1. Tempo disponibile 120 minuti.
2. Non è possibile consultare appunti, slide, libri, persone, siti web, 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.

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) $\rightarrow$ INT

```
if n ≤ 0 then
    return 1
else
    x = 0
    for i = 1, ··· , n^2 do
        x = x+ MYSTERY2(i)
    return MYSTERY1(n/2) + x

function MYSTERY2(INT n) → INT
if n ≤ 0 then
    return 1
else
    return MYSTERY2(n/2)+MYSTERY2(n/2) + n^3
```

Soluzione. Analizziamo prima il costo di MYSTERY2. Tale funzione richiama se stessa due volta su input $n/2$ e le altre operazioni nella chiamata hanno costo costante. L'equazione di ricorrenza di MYSTERY2

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

può essere risolta con il Master Theorem

$$\alpha = \log_2 2 = 1 > 0 = \beta \Rightarrow T'(n) = \Theta(n^\alpha) = \Theta(n)$$

La complessità di MYSTERY1 richiama se stessa una volta su input $n/2$ ed il costo di una chiamata ricorsiva dipende anche dal costo del suo ciclo for interno. In tale ciclo viene richiamata la funzione MYSTERY2 n^2 volte per i che va da 1 ad n^2. Poiché il costo di MYSTERY2 è $\Theta(n)$ il suo contributo alla complessità è dato da:

$$\Theta(1) + \cdots + \Theta(n^2) = \Theta\left(\sum_{i=1}^{n^2} i\right) = \Theta\left(\frac{n^2(n^2+1)}{2}\right) = \Theta\left(n^4\right)$$

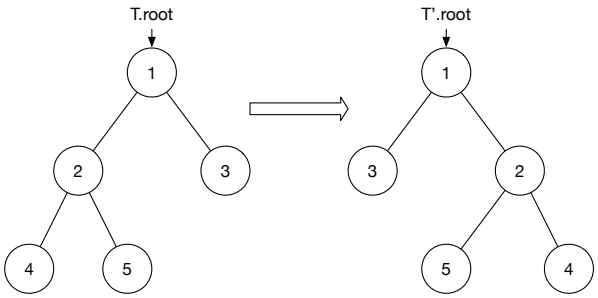
L'equazione di ricorrenza di MYSTERY1 è quindi

$$T(n) = \begin{cases} 1 & n \leq 1 \\ T(n/2) + n^4 & n > 1 \end{cases}$$

e può essere risolta con il Master Theorem

$$\alpha = \log_2 1 = 0 < 4 = \beta \Rightarrow T(n) = \Theta(n^\beta) = \Theta(n^4)$$

2. Sia T un albero binario con valori chiave interi.
   a) Scrivere lo pseudocodice di un algoritmo che, preso T in input, generi l'albero *specchiato* T'. Esempio:



   L'albero T deve rimanere inalterato dopo l'esecuzione dell'algoritmo.
   b) Analizzare la complessità della soluzione proposta

Soluzione.

a) Un algoritmo ricorsivo permette una facile risoluzione del problema. L'algoritmo viene eseguito come di seguito:

$$T'.root = \text{MIRRORTREE}(\text{T.root})$$

e ritorna la radice dell'albero specchiato

Algorithm 2: MIRRORTREE(NODE T) $\rightarrow$ NODE

```
if T == NIL then
    return NIL
else
    tmp = NODE(T.key)
    tmp.left = MIRRORTREE(T.right)
    if tmp.left ≠ NIL then
        tmp.left.parent = tmp
    tmp.right = MIRRORTREE(T.left)
    if tmp.right ≠ NIL then
        tmp.right.parent = tmp
    return tmp
```

Nel caso in cui il nodo dell'albero binario non contenga un puntatore al nodo genitore, le inizializzazioni del campo *parent* possono essere eliminate dallo pseudocodice.

b) Sia $n = |T|$ (numero di nodi in T). L'algoritmo MIRRORTREE visita ogni nodo in T esattamente una volta e il costo per duplicare un nodo è costante. Quindi il costo ottimo/medio/pessimo dell'algoritmo è $\Theta(n)$.

3. Si consideri la seguente istanza del problema dello zaino:
   - capienza dello zaino 8;
   - 4 possibili oggetti rispettivamente con peso 2, 1, 7, 6 e valore 43.2, 65.4, 98.3, 96.3.

Cont.

