PROGRAMACIÓN Y ESTRUCTURAS DE DATOS AVANZADAS

PRUEBA DE EVALUACIÓN CONTINUA 1

Andrea Garea González
48117937M
agarea10@alumno.uned.es
andreagareagonzalez@gmail.com

Contenido

Presentación y breve explicación de la prueba

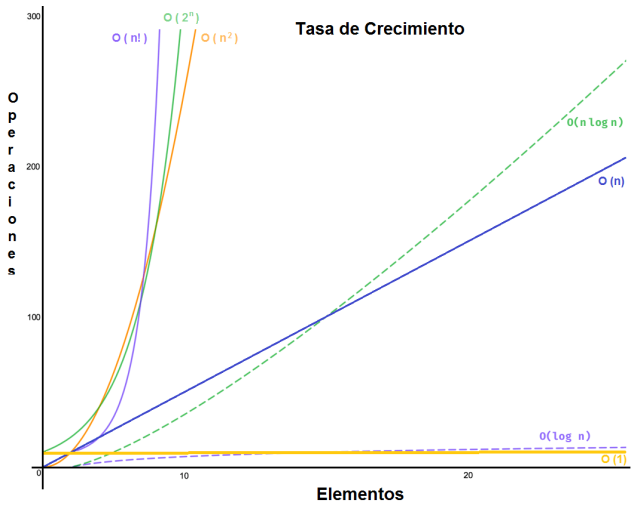
En esta práctica se ha utilizado el **algoritmo voraz** para la resolución del **problema de la mochila**.

El algoritmo voraz es una técnica de resolución en la que, en cada paso, se toma la **decisión más óptima posible** con la información disponible en ese momento para llegar a una solución óptima.

En este caso, utilizaremos el algoritmo voraz para *ordenar los objetos en relación a su peso y valor* (es decir, según el beneficio que proporcionan por unidad de peso). Después se irán añadiendo a la mochila en orden, hasta que no quepan más o no haya más objetos por añadir. Si hay espacio en la mochila para el *objeto completo*, se añade; si no, se *fracciona* y se añade una *parte proporcional* al espacio disponible.

Cuestiones teóricas sobre la prueba

Indica y razona sobre el coste temporal y espacial del algoritmo

1. 

2. $$O(g(x)) = \left\{ \begin{array}{l} f(x) : \text{existen } x_0, c > 0 \text{ tales que} \\ \forall x \geq x_0 > 0 : 0 \leq |f(x)| \leq c|g(x)| \end{array} \right\}$$

Para calcular el coste del algoritmo se ha usado la **cota superior asintótica** (O grande). La característica principal de este análisis es que se calcularán los costes en relación al peor caso.

Analizando el algoritmo, el primer factor, es la ordenación descendente de los objetos en relación a su valor. Para esta ordenación se utiliza método sort() de la clase Arrays, cuyo coste equivale a O(n log n) por utilizar una ordenación Quicksort de doble pivote.

https://docs.oracle.com/javase/7/docs/api/java/util/Arrays.html

```
public static void sort(Object[] a) {
        if (LegacyMergeSort.userRequested)
            legacyMergeSort(a);
        else
            ComparableTimSort.sort(a, 0, a.length, null, 0, 0);
    }
```

El otro factor es el bucle principal que se ejecuta mientras el peso de la mochila sea menor que el peso máximo permitido. En cada iteración del bucle, se recorren todos los objetos de la mochila y se toma una decisión sobre si añadir el objeto completo o una fracción del mismo. Esta operación tiene un coste temporal de O(n).

Teniendo esto en cuenta, se determina que **el coste del algoritmo implementado en esta práctica es O(n log n).**

El **coste espacial** del algoritmo voraz, o sea, el espacio de memoria ocupado, *dependerá de la mochila y del número de objetos que contenga*. Se utilizará:

- El objeto tipo Mochila
- X: array de floats

Explica en qué consiste la función de selección y demuestra su optimalidad

La función de selección se basa en la discriminación de ciertos candidatos para encontrar el más óptimo.

Para este caso, nuestra función de selección es la que dicta qué objetos entran o no en la mochila y de qué forma.

