

**Esercizio 1** (Minimo costo con minimo valore). Progettare un algoritmo che dati in input un valore  $A$  e  $n$  oggetti con rispettivi costi  $c_1, \dots, c_n$  e rispettivi valori  $v_1, \dots, v_n$  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  $\geq a$ .

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

$T[1, a] = \begin{cases} o_1 & \text{altrimenti} \\ \text{NULL} & \text{se } v_1 < a \end{cases}$

```

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
}

```

Ejemplo:

$x = 4$   
 $A = [1 \ 2 \ 3 \ 3 \ 4 \ 4 \ 4 \ 5 \ 6 \ 7]$   
 $A[4] \quad A[8] \Rightarrow 7 - 4 + 1 = 4$   
 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   | 2, 1, 1 | 1, 2, 1 | 1, 1, 2 | 2, 2 | 3, 1 | 1, 3 | 4 |
| mentre quelle senza ordine di $n = 4$ sono:  |         |         |         |      |      |      |   |
| 1, 1, 1, 1   | 2, 1, 1 | 2, 2    | 3, 1    | 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 sarà  $P_o(n)$

S:Set

S.add( {1, n-1} )

S.add( {n-1, 1} )

if(n==2){return S}

for each  $p \in \text{Part}(n-1)$ {

    S.add( {1}  $\cup$  p )

    S.add( p  $\cup$  {1} )

}

return S

}