Mostrare come applicare la tecnica della programmazione dinamica per trovare il valore complessivo massimo che si può ottenere selezionando un insieme di oggetti, fra quelli disponibili, che hanno un peso complessivo inferiore o uguale alla capienza dello zaino. In particolare, mostrare come viene riempita la tabella utilizzata dall'algoritmo basato su programmazione dinamica discusso a lezione e presentato nelle slide del corso.

Soluzione. Sull'istanza indicata nell'esercizio, la soluzione al problema dello zaino basata su programmazione dinamica considera i sotto-problemi $P(i,j)$ (con $i \in [1,4]$ e $j \in [0,8]$) corrispondenti al calcolo del valore complessivo massimo ottenibile utilizzando i primi i oggetti ed uno zaino di capienza j. Usiamo p_i e v_i per indicare rispettivamente il peso ed il valore dell'i-esimo oggetto. Le soluzioni dei sotto-problemi $P(i,j)$ vengono calcolate considerando la seguente formula:

$$P(i,j) \quad = \quad \begin{cases} 0 & \text{se } i = 1 \text{ e } j < p_1 \\ v_1 & \text{se } i = 1 \text{ e } j \geq p_1 \\ P(i-1,j) & \text{se } i > 1 \text{ e } j < p_i \\ max\big(P(i-1,j), v_i + P(i-1, j-p_i)\big) & \text{se } i > 1 \text{ e } j \geq p_i \end{cases}$$

e salvate nella seguente tabella che viene riempita riga-per-riga:

	0	1	2	3	4	5	6	7	8
1	0	0	43.2	43.2	43.2	43.2	43.2	43.2	43.2
2	0	65.4	65.4	108.6	108.6	108.6	108.6	108.6	108.6
3	0	65.4	65.4	108.6	108.6	108.6	108.6	108.6	163.7
4	0	65.4	65.4	108.6	108.6	108.6	108.6	161.7	163.7

Come si nota dalla tabella, la soluzione al problema iniziale, coincidente con il sotto-problema $P(4,8)$, risulta essere 163.7.

4. Progettare un algoritmo che dato un grafo orientato pesato $G = (V, E, w)$ (possibilmente con pesi negativi), un vertice di partenza v, e un valore K, restituisce il numero di vertici in G la cui distanza da v è inferiore a K. Si ricorda che la distanza di un vertice w da v coincide con il costo minimo di un cammino da v a w. Si ricorda inoltre che in caso di presenza di possibili cicli di costo negativo lungo un percorso da v a w, la distanza non è definita.

   Soluzione. Vista la possibile esistenza di archi di costo negativo, il problema della ricerca dei cammini minimi da singola sorgente può essere risolta tramite l'algoritmo di Bellman-Ford. Si può usare tale algoritmo per calcolare le distanze di tutti i vertici dal vertice di partenza v ed infine calcolare quante di queste distanze risultano inferiori a K.

   L'Algoritmo 3 descrive in pseudocodice tale soluzione al problema. La parte iniziale corrisponde all'algoritmo di Bellman-Ford e ha costo computazionale $\Theta(n \times m)$ con n numero dei vertici e m numero degli archi del grafo. A seguire sono presenti il tipico controllo di possibili cicli negativi associato all'algoritmo di Bellman-Ford (costo $\Theta(m)$), e il conteggio del numero di vertici con una distanza calcolata inferiore a K (costo $\Theta(n)$). Sommando tutti questi contributi avremo che il costo computazionale della soluzione proposta risulta essere $\Theta(n \times m) + \Theta(m) + \Theta(n) = \Theta(n \times m)$.

Cont.

Algorithm 3: NumVerticiVicini(Graph G=(V, E, w), Vertex v, Number K) → Integer

```
// esecuzione dell'algoritmo di Bellman-Ford
n ← G.numNodi()
Real D[1..n]; Int pred[1..n];
for each u ∈ {1...n} do
    D[u] = ∞
D[v] ← 0
for i ← 1 to n − 1 do
    for each (u, z) ∈ E do
        if D[u] + w(u, z) < D[z] then
            D[z] = D[u] + w(u, z)
// verifica presenza cicli costo negativo
for each (u, z) ∈ E do
    if D[u] + w(u, z) < D[z] then
        error("il grafo contiene cicli negativi")
// conteggio del numero di vertici a distanza inferiore a K
Integer counter ← 0
for each u ∈ {1...n} do
    if D[u] < K then
        counter = counter + 1
return counter
```

The End.