Esercizio 1 (Insieme indipendente massimale). *Dato un grafo non diretto, un sottoinsieme dei suoi nodi è detto "indipendente" se non contiene vertici adiacenti. Progettare un algoritmo che dato in input un albero T restituisce un suo sottoinsieme indipendente di dimensione massima. Discutere la complessità della soluzione proposta e dimostrarne la correttezza.*

Ordino e considero i nodi in base al grado

```
IIM(T: albero){
    Sol[n]={0,0,...0}
    ordina V(T) in base al livello
    For each u∈V(T){
        iF(¬∃ v∧u Sol[v]==1){
            Sol[u]=1
        }
    }
    return Sol
}
```

Esercizio 2 (Numero di alberi binari). *Progettare un algoritmo di complessità $O(n^2)$ che dato in input un intero positivo n restituisca il numero di diversi alberi binari aventi n nodi (indipendentemente dagli indici assegnati ad ogni nodo).*

Ad esempio, per $n = 1$ la risposta deve essere 1 in quanto esiste solo l'albero formato da una radice, mentre per $n = 3$ la risposta deve essere 5 in quanto i possibili alberi siano i seguenti:

Esercizio 3 (Elemento maggioritario). *Dato un array A, definiamo come elemento maggioritario di A un elemento che occorre più di $\frac{n}{2}$ volte in A. Assumendo che gli elementi di A possano essere confrontati solo tramite l'operatore $=$, dunque non tramite gli operatori $<, >, \leq, \geq$ (e di conseguenza che non possano essere ordinati), progettare un algoritmo che in $O(n \log n)$ restituisca un elemento maggioritario in A. Nel caso in cui l'elemento maggioritario non esista, ritornare $\varnothing$.*

Utilizzo un approccio divide-et-impera

```
Maj (A: array){
    n=A.length()
    if(n==1){ return A[0]}
    if(n==2){
        if(A[0]==A[1]){ return A[0]}
        else{return NULL}
    }
    a=Maj(A[0:⌊n/2⌋])
    b=Maj(A[⌈n/2⌉, n-1])
    c=0
    for(i=0,1...,n-1){
        if(A[i]==a){ c++}
    }
    if(c>n/2){return c}
    c=0
    for(i=0,1...,n-1){
        if(A[i]==b){ c++}
    }
    if(c>n/2){return b}
    return NULL
}
```