La condición para esta selección es que los objetos de la mochila inicial han de estar ordenados por peso/valor de objeto de mayor a menor. De esta forma, se procesarán los objetos de mayor valor en relación a su peso primero:

- Si el peso del objeto total seleccionado no supera al de la mochila, se meterá entero en la mochila final (outputBag).
- Si el peso del objeto total seleccionado supera al de la mochila, se fraccionará de tal forma que solo se seleccionará el producto del producto (y valor) del peso máximo de la mochila menos el actual entre el peso del objeto seleccionado

$$Valor\ fraccionado\ objeto = valorObjeto * [(pesoMaxMochila - pesoActualMochila)\ \div pesoObjeto]$$

- En caso de haber llegado al tope de peso de la mochila, se abandona el bucle y se "cerrará la mochila".

Explica qué otros esquemas pueden resolver el problema y razona sobre su idoneidad

No es posible resolver el problema de la mochila utilizando el algoritmo de divide y vencerás, ya que este algoritmo se basa en la división de un problema en subproblemas más pequeños de la misma naturaleza y en la resolución de estos subproblemas de forma recursiva. Una vez se tienen las soluciones a todos los subproblemas, se combinan para obtener la solución final.

Sin embargo, sí se podría utilizar el algoritmo de búsqueda de ramificación y poda. Este algoritmo consiste en explorar un problema y utilizar una función de poda para descartar las soluciones que sabemos que no son óptimas. El algoritmo comienza explorando todas las soluciones posibles y luego se ramifica en varias direcciones para explorar todas las opciones posibles en cada paso. Cada vez que se encuentra una solución mejor que la anterior, se actualiza como la solución óptima actual. En este caso, se explorarían todas combinaciones posibles de objetos que caben en la mochila y utilizar una función de poda para descartar aquellas que sabemos que no son. A medida que avanzamos en la exploración del espacio de estados, podemos ir actualizando la solución óptima actual cada vez que encontramos una solución mejor. Sin embargo, hacer esto conlleva a la creación de multitud de ramas y multitud de opciones, esto supone que este algoritmo sea exponencialmente mas costoso que el voraz y, por lo tanto, menos óptimo.

Ejemplo de ejecución con distintos tamaños del problema

NOTA: **Las ejecuciones para demostrar la optimalidad de la prueba se han generado mediante otro pequeño script a mayores. Este script está configurado para que los valores de peso y valor no sean inferiores a 0 ni superiores a 100.**

```
import java.io.*;

public class GeneratorClass {

	public static BufferedReader reader = new BufferedReader(new
InputStreamReader(System.in));

	public static void main(String[] args) throws NumberFormatException,
IOException {
		int pointer = 0, totalPeso = 0, totalValor = 0;
		BufferedWriter bw = new BufferedWriter(new
OutputStreamWriter(new FileOutputStream("autogeneratedData.txt"), "utf-
8"));
		System.out.println("Numero de objetos a generar:");
		int totalObjetos = Integer.parseInt(reader.readLine());
		System.out.println(totalObjetos);
		bw.write(totalObjetos + "\n");
		while (pointer <  totalObjetos) {
			int pesoObjeto = (int) Math.floor(Math.random()*(100-
1+1)+1);
			//System.out.println(pesoObjeto);
			int valorObjeto = (int) Math.floor(Math.random()*(100-
1+1)+1);
			bw.write(pesoObjeto + " " + valorObjeto + "\n");
			totalPeso = totalPeso + pesoObjeto;
			totalValor = totalValor + valorObjeto;
			pointer ++;
		}

		System.out.println("El peso de la mochila va a ser de: " +
totalPeso);
		System.out.println("El valor total mochila va a ser de: " +
totalValor);
		System.out.println("Sabiendo esto, indica el peso total
admitido: ");
		int pesoTotal = Integer.parseInt(reader.readLine());
		bw.write(pesoTotal + "");
		bw.close();
	}
}
```

En este apartado se describirán algunas de las pruebas realizadas para la demostración y comprobación de esta práctica.

