BACKTRACKING ALGORITHMS ASSIGNMENT

GROUP MEMBERS:

- Pérez Piqueras, Víctor
- Martínez Riveiro, José Ramón

"Our cargo company has been contracted again to carry goods under the same payment conditions (10€/item) therefore the less trucks they use the better for benefits. In this scenario, an oracle has told the CEO "You can take for granted that this service can be performed by means of just 3 lorries". You are asked to provide a backtracking algorithm to choose goods to be carried in each truck in order to either find a solution or prove the oracle is wrong."

ROUGH DESCRIPTION

This algorithm will check all possible options until the solution is found, like building a searching tree of location of the packets. It'll check for each packet all the combinations allowed discarding those that doesn't fit in the truck, until the last one. By then, the algorithm will tell us whether we can carry all the goods in the 3 lorries or not.

FORMAL DESCRIPTION

- **Solution type of data:** 1 dimensional array of size n(nº of packets): SOL[i]=j means packet in position i of the array will be carried in lorry number=j

- **Exhaustivity:** Not yet chosen: 0→$\boxed{1\to 2\to 3}$→4: backtracking firing condition
- **Dead Node condition:**

  if(LorryCap[SOL(i-1)] - packet[i]<0){SOL(i)++;}

  if (SOL(i)==4{ SOL(i)=0; i=i-1;}

- **Live Node condition:** ¬ DEAD NODE { LorryCap[SOL(i-1)] -= packet[i];}
- **Solution Node condition:** if(i==n)&&(LIVE NODE)

PSEUDO-CODE

```
main()
	int i,lorry=4000;
	int numgoods;
	print "Enter the number of goods: ";
	input numgoods;
	int goods[numgoods];
	int solution[numgoods];
	int LorryFreeSpace[3]={lorry,lorry,lorry};
	for i=0 to numgoods
		print "Introduce the packet"+i+1";
		input goods[i]);
		solution[i]=0;
		fin_for
	switch(backtracking(numgoods,lorry,
solution,LorryFreeSpace,goods))
		case -1:
			print "Oracle was wrong, there is no possible
		combinations to transport all goods\n";
			break;
		fin_case-1
		case 1:
			print " Oracle was wrong, the weight of all the goods
		is greater than the lorries total capacity\n";
			break;
		fin_case1
		case 2:
			print " Oracle was wrong, there is at least 1 packet
		that weights more than 4000\n";
			break;
		fin_case2
		case 3:
			print "Oracle was right\n";
			print "Packet ordering list: ";
			for i=0 to numgoods
				print solution[i];
			end_for
			print "\n";
			break;
		end_case3
	end_switch
end_main
```

```
int backtracking(int numgoods,int lorry,int solution[], int
LorryFreeSpace[], int goods[])
      int i,sum=0;
      for i=0 to numgood
            if lorry<goods[i]
            then
                  return 2;
                  break;
            end_if

            sum+=goods[i];
      end_for
      if sum>(LorryFreeSpace[0]+LorryFreeSpace[1]+LorryFreeSpace[2])
      then
            return 1;
      end_if
      i=0;
      while i<numgoods && i>-1 do
            solution[i]++;
            if solution[i]==4
            then
                  solution[i]=0;
                  if i!=0
                  then
                        LorryFreeSpace[ solution[i-1]-1 ]+=goods[i-1];
                  end_if
                  i--;
            end_if
            else
            if LorryFreeSpace[ solution[i]-1 ]-goods[i] >=0
                  LorryFreeSpace[ solution[i]-1 ]-=goods[i];
                  i++;
            end_if
      end_while
            if i==-1
            then
                  return -1;
            end_if
      return 3;
end_backtracking
```

COMPUTATIONAL COST

- n stands for the number of goods of the input

`for i=0 to numgoods`	→ $O(n)$
`while i<numgoods && i>-1`	→ $O(n^3)$

- The best case would be having all goods already ordered in the way that the algorithm doesn't have to try more than the first option: $\Omega(n)$

- The worst case would be the solution array (n) to the power of the exhaustivity(3), an therefore, the computational cost for the big O is $O(n^3)$.
- We can’t determine the θ order of the algorithm.