque los límites originales también deben quedar ordenados, y puede que no suceda debido al $=$ de la condición 2.2.
4. Finalmente, se devuelve `der`.
 - Resulta que tras todos los intercambios, `der` siempre coincide con el pivote de la siguiente iteración.

OrdenacionRapidaBarajada.java

- 2 Ordenar los elementos de un vector V , barajados aleatoriamente.

 Esta clase funciona exactamente igual que la anterior, excepto por el método `ordenar()`.

```
public static <T extends Comparable<? super T>> void ordenar(T[] V) {
    barajar(V);
    ordRapidaRec(V, 0, V.length-1);
}
```

Inicia el proceso de ordenación de un vector V .

1. Intercambia aleatoriamente los elementos de V .
2. Aplicando el criterio de *divide y vencerás*, este método actúa como **paso inicial** a una llamada recursiva.
 - El primer paso inicia ordenando dentro de $[V_0, V_{L-1}]$, siendo L el tamaño de V .

```
private static <T> void barajar(T[] V) {
    for (int i = V.length-1; i > 0; i--) {
        // Baraja la posición de un elemento con una posición anterior aleatoria
        intercambiar(V, i, aleat.nextInt(i+1));
    }
}
```

Baraja aleatoriamente los elementos de un vector V .

1. Para cada posición i desde $L - 1$ hasta 0, siendo L el tamaño de V :
 - Intercambia $V[i]$ por $V[k]$, siendo k un número al azar entre $[0, i]$.

BuscaElem.java

3

Encontrar el elemento en la posición k de un vector V si este estuviera ordenado. El proceso no debe ordenar el vector V .

```
public static <T extends Comparable <? super T>> T kesimo(T[] V, int k) {
    T[] W = Arrays.copyOf(V, V.length);

    return kesimoRec(W, 0, W.length-1, k);
}
```

Inicia el proceso de búsqueda del valor $V[k]$.

1. Se crea una copia del vector V porque en este caso no debe ordenarse el vector.
2. Aplicando el criterio de *divide y vencerás*, este método actúa como **paso inicial** a una llamada recursiva.
 - El primer paso inicia buscando k dentro de $[W_0, W_{L-1}]$, siendo L el tamaño de W .

```
public static <T extends Comparable <? super T>> T kesimoRec(T[] V, int inf, int sup, int k) {
    // Caso intermedio
    if (inf < sup) {
        int piv = OrdenacionRapida.partir(V, V[inf], inf, sup);

        if (k == piv) {
            return V[k];
        } else if (k < piv) {
            return kesimoRec(V, inf, piv-1, k);
        } else {
            return kesimoRec(V, piv+1, sup, k);
        }
    }

    return V[k];
}
```

Devuelve $V[k]$ o bien, lo sigue buscando en la mitad inferior o superior.

1. **Se comprueba el caso base (implícito):** los extremos de búsqueda se han invertido.
 - Esto solo sucede cuando ya se ha encontrado k , por lo que no haría el caso intermedio.
2. **Se comprueba el caso intermedio:** los extremos de búsqueda no se han invertido.
 - Se parte el vector W con el método `partir()` de la clase `OrdenacionRapida`.
 - Este método devuelve un pivote p .
 - a. Si $k = p$, ya se ha encontrado k y por tanto **se devuelve**.
 - b. Si $k < p$, se busca k entre el límite inferior y el anterior a p .

c. Si $k > p$, se busca k entre el siguiente a p y el límite superior.

OrdenacionJava.java

4

Ordenar los elementos de un vector V usando alguna estructura de `java.util`.

```
public static <T extends Comparable<? super T>> void ordenar(T[] V) {  
    Arrays.sort(V);  
}
```

Ordena un vector usando un método de la librería `java.util`.