PRUEBA 1: 10 ítems (6ms)

Entrada:

```
10
53 44
```

```
36 19
100 92
58 27
26 94
3 92
46 15
58 71
77 87
89 47
200
```

```
java -jar testPEC1preda.jar autogeneratedData.txt ficherosalida.txt
** Resultados en el archivo ficherosalida.txt
Tiempo de ejecucion: 6 ms
```

Salida:

```
**Peso de la mochila: 200.0
** Beneficio total maximizado: 377.12
==========================
** [1] | Peso: 3 | Fraccion en la mochila: 1.0 | Beneficio fraccionado: 92.0
** [2] | Peso: 26 | Fraccion en la mochila: 1.0 | Beneficio fraccionado: 94.0
** [3] | Peso: 58 | Fraccion en la mochila: 1.0 | Beneficio fraccionado: 71.0
** [4] | Peso: 77 | Fraccion en la mochila: 1.0 | Beneficio fraccionado: 87.0
** [5] | Peso: 100 | Fraccion en la mochila: 0.36 | Beneficio fraccionado: 33.120003
==========================
```

PRUEBA 2: 100 ítems (8ms)

Para estas pruebas, dejo adjuntos los resultados en el propio .rar

Entrada: ficheroEntrada100items.txt

```
java -jar testPEC1preda.jar autogeneratedData.txt ficheroSalida100items.txt
** Resultados en el archivo ficheroSalida100items.txt
Tiempo de ejecucion: 8 ms
```

Salida: ficheroSalida100items.txt

PRUEBA 3: 1000 ítems (30ms)

Entrada: ficheroEntrada1000items.txt

```
>java -jar testPEC1preda.jar autogeneratedData.txt ficheroSalida1000items.txt
** Resultados en el archivo ficheroSalida1000items.txt
Tiempo de ejecucion: 30 ms
```

Salida: ficheroSalida1000items.txt

PRUEBA 4: 10000 ítems (101ms)

Entrada: ficheroEntrada10000items.txt

```
>java -jar testPEC1preda.jar autogeneratedData.txt ficheroSalida10000items.txt
** Resultados en el archivo ficheroSalida10000items.txt
Tiempo de ejecucion: 101 ms
```

Salida: ficheroSalida10000items.txt

PRUEBA 5: No introducir ficheros de entrada y salida

Resultado: no ejecutable

```
C:\Users\andre\eclipse-workspace\PEC1_PREDA_Andrea_Garea_Gonzalez-20221025T111942Z-001\PEC1_PREDA_Andrea_
>java -jar testPEC1preda.jar

C:\Users\andre\eclipse-workspace\PEC1_PREDA_Andrea_Garea_Gonzalez-20221025T111942Z-001\PEC1_PREDA_Andrea_
```

PRUEBA 6: No introducir fichero de entrada

Resultado: Necesitamos incluir los pesos por consola

```
>java -jar testPEC1preda.jar salidaSimple.txt
** INFO: No se ha proporcionado fichero de entrada.
** INFO: Se requeriran los datos durante la ejecucion.
** Introduce numero (> 0) de objetos en la mochila:
4
** Introduce peso (> 0) del objeto 0:
21
** Introduce valor (> 0) del objeto 0:
11
** Introduce peso (> 0) del objeto 1:
20
** Introduce valor (> 0) del objeto 1:
21
** Introduce peso (> 0) del objeto 2:
26
** Introduce valor (> 0) del objeto 2:
9
** Introduce peso (> 0) del objeto 3:
20
** Introduce valor (> 0) del objeto 3:
98
** Introduce el peso total (> 0) de la mochila:
211
** Resultados en el archivo indicado.
```

