Practica Evaluable 2

Ordenación Rápida

J. Javier Quirante Pérez

2ºD Ingeniería Informática

**Objetivo de la práctica**

En esta práctica, el alumno estudiará y desarrollará diversos algoritmos de ordenación en Java para ordenar ascendentemente un vector de valores.

a) En la clase OrdenacionRapida implementar el método ordRapidaRec que tome como parámetros un vector v y dos enteros, izq y der, y ordene los elementos de dicho vector desde izq a der (los restantes elementos deberán quedar como en el vector original). Tal como se ha definido la cabecera del método, será posible ordenar vectores con cualquier tipo de elementos, siempre que dichos elementos implementen el interfaz Comparable; es decir, siempre que definan el método compareTo. En la cabecera, T es un tipo genérico que denota el tipo de los elementos del vector. Para poder comparar elementos de tipo T el alumno tendrá que usar el método compareTo, que se invoca sobre un objeto y que toma

como parámetro otro objeto, devolviendo 0 si ambos son iguales, menor que 0 si el objeto sobre el que se invoca es menor al objeto parámetro, o mayor que 0 en caso contrario.

b) Implementar una variante del método de ordenación rápida, en la clase OrdenacionRapidaBarajada, de manera que previamente a la ordenación se mezclen aleatoriamente los elementos del vector. Comprobar ahora el funcionamiento de esta variante del método.

c) Aprovechando el método partir implementado en el primer apartado, completar en la clase BuscaElem la implementación eficiente del método kesimoRec que tome como parámetros un vector v (desordenado) y tres enteros, y encuentre el elemento que debería estar en la posición k (si este estuviera ordenado, pero sin ordenar el vector) de dicho vector desde izq a der.

d) Implementar el método de ordenar en la clase OrdenacionJava, usando alguna de las estructuras de datos predefinidas en el paquete java.util de la biblioteca estándar de Java. Comparar la eficiencia de esta implementación con los demás métodos de ordenación.

Para abordar esta práctica, se realizarán 3 implementaciones para resolver la ordenación de distinta forma. La entrega está compuesta por 3 clases, BuscaElem, OrdenacionRapidaBarajada y OrdenacionRapida. Adicionalmente se proporcionan otras clases para probar y comparar los resultados de nuestras clases, que quedan fuera de la entrega.

**OrdenacionRapida**

**Código**

public class OrdenacionRapida extends Ordenacion {

public static <T extends Comparable<? super T>> void ordenar(T[] v) {

ordRapidaRec(v, 0, v.length-1);

}

// Debe ordenar ascendentemente los primeros @n elementos del vector @v con

// una implementaci�n recursiva del metodo de ordenaci�n r�pida.

public static <T extends Comparable<? super T>> void ordRapidaRec(T[] v, int izq, int der) {

// A completar por el alumno

if(izq < der){

int s = partir(v,v[izq], izq, der);

ordRapidaRec(v, izq,s-1);

ordRapidaRec(v,s+1, der);

}

}

public static <T extends Comparable<? super T>> int partir(T[] v, T pivote, int izq, int der) {

// A completar por el alumno

int i=izq,j=der;

while(i<j){

while(v[j].compareTo(pivote)>0 && j>0) j--;

while(v[i].compareTo(pivote)<=0 && i<v.length-1) i++;

if(i<j) intercambiar(v,i,j);

}

if(pivote.compareTo(v[j])>0) intercambiar(v,izq,j);

return j;

}

// Peque�os ejemplos para pruebas iniciales.

public static void main (String[] args) {

// Un vector de enteros

Integer[] vEnt = {3,8,6,5,2,9,1,1,4};

ordenar(vEnt);

System.out.println(vectorAString(vEnt));

// Un vector de caracteres

Character[] vCar = {'d','c','v','b'};

ordenar(vCar);

System.out.println(vectorAString(vCar));

}

}

**Análisis del código**

En esta clase se implementan varios métodos, el primero de ellos es ordenar, cuya función es llamar a ordRapidaRec e inicializar la primera y la última posición del vector dado, que le entra por parámetro.

public static <T extends Comparable<? super T>> void ordenar(T[] v) {

ordRapidaRec(v, 0, v.length-1);

}

Ahora bien, el método ordRapidaRec es un método recursivo cuya función es ordenar el vector. Este método se ha implementado siguiendo el algoritmo QuickSort, donde primero se comprueba que izq < der y, después, mediante el método partir nos devuelve un valor intermedio del vector, el cual dividiremos en dos partes que serán recorridas recursivamente.

public static <T extends Comparable<? super T>> void ordRapidaRec(T[] v, int izq, int der) {

// A completar por el alumno

if(izq < der){

int s = partir(v,v[izq], izq, der);

ordRapidaRec(v, izq,s-1);

ordRapidaRec(v,s+1, der);

}

}

Para finalizar tenemos el método partir, al cual le entra por parámetro el vector, un pivote y los valores izq y der. El objetivo de este método es tener los valores menores que el pivote a la izquierda y los mayores a la derecha, utilizando el método intercambiar de la clase Ordenación.

