1. Tempo disponibile 120 minuti.
2. Non è possibile consultare appunti, slide, libri, persone, siti web, AI generative, ecc.
3. Scrivere in modo leggibile, su ogni foglio, nome, cognome e numero di matricola.
4. Le soluzioni agli esercizi che richiedono di progettare un algoritmo devono:
   - spiegare a parole l'algoritmo (se utile, anche con l'aiuto di esempi o disegni),
   - fornire e commentare lo pseudo-codice (indicando il significato delle variabili),
   - calcolare la complessità (con tutti i passaggi matematici necessari),
   - se l'esercizio ammette più soluzioni, a soluzioni computazionalmente più efficienti e/o concettualmente più semplici sono assegnati punteggi maggiori.
5. **Nel caso di utilizzo di tecniche differenti da quelle descritte nel materiale didattico del corso, come ad esempio tecniche suggerite da intelligenze artificiali generative tipo ChatGPT, dovete giustificare la vostra scelta commentando le differenze ed i vantaggi (utilizzando, ove applicabile, l'analisi dei costi computazionali) rispetto all'uso delle tecniche descritte a lezione.**

IMPORTANTE: Risolvere gli esercizi 1–2 e gli esercizi 3–4 su fogli separati. Infatti, al termine, dovrete consegnare gli esercizi 1–2 separatamente dagli esercizi 3–4.

1. Calcolare la complessità $T(n)$ del seguente algoritmo MYSTERY1.

Algorithm 1: MYSTERY1(INT n)

```
if n ≤ 0 then
    return 1
else
    return MYSTERY1(n − 1) + MYSTERY2(n) + n

function MYSTERY2(INT n) → INT
if n ≤ 0 then
    return 1
else
    for i = 1, ··· , n do
        x = i
    return MYSTERY2(x/2) + x^2 log x + 1
```

Soluzione. Analizziamo prima il costo di MYSTERY2. Il ciclo for ha un costo lineare rispetto ad n e al termine di tale ciclo $x = n$. Inoltre, la funzione richiama se stessa una volta su input $x/2 = n/2$ mentre le altre operazioni nella chiamata hanno costo costante. L'equazione di ricorrenza di MYSTERY2

$$T'(n) = \begin{cases} 1 & n \leq 1 \\ T'(n/2) + n & n > 1 \end{cases}$$

può essere risolta con il Master Theorem

$$\alpha = \log_2 1 = 0 < 1 = \beta \Rightarrow T'(n) = \Theta(n^\beta) = \Theta(n)$$

La funzione MYSTERY1 esegue una chiamata ricorsiva su input $n - 1$ e una chiamata alla funzione MYSTERY2 con input n. L'equazione di ricorrenza che descrive MYSTERY1 è la seguente:

$$T(n) = \begin{cases} 1 & n \leq 1 \\ T(n-1) + n & n > 1 \end{cases}$$

Tale equazione di ricorrenza può essere risolta con il metodo iterativo:

$$\begin{aligned} T(n) &= T(n-1) + n \\ &= T(n-2) + (n-1) + n \\ &= \cdots \\ &= T(n-i) + \sum_{k=0}^{i-1} (n-k) \end{aligned}$$

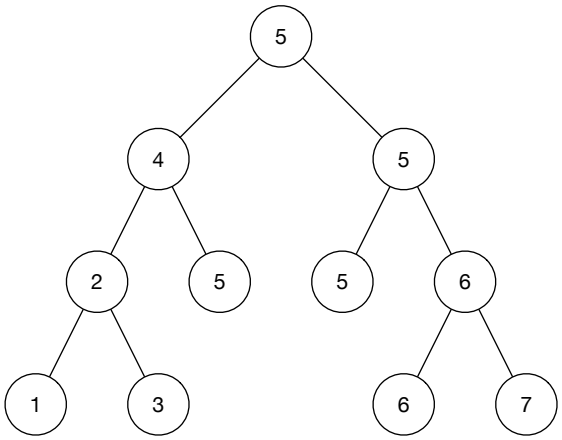
La ricorsione termina quando $n - i = 0 \Rightarrow i = n$. Sostituendo $i = n$ nell'equazione sopra otteniamo:

$$T(n) = T(0) + \sum_{k=0}^{n-1} (n-k) = 1 + \sum_{k=1}^{n} k = 1 + \frac{n(n+1)}{2} = \Theta(n^2)$$

2. Considerare le seguenti sequenze di valori chiave ottenuti visitando un Albero Binario di Ricerca:
   - pre-ordine: 5 4 2 1 3 5 5 5 6 6 7
   - post-ordine: 1 3 2 5 4 5 6 7 6 5 5

   Disegnare un Albero Binario di Ricerca compatibile con tali visite (ce ne potrebbe essere più di uno). Spiegare come è stato individuato l'albero compatibile con le visite.

   Soluzione.
   - Da entrambe le visite possiamo capire che la chiave del nodo radice è 5 (primo nodo visitato con la visita in pre-ordine e ultimo con quella in post-ordine)
   - Dalla visita in pre-ordine possiamo dedurre che il figlio sinistro della radice ha chiave 4 poiché $4 < 5$
   - Dalla vista in post-ordine, possiamo identificare il penultimo 5 come la chiave del figlio destro della radice (non c'è ambiguità poiché abbiamo già identificato 4 come chiave del figlio sinistro della radice)
   - Osservando che c'è un unico valore chiave 4, dalla visita in post-ordine possiamo separare le chiavi nei sottoalberi sinistro e destro della radice:
     - sottoalbero sinistro post-ordine: 1 3 2 5 4
     - sottoalbero destro post-ordine: 5 6 7 6 5
   - Le corrispondenti visite in pre-ordine sono:
     - sottoalbero sinistro pre-ordine: 4 2 1 3 5
     - sottoalbero destro pre-ordine: 5 5 6 6 7
   - Riapplicando ricorsivamente lo stesso ragionamento ai sottoalberi:
     - Le chiavi dei figli del figlio sinistro della radice (4) sono in ordine 2 e 5
     - Le chiavi dei figli del figlio destro della radice (5) sono in ordine 5 e 6
   - I vincoli imposti dall'Albero Binario di Ricerca ci permettono di posizionare le rimanenti chiavi senza ambiguità: $1, 3$ nel sottoalbero sinistro e $6, 7$ in quello destro
   - Abbiamo un'unica soluzione:



Cont.