PRUEBA 7: Incluir traza (-t) y ayuda (-h) sin fichero de salida

Entrada (enunciado de la práctica):

```
3
18 25
15 24
10 15
20
```

Resultado: muestra la ayuda, lo que va haciendo y el resultado por pantalla.

```
>java -jar testPEC1preda.jar prueba1.dat -t -h

// MAIN //
SINTAXIS: mochila-voraz [fichero entrada] [fichero salida] [-t][-h]
-t              Traza el algoritmo
-h              Muestra esta ayuda
[fichero entrada]               Nombre del fichero de entrada
[fichero salida]                Nombre del fichero de salida
        // Leyendo datos de entrada...
        // Preparando fichero de salida...
** INFO: Fichero de salida no proporcionado, el resultado se proporcionara por pantalla.
        // Se obtiene numero de objetos: 3
        // Objeto 0 guardado en la mochila original:
                // Peso: 18
                // Valor: 25
        // Objeto 1 guardado en la mochila original:
                // Peso: 15
                // Valor: 24
        // Objeto 2 guardado en la mochila original:
                // Peso: 10
                // Valor: 15
        // Se obtiene el peso total: 20

// ALGORITMO VORAZ //
        // Se ordenan los objetos segun su relacion peso/valor
        // Objeto 0 con peso 15 y valor 24.0
                // No supera el total de la mochila, se mete entero con su valor entero.
        // Objeto 1 con peso 10 y valor 15.0
                // Al superar el peso, se fracciona su peso en un 50.0% quedando un nuevo valor de 7.5
        // Objeto 2 con peso 18 y valor 25.0
** Peso de la mochila: 20.0
** Beneficio total maximizado: 31.5
=========================
** [1] | Peso: 15 | Fraccion en la mochila: 1.0 | Beneficio fraccionado: 24.0
** [2] | Peso: 10 | Fraccion en la mochila: 0.5 | Beneficio fraccionado: 7.5
=========================
Tiempo de ejecucion: 23 ms
```