Finalmente, hacemos la prueba con dos vectores distintos, uno con elementos de tipo Integer y otro con elementos de tipo Character, para comprobar el funcionamiento de la clase.

public static void main (String[] args) {

// Un vector de enteros

Integer[] vEnt = {3,8,6,5,2,9,1,1,4};

ordenar(vEnt);

System.out.println(vectorAString(vEnt));

// Un vector de caracteres

Character[] vCar = {'d','c','v','b'};

ordenar(vCar);

System.out.println(vectorAString(vCar));

}

**OrdenacionRapidaBarajada**

**Código**

public class OrdenacionRapidaBarajada extends OrdenacionRapida {

// Implementaci�n de QuickSort con reordenaci�n aleatoria inicial (para comparar tiempos experimentalmente)

public static <T extends Comparable<? super T>> void ordenar(T[] v) {

// A completar por el alumno

barajar(v);

ordRapidaRec(v,0,v.length-1);

}

// reordena aleatoriamente los datos de un vector

private static <T> void barajar(T[] v) {

// A completar por el alumno

for(int i = v.length - 1; i > 0; i--){

int p = aleat.nextInt(i+1);

intercambiar(v,i,p);

}

}

}

**Análisis del código**

La clase OrdenacionRapidaBarajada es una clase que hereda de OrdenacionRapida y está compuesta por el método ordenar y barajar.

El método ordenar realiza una ordenación aleatoria de los elementos del vector con el método barajar para, posteriormente, aplicar la ordenación rápida recursiva.

public static <T extends Comparable<? super T>> void ordenar(T[] v) {

// A completar por el alumno

barajar(v);

ordRapidaRec(v,0,v.length-1);

}

El método barajar está compuesto por un bucle for donde se irán intercambiando los elementos del vector aleatoriamente el elemento Random aleat de la clase ordenación.

// reordena aleatoriamente los datos de un vector

private static <T> void barajar(T[] v) {

// A completar por el alumno

for(int i = v.length - 1; i > 0; i--){

int p = aleat.nextInt(i+1);

intercambiar(v,i,p);

}

}

}

**BuscaElem**

**Código**

import java.util.Scanner;

public final class BuscaElem{

public static <T extends Comparable<? super T>> T kesimo(T v[], int k) {

return kesimoRec(v,0,v.length-1,k);

}

public static <T extends Comparable<? super T>> T kesimoRec(T v[], int izq, int der, int k) {

// A IMPLEMENTAR POR EL ALUMNO

if(izq<der){

int q = OrdenacionRapida.partir(v,v[izq],izq,der);

if(k == q){

return v[k];

}else if (k < q){

return kesimoRec(v,izq,q-1,k);

}else{

return kesimoRec(v, q+1, der, k);

}

}

return v[k];

}

/\*\*

\* @param args

\*/

public static void main(String[] args) {

int maxvector;

int i,k;

Scanner sc = new Scanner(System.in);

System.out.print("Introduce el numero de posiciones del vector: ");

maxvector=sc.nextInt();

Integer v[]=new Integer[maxvector];

System.out.print("Introduce "+maxvector+" enteros separados por espacios: ");

for (i=0;i<maxvector;i++) v[i]=sc.nextInt();

System.out.print("Introduce la posicion k deseada (de 1-"+maxvector+"): ");k=sc.nextInt();

Integer elem=kesimo(v,k-1);

System.out.print("El elemento "+k+"-esimo es: "+elem);

}

}

**Análisis del código**

En primer lugar, tenemos el método kesimo que llamará al método kesimoRec, tomando el vector pasado por parámetro, inicializando la primera y la última posición del vector y la posición “k”, donde se desea buscar.

public static <T extends Comparable<? super T>> T kesimo(T v[], int k) {

return kesimoRec(v,0,v.length-1,k);

}

A continuación, en el método kesimoRec primero comprobamos que los valores izq y der son válidos y después haremos uso del método partir de la clase OrdenacionRapida. El objetivo es hacer llamadas recursivas hasta que la posición k introducida por el usuario coincida con la posición del array ordenado para poder devolver el valor en el orden correspondiente.

public static <T extends Comparable<? super T>> T kesimoRec(T v[], int izq, int der, int k) {

// A IMPLEMENTAR POR EL ALUMNO

if(izq<der){

int q = OrdenacionRapida.partir(v,v[izq],izq,der);

if(k == q){

return v[k];

}else if (k < q){

return kesimoRec(v,izq,q-1,k);

}else{

return kesimoRec(v, q+1, der, k);

}

}

return v[k];

}

Finalmente tenemos un método main ya proporcionado para probar el funcionamiento del programa.

/\*\*

\* @param args

\*/

public static void main(String[] args) {

int maxvector;

int i,k;

Scanner sc = new Scanner(System.in);

System.out.print("Introduce el numero de posiciones del vector: ");

maxvector=sc.nextInt();

Integer v[]=new Integer[maxvector];

System.out.print("Introduce "+maxvector+" enteros separados por espacios: ");

for (i=0;i<maxvector;i++) v[i]=sc.nextInt();

System.out.print("Introduce la posicion k deseada (de 1-"+maxvector+"): ");k=sc.nextInt();

Integer elem=kesimo(v,k-1);

System.out.print("El elemento "+k+"-esimo es: "+elem);

}