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 $T(n)$ nel **caso pessimo** del seguente algoritmo MYSTERY1 quando
   a. A è un array ordinato e SEARCH implementa una ricerca binaria
   b. A non è ordinato e SEARCH implementa una ricerca lineare

Algorithm 1: MYSTERY1(INT $A[1, \cdots, n]$) $\rightarrow$ int

```
k = 0
for i = 1, ··· , n do
    x = MYSTERY2(i)
    if SEARCH(A, x) == true then
        k = k + 1
return k

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

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

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

può essere risolta con il Master Theorem

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

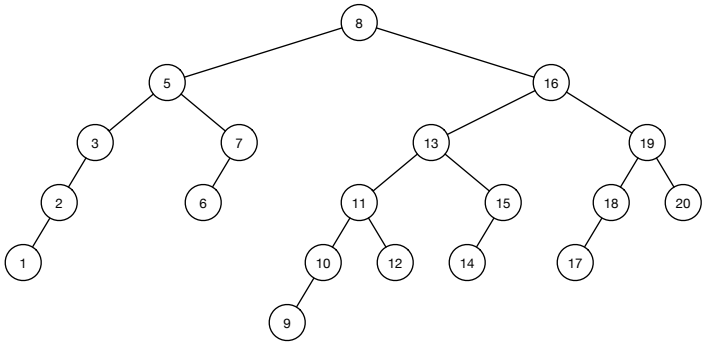
La complessità di MYSTERY1 dipende dal costo del suo ciclo for interno. Tale ciclo viene eseguito n volte per i che va da 1 ad n. In ogni iterazione esegue una chiamata a MYSTERY2 con input i ed una ricerca su array. Poiché il costo di MYSTERY2 è $\Theta(\log n)$ il suo contributo alla complessità è dato da:

$$\Theta(\log 1) + \cdots + \Theta(\log n) = \Theta\left(\sum_{i=1}^{n} \log i\right) = \Theta\left(\log \prod_{i=1}^{n} i\right) = \Theta\left(\log n!\right) = \Theta\left(n \log n\right)$$

Il costo pessimo di n ricerche su un array di dimensione n dipende dal tipo di implementazione per la funzione SEARCH

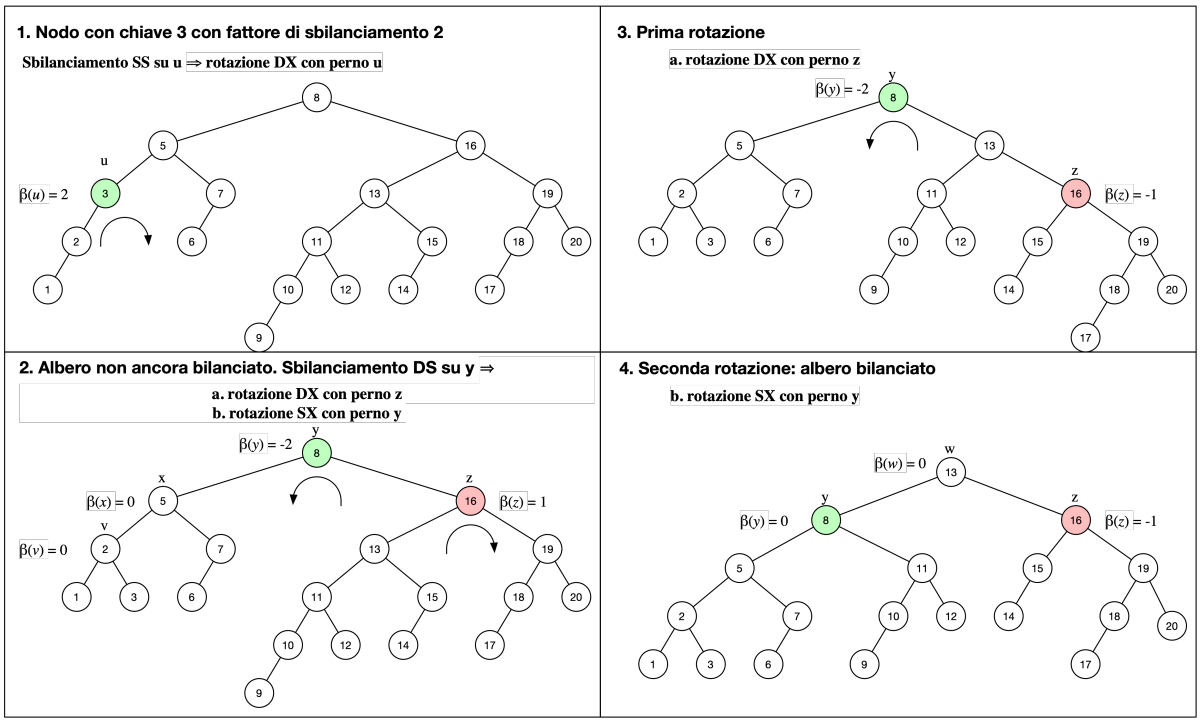
a. Se A è ordinato e SEARCH implementa una ricerca binaria, il costo pessimo di una ricerca è $\Theta(\log n)$ (quando il valore cercato non è nell'array). In tal caso, n ricerche hanno costo pessimo $n \times \Theta(\log n) = \Theta(n \log n)$. In questo caso il costo pessimo di MYSTERY1 è
$$T(n) = \Theta(n \log n) + \Theta(n \log n) = \Theta(n \log n)$$
b. Se A non è ordinato e SEARCH implementa una ricerca lineare, il costo pessimo di una ricerca è $\Theta(n)$ (quando il valore cercato non è nell'array). In tal caso, n ricerche hanno costo pessimo $n \times \Theta(n) = \Theta(n^2)$. In questo caso il costo pessimo di MYSTERY1 è
$$T(n) = \Theta(n \log n) + \Theta(n^2) = \Theta(n^2)$$