Listado del código fuente completo

Clase Main

```
package preda_pec1;

import java.io.*;
import java.util.Arrays;
import java.util.List;

public class Main {

	public static BufferedReader reader = new BufferedReader(new
InputStreamReader(System.in));

	/*
	 * Comprueba la validez del fichero de entrada
	 * @param file
	 * @return boolean
	 */
	public static Boolean checkInput(String file) throws IOException {
		File f = new File(file);
		if(!f.exists() || !f.isFile() || !f.canRead()) {
			System.out.println("** INFO: No se ha proporcionado
fichero de entrada.");
			System.out.println("** INFO: Se requeriran los datos
durante la ejecucion.");
			return false;
		} else {
			try {
				BufferedReader br = new BufferedReader(new
InputStreamReader(new FileInputStream(file), "utf-8"));
				int controllInput =
Integer.parseInt(br.readLine());
			} catch (Exception e) {
				System.out.println("** INFO: No se ha proporcionado
fichero de entrada.");
				System.out.println("** INFO: Se requeriran los
datos durante la ejecucion.");
				return false;
			}
			return true;
		}
	}

	/*
	 * Comprueba la validez del fichero de salida
	 * @param file
	 * @return boolean
	 */
	public static Boolean checkOutput(String file) {
		if(!file.contains(".txt") && !file.contains(".dat")) {
			System.out.println("** INFO: Fichero de salida no
proporcionado, el resultado se proporcionara por pantalla.");
			return false;
		}
		File f = new File(file);
		f=f.getAbsoluteFile();
```

```
            if(!f.getParentFile().exists() || !f.getParentFile().canWrite())
{
                  System.out.println("** ERROR: No se puede manipular el
fichero de salida.");
                  return false;
            }
            if(f.exists()) {
                  if(!f.isFile() || !f.canWrite()) {
                        System.out.println("** ERROR: El archivo de salida
no es un fichero o no puede sobreescribirse.");
                        return false;
                  }
            }
            return true;
      }

      /*
       * Menu de ayuda
       */
      public static void getHelp() {
            System.out.println("SINTAXIS: mochila-voraz [fichero entrada]
[fichero salida] [-t][-h] ");
            System.out.println("-t\t\tTraza el algoritmo");
            System.out.println("-h\t\tMuestra esta ayuda");
            System.out.println("[fichero entrada]\t\tNombre del fichero de
entrada");
            System.out.println("[fichero salida]\t\tNombre del fichero de
salida");
      }

      /*
       * Imprime los datos finales en el fichero parametrizado
       * @params Mochila
       * @params bw fichero sobre el que escribir
       */
      public static void interpretaData(Mochila m, BufferedWriter bw) throws
IOException {
            bw.write("**Peso de la mochila: " + m.getPesoMochila() + "\n");
            bw.write("** Beneficio total maximizado: " +
m.getBeneficioTotal() + "\n");
            bw.write("==========================" + "\n");
            for (int b = 0; b < m.objetos.length; b++) {
                  if(m.objetos[b] != null)
                        bw.write("** [" + (b+1) +"] | Peso: " +
m.getObjetos()[b].getPeso() + " | Fraccion en la mochila: " +
m.getObjetos()[b].getValorFraccMoch() + " | Beneficio fraccionado: " +
m.getObjetos()[b].getBeneficio() + "\n");
                  else
                        break;
            }
            bw.write("==========================");
      }

      /*
       * Muestra los datos finales por consola
       * @params Mochila
       */
      public static void interpretaData(Mochila m) {
```

```
            System.out.println("** Peso de la mochila: " +
m.getPesoMochila());
            System.out.println("** Beneficio total maximizado: " +
m.getBeneficioTotal());
            System.out.println("=========================");
            for (int b = 0; b < m.objetos.length; b++) {
                  if(m.objetos[b] != null)
                        System.out.println("** [" + (b+1) +"] | Peso: " +
m.getObjetos()[b].getPeso() + " | Fraccion en la mochila: " +
m.getObjetos()[b].getValorFraccMoch() + " | Beneficio fraccionado: " +
m.getObjetos()[b].getBeneficio());
                  else
                        break;
            }
            System.out.println("=========================");
      }

      /*
       * Metodo principal que ejecuta el programa
       * @param args argumentos de entrada que proporciona el usuario
       */
      public static void main(String[] args) throws IOException,
NumberFormatException {
            long startTime = System.currentTimeMillis();
            //////
            if(args.length != 0) {
                  List<String> transformedData = Arrays.asList(args);
                  Boolean needHelp = transformedData.contains("-h"); //
Muestra la ayuda
                  Boolean isTraza = transformedData.contains("-t"); //
Muestra la traza de lo que el script hace
                  BufferedWriter bw = null;
                  BufferedReader br = null;
                  //////

                  if(isTraza) {
                        System.out.println("");
                        System.out.println("// MAIN //");
                  }

                  // MARK: GET HELP
                  if(needHelp)
                        getHelp();

                  // MARK: PREPARA FICHEROS O SALIDAS
                  if(isTraza)
                        System.out.println("      // Leyendo datos de
entrada...");
                  Boolean hasInputFile = checkInput(args[0]);
                  if(hasInputFile)
                        br = new BufferedReader(new InputStreamReader(new
FileInputStream(args[0]), "utf-8")); // Lectura del fichero de entrada

                  if(isTraza)
                        System.out.println("      // Preparando fichero de
salida...");
                  if((hasInputFile && args.length > 1 &&
checkOutput(args[1])) || (!hasInputFile && checkOutput(args[0]))) { // Si ha
```

```
aportado fichero de salida, el resultado se imprimira en el mismo en caso de
ser valido
                        if(hasInputFile)
                                bw = new BufferedWriter(new
OutputStreamWriter(new FileOutputStream(args[1]), "utf-8"));
                        else
                                bw = new BufferedWriter(new
OutputStreamWriter(new FileOutputStream(args[0]), "utf-8"));
                }

                // MARK: OBTIENE OBJETOS
                int numObjetos = 1, pointer = 0;
                if(hasInputFile) {
                        numObjetos = Integer.parseInt(br.readLine()); // La
primera linea siempre sera el total de objetos a leer
                } else {
                        do {
                                System.out.println("** Introduce numero (> 0)
de objetos en la mochila:");
                                numObjetos =
Integer.parseInt(reader.readLine());
                        } while (numObjetos < 0);
                }
                Mochila inputBag = new Mochila(numObjetos);
                if(isTraza)
                        System.out.println("      // Se obtiene numero de
objetos: " + numObjetos);


                // MARK: OBTIENE PESOS Y VALOR DE LOS OBJETOS
                while (pointer <  numObjetos) { // Se leen los datos de
todos los objetos
                        int peso = 0, valor = 0;
                        if(hasInputFile) {
                                String linea = br.readLine(); // lectura
                                peso = Integer.parseInt(linea.split(" ")[0]);
// primer dato = peso del objeto
                                valor = Integer.parseInt(linea.split("
")[1]); // segundo dato = valor del objeto
                        } else {
                                do {
                                        System.out.println("** Introduce peso
(> 0) del objeto " + pointer + ":");
                                        peso =
Integer.parseInt(reader.readLine());
                                } while (peso < 0);
                                do {
                                        System.out.println("** Introduce valor
(> 0) del objeto " + pointer + ":");
                                        valor =
Integer.parseInt(reader.readLine());
                                } while (valor < 0);
                        }

                        inputBag.newObjeto(new Objeto(peso,valor,1));

                        if(isTraza) {
                                System.out.println("      // Objeto "  +
pointer + " guardado en la mochila original:");
```

```
                                    System.out.println("            // Peso: "+
peso);
                                    System.out.println("            // Valor: "+
valor);
                              }
                              pointer ++;
                        }

                        // MARK: OBTIENE PESO MAXIMO MOCHILA
                        int pesoMaxMochila = 1;
                        if(hasInputFile) {
                              pesoMaxMochila = Integer.parseInt(br.readLine());
// la ultima linea sera el peso total de la mochila
                        } else {
                              do {
                                    System.out.println("** Introduce el peso
total (> 0) de la mochila:");
                                    pesoMaxMochila =
Integer.parseInt(reader.readLine());
                              } while (pesoMaxMochila < 0);
                        }
                        if(isTraza)
                              System.out.println("      // Se obtiene el peso
total: " + pesoMaxMochila);

                        // MARK: CALCULO
                        if(bw != null) { // en caso de haberse proporcionado
fichero de salida se mostraran los datos a traves de el

      interpretaData(preda_pec1.AlgoritmoVoraz.algoritmo(pesoMaxMochila,
isTraza, inputBag), bw);
                              System.out.println("** Resultados en el archivo
indicado.");
                              bw.close();
                        }else { // Sino se escribe por consola

      interpretaData(preda_pec1.AlgoritmoVoraz.algoritmo(pesoMaxMochila,
isTraza, inputBag));
                        }
                        if(br!=null)
                              br.close();
                        long endTime = System.currentTimeMillis() - startTime;
                        System.out.println("Tiempo de ejecucion: " + endTime + "
ms");
                  }
            }
      }
```

