Esercizio 1 (Minimo costo con minimo valore). *Progettare un algoritmo che dati in input un valore A e n oggetti con rispettivi costi $c_1, \ldots, c_n$ e rispettivi valori $v_1, \ldots, v_n$ restituisca un sottoinsieme degli oggetti che minimizzi la somma dei costi ma la cui somma dei valori sia al minimo A. La complessità richiesta è pari a $O(nA)$.*

$T[k, a]$ = sotto-insieme dei primi k oggetti di costo minimo con valore $\geq a$.

$k \in n$ $a \in A$

$T[0, a] = \text{NULL} \quad \forall a > 0 \qquad T[k, 0] = \{\}$

$$T[1, a] = \begin{cases} c_1 & \text{altrimenti} \\ \text{NULL} & \text{se } v_1 < a \end{cases}$$

Esercizio 2 (Numero di occorrenze). *Progettare un algoritmo di complessità $O(\log n)$ che dato in input un array ordinato A di n interi ed un valore x restituisce il numero di occorrenze di x in A.*

```
occ(A:array, x:int){
    i = A.length()/2
    K = NULL
    while(true){ // O(log n)
        if(A[i] < x){
            if(A[i+1]==x){
                K = i+1
                break
            }
            i = i + i/2
        }else if(A[i]>x) { i = i/2 }
        else{
            if(A[i-1]≠A[i]){
                K = i
                break
            }
        }
    }
    i = A.length()/2
    q = NULL
    while(true){ // O(log n)
        if(A[i] > x){
            if(A[i-1]==x){
                q = i-1
                break
            }
            i = i/2
        }else if(A[i]<x) { i = i + i/2 }
        else{
            if(A[i+1]≠A[i]){
                q = i
                break
            }
        }
    }
    return  q-K+1
}
```

Prendo l'indice in cui compare x più a destra e lo sottraggo all'indice in cui compare x più a sinistra.

Esempio:

$x = 4$

$A = [1\ 2\ 3\ 3\ 4\ 4\ 4\ 4\ 5\ 6\ 7]$

$A[4]$ $\quad$ $A[7] \Rightarrow 7 - 4 + 1 = 4$

occorrenze di x

Esercizio 3 (Partizioni di n). *Dato un numero intero positivo n, definiamo come partizioni di tutte le sequenze di elementi inferiori o uguali ad n la cui somma è esattamente n. In particolare, distinguiamo tra partizioni con ordine e senza ordine. Ad esempio le partizioni con ordine di $n = 4$ sono:*

$$1,1,1,1 \quad 2,1,1 \quad 1,2,1 \quad 1,1,2 \quad 2,2 \quad 3,1 \quad 1,3 \quad 4$$

mentre quelle senza ordine di $n = 4$ sono:

$$1,1,1,1 \quad 2,1,1 \quad 2,2 \quad 3,1 \quad 4$$

In generale, indichiamo con $P_o(n)$ il numero di partizioni di n con ordine e con $P(n)$ il numero di partizioni di n senza ordine.

1. *Progettare un algoritmo di complessità $O(n^2)$ che dato in input il numero n restituisca $P_o(n)$*
2. *Utilizzare la soluzione del punto precedente per dimostrare che se $n \geq 1$ allora $P_o(n) = 2^{n-1}$*
3. *Progettare un algoritmo di complessità $O(n^3)$ che dato in input il numero n restituisca $P(n)$*

```
Part(n:intero){   // la cardinalita' dell'output sara' Po(n)
    S:Set
    S.add({1,n-1})
    S.add({n-1,1})
    if(n==2){return S}
    for each p∈Part(n-1){
        S.add({1}∪p)
        S.add(p∪{1})
    }
    return S
}
```