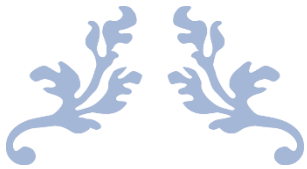

TECNICAS DE PROGRAMACION AVANZADAS

DIGITAL BLOCK 2

VICTOR PEREZ PEREZ

21/02/2021

Contenido

Comprobación de QuickSort 2

Código 2

Ordenación (un pivote) 4

Código 4

Ordenación (dos pivotes) 6

Código 6

Realizada Version2 con todos los métodos funcionales y sin errores

Comprobación de QuickSort

Código

```
package DigitalBlock2;

public class ComprobacionQuickSort {

	public static void main(String[] args) {
		/**
		 * creamos un array con 10 elementos
		 */
		double[] misDatos = {4,8,3,3,7,6,2,10,2,4};

		/**
		 * Llamamos a la funcion quickSort y le pasamos como parametros
		 * el array, la posicion inicial (0) y la final (array-1)
		 */
		Ordenacion.quickSort(misDatos,0,9);

		/**
		 * Aunque el enunciado decia que debiamos implementar la funcion
		 * de abajo, si le pasamos como parametros el 1 y el 10, el programa
		 * nos mostrara un error, ya que nuestro array va desde la posicion
		 * 0 hasta la 9, por lo que el 10 estaria fuera de esa dimension y
		 * produciria un error a la hora de ejecutarse
		 */
			//Ordenacion.quickSort(array,1,10);

		/**
		 * Creamos una variable con la longitud del array, y realizamos un bucle
		 * for con el objetivo de imprimir el array final una vez ordenado
		 * segun el metodo aplicado
		 */
		System.out.println("Primer apartado");
		int longitud = misDatos.length;
		for (int i=0;i<longitud;i++) {
			System.out.print(misDatos[i]+ "  ");
		}
		System.out.println("  ");


		System.out.println("Segundo apartado");

		/**
		 * creamos un array con 10 elementos
		 */
		int[] misDatos2 = {5,2,7,9,6,3,5,5,3,0};

		/**
		 * Llamamos a la funcion quickSortBis y le pasamos como parametros
		 * el array, la posicion inicial (0) y la final (array-1)
		 */
		Ordenacion2.quickSortBis(misDatos2,0,9);

		/**
```

```
                * Creamos una variable con la longitud del array, y realizamos un bucle
                * for con el objetivo de imprimir el array final una vez ordenado
                * segun el metodo aplicado
                */
               int longitud2 = misDatos2.length;
               for (int i=0;i<longitud2;i++) {
                        System.out.print(misDatos2[i]+ "  ");
               }
        }

}
```

Ordenación (un pivote)

Código

```
package DigitalBlock2;

public class Ordenacion {

	/**
	 * Esta funcion recibe como parametros el array, la posicion inicial (0) y la final (9)
	 */
	public static void quickSort(double[] array, int inicio, int fin) {
		/**
		 * Si la posicion inicial es mas pequeña que la final, entramos al bucle.
		 */
		if(inicio<fin) {
			/**
			 * Creamos una variable que trabajar como un pivote para distinguir los mayores
			 * y menores, pudiendo ordenar la lista de esa manera
			 */
			int pivote = pivotar(array, inicio, fin);
			/**
			 * La funcion se llama a si misma, pero ahora el array inicial se divide en dos partes:
			 *		la de los numeros mas grandes que el pivote y otra con los menores.
			 */
			quickSort(array, inicio, pivote-1);
			quickSort(array, pivote+1, fin);

		}
	}

	public static int pivotar(double[] array, int inicio, int fin) {
		/**
		 * Creamos variable pivote, que almacena el valor del inicio de la lista que le pasemos
		 *
		 * Ademas, una variable pivot, que almacenara el valor de la lista que este en la posicion
		 * de pivote (inicio)
		 */
		int pivote=inicio;
		double pivot = array[pivote];

		/**
		 * Si el inicio es mas pequeño que el fin, entramos al bucle
		 */
		if(inicio < fin) {
			/**
			 * Creamos bucle for:
			 *			donde j es el inicio mas 1, asi no coincide con el pivot
			 *			j tiene que ser siempre mas pequeño o igual que el fin
			 *			j avanza de uno en uno, para poder comparar cada posicion de la lista
			 */
			for(int j = inicio+1; j<=fin; j++) {
```

```
                                        /**
                                         * Si la posicion en la que estamos es mayor que el pivot,
aumentamos la variable pivote en uno
                                         */
                                        if(array[j] > pivot) {
                                                pivote++;
                                                /**
                                                 * Si la varible aumentada en uno (pivote), es diferente a la j,
procedemos a hacer un intercambio
                                                 * de posiciones en la lista, moviendo el elemento mayor
antes del pivote, y los menores despues
                                                 * de este.
                                                 * De esta manera conseguimos nuestra lista no creciente.
                                                 */
                                                if (pivote!=j) {
                                                        /**
                                                         * Creamos una variable para almacenar la posicion
pivote
                                                         * Igualamos el valor de esa posicion con el valor de
la posicion j
                                                         * Cambios el valor de la posicion j, igualandolo a la
variable que contiene pivote almancenada
                                                         */
                                                        double intercambio1 = array[pivote];
                                                        array[pivote] = array[j];
                                                        array[j] = intercambio1;
                                                }
                                        }
                                }
                                /**
                                 * Creamos una variable para almacenar la posicion del inicio
                                 * Igualamos el valor de esa posicion con el valor de la posicion pivote
                                 * Cambios el valor de la posicion pivote, igualandolo a la variable que
contiene inicio almancenada
                                 *
                                 * Asi, movemos el pivote para que se quede en su sitio correcto
                                 */
                                double intercambio2 = array[inicio];
                                array[inicio] = array[pivote];
                                array[pivote] = intercambio2;

                        }

                        /**
                         * Devolvemos la pivote, que es donde se encuentra nuestro pivote
                         * En la funcion quickSort, se denomina pivote
                         */
                        return pivote;
                }

}
```