Clase Mochila

```
package preda_pec1;

public class Mochila{

	float pesoMochila;
	Objeto[] objetos;
	float beneficioTotal;

	int peso;
	int beneficio;
	float fraccion;

	// CONSTRUCTOR
	public Mochila(int totalObjetos) {
		super();
		this.pesoMochila = 0;
		this.objetos = new Objeto[totalObjetos]; // Contiene objetos en
forma de array
		this.beneficioTotal = 0;
	}

	// GETTERS y SETTERS necesarios para el propio objeto mochila
	public float getPesoMochila() {
		return pesoMochila;
	}
	public void setPesoMochila(float pesoMochila) {
		this.pesoMochila = pesoMochila;
	}
	public Objeto[] getObjetos() {
		return objetos;
	}
	public float getBeneficioTotal() {
		return beneficioTotal;
	}
	public void setBeneficioTotal(float beneficioTotal) {
		this.beneficioTotal = beneficioTotal;
	}

	// SETTERS necesarios para el objeto alojado en la mochila
	public void setPeso(int peso) {
		this.peso = peso;
	}
	public void setFraccion(float fraccion) {
		this.fraccion = fraccion;
	}

	/*
	 * Creacion de un nuevo objeto tipo Objeto que se guardara en el
objeto Mochila
	 * @params Objeto
	 */
	public void newObjeto(Objeto o) {
		for(int i = 0; i < this.objetos.length ; i ++) {
			if(this.objetos[i] == null) { // Escribe en una posicion
no ocupada
				this.objetos[i] = o; // Se guarda el objeto
parametrizado
```

```
                        this.beneficioTotal += o.getBeneficio(); // Añade
valor a la mochila
                        this.peso += o.getPeso(); // Establece el peso del
objeto
                        this.pesoMochila += o.getPeso() *
o.getValorFraccMoch(); // Añade peso al objeto Mochila (peso del objeto *
valor fraccionado del objeto)
                        break;
                    }
                }
            }

        /*public static void quicksort(Objeto obj[], int izq, int der) {
                Objeto pivote=obj[izq]; // se obtiene el pivote
                int i=izq;          // izquierda a derecha
                int j=der;          // derecha a izquierda
                Objeto aux;

                while(i < j){                           // mientras no se
crucen las búsquedas
                    while(obj[i].getBeneficio() <= pivote.getBeneficio() && i <
j) i++; // busca elemento mayor que pivote
                    while(obj[j].getBeneficio() > pivote.getBeneficio()) j--;
// busca elemento menor que pivote
                    if (i < j) {
                        aux= obj[i];                    // los intercambia
                        obj[i]=obj[j];
                        obj[j]=aux;
                    }
                  }

                  obj[izq]=obj[j];
                  obj[j]=pivote;      // los menores a la izquierda / mayores a
la derecha

                  if(izq < j-1)
                     quicksort(obj,izq,j-1);
                  if(j+1 < der)
                     quicksort(obj,j+1,der);

              }*/
}
```

