Algorithm 1

The company Real Time System (RTS) intends to design and implement an algorithm for managing data from an environmental sensor.
The data produced by the sensor consists of pairs <info, key>;
Info type: character;
Key Type: An integer in the range [1,100].
The production of pairs by the sensor can be 'simulated' using the pseudorandom generators of Java, (java.util.Random); as a seed use your own serial number to generate both the integer values uniformly distributed in the interval [1,100], and the values of the variable info).
Initially the sensor produces **K** pairs which are conveniently stored in a data structure S.
The value of **K** is odd, is in the range [11, 27] and is chosen at random.
The operations envisaged on S:

a) select the pair <info, key>, having the **median** value between the K as the key value keys present in the S.

b) after having memorized the initial **K** pairs, ask the sensor for a new pair <info, key>. If the **key** value of the generated pair is present in S, replace the pair present in S, with that generated by the sensor. Otherwise, the generated pair is discarded and it continues with another generation.

The production of pairs by the sensor can be 'simulated' using the pseudorandom generators of Java, (as the initial seed use the value 570049).
RST requires that:

1) the structure **S** is a **binary tree;**
2) the maximum number of accesses to the elements of the binary tree **S** to identify the pair referred to in point a), is equal to 1;
3) the maximum number of accesses to the nodes of S, necessary as required in point b), is not greater than |_log (**K**)_|.

Briefly discuss the proposed solution considering the (satisfiable?) Requirements defined above (points " 2" and "3").

To verify the correctness of the proposed implementation, it is required to carry out some tests that foresee the modification of the state of the **S** structure. In particular, the Java program must:

1) Implement an appropriate algorithm for visiting the tree **S** that allows to print, on the screen, the pairs by increasing value of the keys. Also print the value of **K** and the value of the initial seed;

2) Print on the screen the pair <info, key>, having as the key value the **median** value between the **K** keys present in the structure S.

Assuming to perform **K** generations of new elements <info, key> as described in point b), print the **K** pairs generated and, similarly to what is required in point i), print on the screen the pairs for increasing value of the keys.

Algorithm 2

The company Area51 must manage a set **D** composed of N elements, where the single element is made up of a pair <selector, info>:

selector: integer in [1, K], K>1;
info: character in A, A = {a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u , v, w, x, y, z}.

There are no two or more identical pairs. Furthermore, the company reports that the value of **K** will never be higher than 400.

The operations envisaged on D:

a) given a pair <selector, info> as input, check that it is present in D: if the pair is present, print the message 'element present' on the screen, if the pair is not present, print the ' element not present '.

b) given a specific value of the info variable as input, search and print the list of selectors of the pairs having the same value of the variable info given in input. Obviously the search could produce an empty list, in this case print the 'empty list' message on the screen.

c) given in input a specific value of the selector variable, search and print on the screen the list of selectors of the pairs having the same value of the selector variable given in input. Obviously the search could produce an empty list, in this case print the 'empty list' message on the screen.

Design and implement in Java **D** so that:

1. the number of accesses required to search for an element as described in point a), or not greater than 1+ number of pairs having the value of the same variable as the input pair as the value of the selector variable;
2. the number of accesses necessary for the search, and possible printing of the elements, as described in point b), is no more than 1+ number of pairs having the value of the info variable as the value given in input;
3. the number of accesses necessary for the search, and possible printing, as described in point c), must not exceed 1+ number of pairs having the value of the selector variable as the value given in input.

Discuss the proposed solution considering the specifications (satisfactory requirements?) Referred to in points "1", "2" and "3".

To test the developed program, read the **N**, **K** and pairs values from an input file as shown in the example below:

```
15 50
15 c
18 d
30 c
15 d
27 a
27 b
27 c
30 a
30 b
15 f
30 g
27 g
27 d
30 f
```

30 w

For the operation described in a), check for the presence of the following pairs:
<27, c>, <30, c>, <44, c>.

For the operation described in b), use the characters a, d, s, f, u for the verification test;

For the operation described in c), use the values 27, 30, 4 for the verification test.
Print the results of the checks carried out on the screen.

Algorithm 3

A ferry is capable of carrying any number of vehicles as long as the combined weight does not exceed *W.*
kg. At boarding there are *n* >= 1 vehicles with weights P [1..n]. The vehicles must be embarked in the order in which they are present in the array (first the vehicle with weight P [1], then the one with weight P [2] and so on), as long as the weight of the embarked vehicles remains less than or equal to W. When it is no longer possible to load other vehicles, the ferry makes a journey to bring the load to its destination, returning empty. All weights are positive real numbers.

- Write an efficient algorithm ***(using the Greedy technique)*** which, given the capacity *W* of the ferry and the weights P [1..n] of the vehicles, returns the number of trips (considering only the outward journey) that it is necessary to carry out to transport all vehicles. For example, if *W* = 1000 and *P* = [100, 300, 150, 400, 100, 400, 300, 400, 350, 500], the algorithm must return 4 as 4 trips are required: the first to transport the first 4 vehicles (total weight 100 + 300 + 150 + 400 = 950), the second to transport the next 3 vehicles (total weight 100 + 400 + 300 = 800), the third to transport the next 2 vehicles (total weight 400 + 350 = 750) and finally the last trip for the remaining vehicle.