3. Una ditta di catering deve trasportare su un proprio furgone il numero massimo di bottiglie di acqua. Le bottiglie sono contenute in casse; ogni cassa ha un suo peso ed un suo numero di bottiglie contenute. Non esiste una precisa relazione tra numero di bottiglie e peso visto che le casse sono fra loro diverse e costruite con materiali differenti. Il furgone ha una capacità massima, quindi si può caricare solo un insieme di casse la cui somma dei pesi risulta essere inferiore a tale capacità. Per aiutarli bisogna progettare un algoritmo che presi in input la capacità massima K del furgone e due array $p[1..n]$ e $b[1..n]$ che descrivono rispettivamente il peso ed il numero di bottiglie di ognuna delle n possibili casse ($p[i]$ e $b[i]$ sono rispettivamente il peso ed il numero di bottiglie della i-esima cassa), stampa gli indici delle casse che permettono di caricare sul furgone il numero massimo di bottiglie. Assumere che tutti i numeri in input siano interi strettamente positivi.

   Soluzione. Si può adottare programmazione dinamica considerando i problemi $P(i, j)$, con $i \in [1, n]$ e $j \in [0, K]$, dove:

   P(i,j) = numero massimo di bottiglie che si possono caricare avendo a disposizione le prime i casse ed un furgone con capacità j.

   Tali problemi possono essere risolti procedendo in modo induttivo rispetto a i considerando che:

$$P(i,j) = \begin{cases} 0 & \text{se } i = 1 \text{ e } j < p[1] \\ b[1] & \text{se } i = 1 \text{ e } j \geq p[1] \\ P(i-1,j) & \text{se } i > 1 \text{ e } j < p[i] \\ max\{P(i-1,j), b[i] + P(i-1, j-p[i])\} & \text{se } i > 1 \text{ e } j \geq p[i] \end{cases}$$

   Utilizzando questa tecnica, è possibile calcolare il massimo numero di bottiglie $P(n, K)$ che si possono caricare sul furgone. Per capire quali casse caricare, si può utilizzare, oltre alla solita matrice contenente le soluzioni ai problemi $P(i, j)$, una matrice ausiliaria booleana che per ogni coppia i, j indica se la i-esima cassa fa parte della soluzione al problema $P(i, j)$.

   Tale soluzione è riportata come Algoritmo 2; il costo computazionale di tale algoritmo è $\Theta(n \times K)$ visto che esegue solo operazioni di costo costante ed il costrutto di maggior costo corrisponde con due for annidati dove il for esterno esegue $\Theta(n)$ cicli mentre l'interno ne esegue $\Theta(K)$.

Algorithm 2: BottiglieFurgone(Natural K, Natural $p[1..n]$, Natural $b[1..n]$)

```
Natural P[1..n, 0..K]
Boolean B[1..n, 0..K]
for j = 0 to K do
    if (j < p[1]) then
        P[1, j] = 0; B[1, j] = false
    else
        P[1, j] = b[1]; B[1, j] = true
for i = 2 to n do
    for j = 0 to K do
        if (j < p[i]) OR (P[1 − 1, j] ≥ b[i] + P[1 − 1, j − p[i]]) then
            P[i, j] = P[1 − 1, j]; B[i, j] = false
        else
            P[i, j] = b[i] + P[1 − 1, j − p[i]]; B[i, j] = true
Natural cassa = n
Natural residuo = K
while cassa > 0 do
    if B[cassa, residuo] then
        print("caricare cassa " + cassa)
        residuo = residuo − p[cassa]
    cassa = cassa − 1
```

4. Progettare un algoritmo che, dato un grafo orientato aciclico $G = (V, E)$, stampa i vertici del grafo seguendo un ordinamento topologico dei vertici stessi.

Cont.

Soluzione. L'ordinamento topologico può essere calcolato su un grafo orientato aciclico effettuando una visita in profondità e poi stampando i vertici in ordine inverso di chiusura. Un modo per memorizzare i vertici da stampare prevede l'uso di una lista di vertici; al momento della chiusura, i vertici vengono inseriti in testa a tale lista tramite l'operazione *prepend*. Tale soluzione è riportata come Algoritmo 3; il costo computazionale di tale algoritmo è il medesimo della visita in profondità, ovvero $O(n + m)$, con n numero dei vertici e m numero degli archi, assumendo implementazione del grafo tramite liste di adiacenza.

Algorithm 3: StampaTopologica(Graph (V, E))

```
// Uso della lista dei vertici
List<Vertex> l
// Inizializzazione marcatura e indicazione dei parent dei vertici
for v ∈ V do
    v.mark ← false
// Esecuzione della DFS
for v ∈ V do
    if (v.mark = false) then
        DFSvisit(v)
// Stampa dei vertici estraendoli dalla lista
while l ≠ null do
    print(l.head)
    l = l.next

DFSvisit(Vertex u)
u.mark ← true
for v ∈ u.adjacents do
    if (v.mark = false) then
        DFSvisit(v)
l.prepend(u)
```

The End.