GREEDY ALGORITHMS ASSIGNMENT

GROUP MEMBERS:

- Pérez Piqueras, Víctor
- Martínez Riveiro, José Ramón

"A cargo company has been contracted to carry goods from a warehouse to another one on a lorry with a load capacity of 4 Tm. The company will be paid 10€ for each object transported, regardless of its weight. As there are goods with very different weights, you are asked to provide a greedy algorithm to choose the goods to be carried so that the money paid will be maximum. This algorithm must ask the user for the weights of the goods and after executing should show the weights of the chosen goods and the amount of the bill for the transport service."

ROUGH DESCRIPTION

A greedy algorithm is a mathematical process that looks for simple solutions to complex problems by deciding which next step will provide the most obvious benefit.

In this problem we simplify the solution by taking as candidates to work with the objects that we are going to transport.

Such algorithms are called greedy because while the optimal solution to each smaller instance will provide an immediate output (if we carry an object or not), the algorithm doesn't consider the larger problem as a whole. Once a decision has been made, it is never reconsidered.

FORMAL DESCRIPTION

- Candidates: Packets(indexes of the array of packets)
- Ordination: by increasing weight; i=0; j=0; Total=4000; W=0; N=0;
- Feasibility: if(P[i]+W <= Total){ W+=P[i]; N++; } i++;
- Selection: None
- Ending condition: if(i==length) print(packets);

EXPLANATION OF THE STEPS

- Read the number of packets and the weight of each one from keyboard: *O(n)*
- Sort the packets by increasing weight using the Quick Sort Algorithm: Quicksort is a comparison sort, meaning that it can sort items of any type for which a "less-than" relation (formally, a total order) is defined. In efficient implementations it is not a stable sort, meaning that the relative order of equal sort items is not preserved. Quicksort can operate in-place on an array, requiring small additional amounts of memory to perform the sorting. It is very similar to selection sort, except that it does not always choose worst-case partition. Mathematical analysis of quicksort shows that, on average, the algorithm takes $O(n \cdot \log n)$ comparisons to sort n items. In the worst case, it makes $O(n^2)$ comparisons, though this behaviour is rare. In this code we used an already programmed function that sorts this way, called “qsort()”.
- Print the packets ordered: *O(n)* (optional)
- While the packet index is not the maximum number of packets, perform the addition of the cumulative weight of packets plus the weight of the packet we are checking, whenever the sum is equal or lower than the maximum weight of the lorry, and add the packets that fulfil the condition to an array of carried packets: *O(n)*
- Print the total number of packets, the amount of money to pay and the weight left on the lorry: *O(1)*
- Print the packets that will be carried: *O(n)*

PSEUDO-CODE

```
compare (constant void * a, constant void * b)
        return ( *(int*)a - *(int*)b )
end_compare

Feasibility(packet,sum,total)
        if packet+sum<=total
        then
        return true
        end_if
        else
        return false
        end_else
end_ Feasibility

Main ()
        totalWeight=4000;
        weight=0;
        numberOfPackets;
        i,j=0;
        cash=0;
        Print   "Enter the number of packets:"
        input numberOfPackets
        packetWeight[numberOfPackets];
        packets[numberOfPackets];
        for i=0 to numberOfPackets
                Print  "Introduce weight for packet number " numberOfPackages
                input packetWeight[i]
        end _for
        qsort(packetWeight, numberOfPackets, sizeof(int), compare);
        Print "Total of packets ordered:"
        for i=0 to numberOfPackets
                Print "packet:" packetWeight[i]
        end _for
        j=0;
        i=0;
        while i<numberOfPackets do
                if feasibility(packetWeight[i],weight,totalWeight)
                then
                weight+=weight+=packetWeight[i]
                        cash++;
                        packets[j]=packetWeight[i];
                        j++;

        end_if
                i++;
        end_while
        Print "Number of packets selected for the service ": cash
```

```
        Print "Total amount of money to pay for the service:" cash*10 "euros"
        Print "Packets selected for the transport:"
        for i=0 to cash
        then
                Print "Packet " packets[i]
        end_for
        Print "Weight left :" totalWeight-weight
end_Main
```

COMPUTATIONAL COST

- `for i=0 to numberOfPackets` ←O(N)
- `qsort(packetWeight, numberOfPackets, sizeof(int), compare)` ←$O(N^2)$
- `for i=0 to numberOfPackets` ←O(N)
- `while i<numberOfPackets do` ←O(N)
- `for i=0 to cash` ←O(N)

-We cannot provide a Θ of the computational cost.

-For the most expensive part of the code, that is the quick-sort, we know that the average computational cost is n·log(n) → ***Ω(n·log(n))*** *but in the worst case it will raise up to n^2* → ***$O(n^2)$.***

-So the estimation of the O of the whole algorithm is ***$O(n^2)$.***