- Write the computational cost of the implemented algorithm at the beginning of the Java program / source file *Algorithm3.java.*

The program accepts on the command line the **name of an input file,** having the following structure:

1000	*ferry capacity (W)*
10	*number of vehicles (n) vehicle*
100	*weight 1*
300	
150	
400	
100	
400	
300	
400	
350	
500	*vehicle weight n*

The program, running on the previous input, should display the following output:

Number of trips: 4
Total load per trip number 1: 950
Total load per trip number 2: 800
Total load per trip number 3: 750
Total load per trip num. 4: 500
Number of iterations of the algorithm: 10
Computational cost of the algorithm: O (...)

In this case 4 trips are made, with the total weight for each trip: 950, 800, 750 and 500. The program must also display the total number of iterations of the implemented algorithm (10 iterations for this input example) and an estimate the computational cost of the algorithm.

Algorithm 4

We have *n* >= 1 objects whose weights (in kg) are positive integers, less than or equal to 23kg. We also have an unlimited supply of suitcases, where each is able to carry a maximum weight ***P*** equal to 23kg, in which we want to arrange the objects so as not to exceed the maximum weight. The objects can be placed in the suitcase without respecting any pre-established order, as long as the total sum of the weights of the objects in the same suitcase does not exceed 23kg.
Implement an algorithm to allocate items in suitcases, and minimizing the total number of suitcases used.
Since there are no efficient algorithms to solve this problem, we ask to implement the following heuristics:

- You fill each suitcase in turn, using the objects still to be placed in the suitcases, in order to maximize the space occupied without exceeding the maximum weight *P* of the suitcase;
- for this the use of an approach based on ***dynamic programming is required.***
- The algorithm ***ends when all the objects*** have been placed in the suitcases.

The program accepts on the command line the name of an input file, having the following structure:

8		*number of objects (n)*
object1	14	*name and weight of the item 1*
object2	2	
object3	13	
object4	12	
object5	7	
object6	8	
object7	6	
object8	15	*name and weight of item "n"*

Object names can be assumed to contain no spaces. The program, running on the previous input, should display the following output:

```
Suitcase: 1
object1 14
object2 2
object5 7
Residual capacity: 0

Suitcase: 2
object6 8
object8 15
Residual capacity: 0

Suitcase: 3
object3 13
object7 6
Residual capacity: 4

Suitcase: 4
object4 12
Residual capacity: 11
```

In this case, four suitcases are used; for each, a *Suitcase X* header is displayed followed by the list of object names (in any order) with their weights. Finally, the line Remaining capacity: *Y*
indicates the space (in kg) still available in the suitcase. A blank line separates each block from the next.
The Java program *(Algorithm4.java)* must contain a precise description (or a block of comments) of the dynamic programming algorithm used to allocate the objects in the suitcases. In particular, indicate:

- how the subproblems are defined;
- how the solutions to these subproblems are defined;

- how the solutions are calculated in the "base" cases;
- how the solutions are calculated in the general case.

Analyze the asymptotic cost of the implemented algorithm (we refer to the complete algorithm, not just the dynamic programming algorithm used to fill each suitcase). The analysis can be saved in the source file *Algorithm4.java* or in a separate file in PDF format with the name Algorithm.pdf.

Algorithm 5

A telecommunication network of an Internet Service Provider is represented by an oriented graph *G = (V,* E) composed of *n >=2* nodes labeled as {0, 1, ... *n* - 1}, which represent the routers and *m >= 1* arcs which represent the links (cables) that connects couple of routers/nodes.
Each arc is labeled with a positive real value, which indicates the time (in seconds) it takes to cross the corresponding link. On each of the *n* routers, the packet can undergo a waiting time: before it can be transmitted in transmission on any outgoing link from the router (outgoing arcs from the corresponding node) it is necessary to wait for the other packets waiting in the queue to be served. Assuming that an ***expected double function (int i, double t) is given,*** which accepts as parameters the identifier of a node *i* (integer between 0 and *n* - 1), and the instant in time *t* in which the packet arrives to the router (to the node). The function returns a real value *>=0*, which indicates the waiting time before the packet will be emitted on one of the node's exits. Therefore, supposing that the packet arrives at node *i* at time T, it will be possible to be emitted on one of the outgoing arcs at the instant *T + wait (i, T).*

Implement a program to calculate the minimum time required to pass a packet from node 0 to node *n* - 1, if possible, assuming you are at node 0 at instant *t* = 0. Pay attention to the fact that, from the above , you can take one of the outgoing links from node 0 at the instant *waiting (0, 0).*
The program accepts a single command line parameter, which represents a file containing the telecommunication network description. An example is the following:

```
5                   number of nodes n
7                   number of arches m
0 1 132.3           origin, destination and time in milliseconds of travel arc 0
1 2 12.8
0 3 23.81
3 2 42.0
2 4 18.33
3 4 362.92
3 1 75.9            origin, destination and time in milliseconds of travel arc m-1
```

If node *n* - 1 is not reachable from node 0, the program prints "unreachable" and terminates. Otherwise, the program prints two lines: the first contains a real number which indicates the minimum time necessary to reach the destination (node *n* - 1) starting from node 0 at instant 0; the time must obviously include waiting times in the various routers. The second line contains the sequence of nodes visited, in the order in which they are traversed.

For example, considering a hypothetical implementation of the *wait ()* function which always returns the value 5.0 (constant), the algorithm applied to the previous example will print on the screen:

```
99.14
0 3 2 4
```

Note that the implemented algorithm must still work correctly with any implementation of the expected function (), naturally producing different results than those provided.