Clase AlgoritmoVoraz

```
package preda_pec1;

import java.util.*;

public class AlgoritmoVoraz {

	/*
	 * Algoritmo voraz
	 * @params array de pesos obtenidos del fichero de entrada
	 * @params array de valores obtenidos del fichero de entrada
	 * @params M valor total de la mochila
	 * @return Mochila
	 */
	public static Mochila algoritmo(int _pesoMaxMochila, boolean _isTraza,
Mochila _inputBag) {
		if(_isTraza) {
			System.out.println("");
			System.out.println("// ALGORITMO VORAZ //");
		}

		//Mochila mochOriginal = new Mochila(p.length);
		Mochila outputBag = new Mochila(_inputBag.objetos.length);
		float[] x = new float[_inputBag.objetos.length];
		float peso = 0;

		// Documentacion utilizada:
https://docs.oracle.com/javase/7/docs/api/java/util/Arrays.html
		if(_isTraza)
			System.out.println("	// Se ordenan los objetos segun
su relacion peso/valor");
		Arrays.sort(_inputBag.objetos); // Metodo de ordenacion segun
los valores del objeto O(n log n)

		while (peso < _pesoMaxMochila) {
			for (int i = 0; i < _inputBag.objetos.length; i++) {
				if(_isTraza)
					System.out.println("	// Objeto " + i + "
con peso " + _inputBag.objetos[i].getPeso() + " y valor " +
_inputBag.objetos[i].getBeneficio());
				if(peso + _inputBag.objetos[i].getPeso() <=
_pesoMaxMochila) {
					x[i] = 1;
					peso = (float)peso +
_inputBag.objetos[i].getPeso();
					outputBag.newObjeto(_inputBag.objetos[i]);
					if(_isTraza)
						System.out.println("		// No
supera el total de la mochila, se mete entero con su valor entero.");
				} else if((float)(_pesoMaxMochila -
peso)/_inputBag.objetos[i].getPeso() != 0){
					x[i] = (float)(_pesoMaxMochila -
peso)/_inputBag.objetos[i].getPeso();
					peso = _pesoMaxMochila;
					float nuevoBeneficio =
_inputBag.objetos[i].getBeneficio()*x[i];
					outputBag.newObjeto(new
Objeto(_inputBag.objetos[i].getPeso(), nuevoBeneficio, x[i]));
					if(_isTraza)
```

```
                                        System.out.println("            // Al
superar el peso, se fracciona su peso en un "+ x[i]*100 +"% quedando un nuevo
valor de " + nuevoBeneficio);
                            } else {
                                  break;
                            }
                      }

              }
              return outputBag;
       }
}
```

Clase Objeto

```java
package preda_pec1;

// Documentacion utilizada:
https://docs.oracle.com/javase/8/docs/api/java/util/Comparator.html
public class Objeto implements Comparable<Objeto>{

	int peso;
	float beneficio;
	float valorFracc;
	float valorFraccMoch;

	// CONSTRUCTOR
	public Objeto(int peso, float beneficio, float valorFraccMoch) {
		super();
		this.peso = peso;
		this.beneficio = beneficio;
		this.valorFracc = (float)peso/(float)beneficio;
		this.valorFraccMoch = valorFraccMoch;
	}

	// GETTERS y SETTERS utilizados
	public int getPeso() { return peso; }
	public float getValorFracc() { return valorFracc; }
	public float getBeneficio() { return beneficio; }
	public float getValorFraccMoch() { return valorFraccMoch; }

	/*
	 * Compara objetos segun la fraccion
	 * @return 0 si el objeto parametrizado tiene el mismo valor que el
actual
	 * @return 1 si el objeto parametrizado tiene un valor inferior al
actual
	 * @return -1 si el objeto parametrizado tiene un valor superior al
actual
	 */
	@Override
	public int compareTo(Objeto o) {
		if(o.valorFracc == this.valorFracc) {
			return 0;
		} else if (o.valorFracc < this.valorFracc) {
			return 1;
		} else {
			return -1;
		}
	}
}
```