Ordenación (dos pivotes)

Código

```
package DigitalBlock2;

public class Ordenacion2 {

	public static void quickSortBis(int[] array, int inicio, int fin) {
		/**
		 * Si la posicion inicial es mas pequeña que la final, entramos al bucle.
		 */
		if(inicio<fin) {
			/**
			 * Creamos una variable para poder almacenar la posicion del primer
			 * y ultimo pivote que encontremos en nuestra lista
			 *
			 * Rellenamos el array creado con el inicio
			 *
			 */
			int[]posicionesPivote = {inicio,inicio};
			pivotarBis(array, inicio, fin, posicionesPivote);
			/**
			 * La funcion se llama a si misma, pero ahora el array inicial se divide en dos
partes:
			 * la de los numeros mas grandes que el pivote y otra con los menores.
			 *
			 */
			quickSortBis(array, inicio, posicionesPivote[0]);
			quickSortBis(array, posicionesPivote[1]+1, fin);
		}
	}

	public static void pivotarBis(int[] array, int inicio, int fin, int[] posicionesPivote) {
		/**
		 * Creamos una variable pivot, que sera una variable que utilizaremos como pivote.
Esta sera el inicio de la lista
		 */
		int pivot = array[inicio];

		/**
		 * Creamos bucle for:
		 *			donde j es el inicio mas 1, asi no coincide con el pivot
		 *			j tiene que ser siempre mas pequeño o igual que el fin
		 *			j avanza de uno en uno, para poder comparar cada posicion de la lista
		 */
		for(int j = inicio+1; j<=fin; j++) {
			/**
			 * Cuando encontramos un numero mas pequeño que el pivote, entramos al
bucle
			 */
			if(array[j] < pivot) {
				posicionesPivote[0]++;
				/**
				 * Si la varible aumentada en uno (pivote), es diferente a la j,
procedemos a hacer un intercambio
```

```
                                        * de posiciones en la lista, moviendo el elemento mayor antes del
pivote, y los menores despues
                                        * de este.
                                        * De esta manera conseguimos nuestra lista no creciente.
                                        */
                                       if(posicionesPivote[0] != j) {
                                                /**
                                                 * Creamos una variable para almacenar la posicion pivote
                                                 * Igualamos el valor de esa posicion con el valor de la
posicion j
                                                 * Cambios el valor de la posicion j, igualandolo a la variable
que contiene pivote almancenada
                                                 */
                                                int inter1 = array[posicionesPivote[0]];
                                                array[posicionesPivote[0]] = array[j];
                                                array[j] = inter1;
                                       }
                              }

                              /**
                               * Cuando encontramos un numero que es igual que el pivote, entramos al
bucle
                               */
                              if(array[j] == pivot) {
                                       if(posicionesPivote[1] == inicio) {
                                                int inter2 = array[posicionesPivote[1]];
                                                array[posicionesPivote[1]] = array[j];
                                                array[j] = inter2;
                                       }
                              }
                     }
                     /**
                      * Creamos una variable para almacenar la posicion del inicio
                      * Igualamos el valor de esa posicion con el valor de la posicion pivote
                      * Cambios el valor de la posicion pivote, igualandolo a la variable que contiene inicio
almancenada
                      *
                      * Asi, movemos el pivote para que se quede en su sitio correcto
                      */
                     int inter3 = array[inicio];
                     array[inicio] = array[posicionesPivote[0]];
                     array[posicionesPivote[0]] = inter3;

                     /**
                      * Acabamos la funcion con el array posicionesPivote rellenado con los datos correctos
                      */
            }

}
```