2. Considerare il seguente albero AVL, sbilanciato in seguito ad un'operazione di rimozione.



Individuare i nodi sbilanciati e ribilanciare l'albero con opportune operazioni di rotazione. Indicare chiaramente quali sono i nodi sbilanciati, il tipo di sbilanciamento e le operazioni eseguite per ribilanciare l'albero.

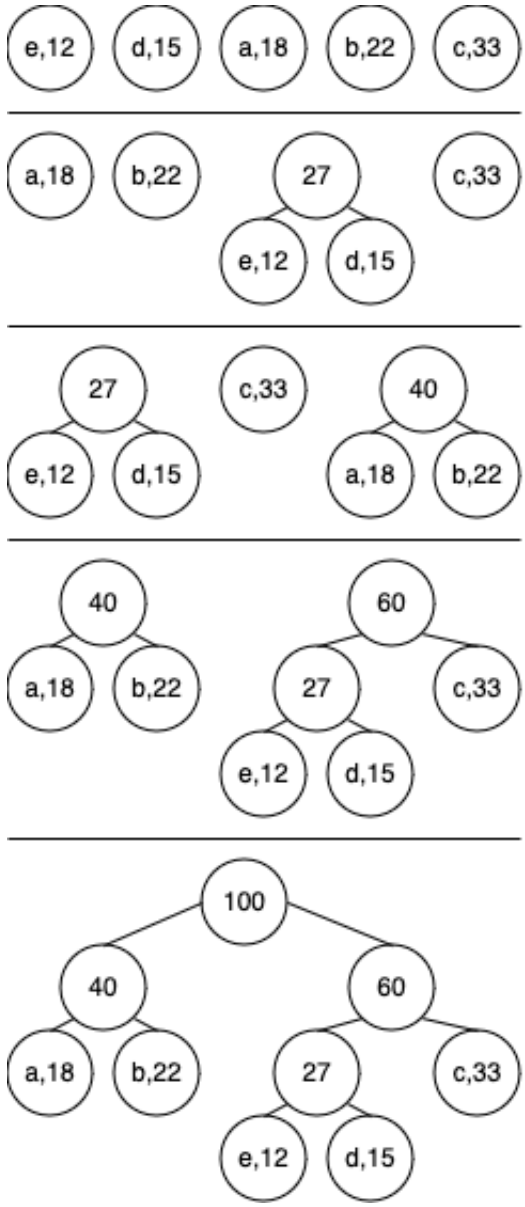
Soluzione. Il sotto albero destro della radice e il nodo radice sono bilanciati. Lo sbilanciamento riguarda il sottoalbero sinistro della radice, nello specifico il nodo con chiave 3 è il nodo più profondo con fattore di sbilanciamento -2. Sono necessarie rotazioni a cascata per ribilanciare l'albero.



Cont.

3. Si consideri l'esecuzione dell'algoritmo di Huffman a partire da un documento da codificare che contiene solo le lettere a, b, c, d, e con le seguenti frequenze: a 18%, b 22%, c 33%, d 15%, e 12%. Descrivere passo-passo la creazione dell'albero binario che rappresenta la codifica di Huffman per questo documento.

   Soluzione. Si rappresenta graficamente l'evoluzione del contenuto della coda con priorità utilizzata dall'algoritmo per memorizzare gli alberi creati ad ogni iterazione. Ogni albero ha come chiave il proprio indice di frequenza (riportato nella radice dell'albero). Le foglie dell'albero contengono anche indicazione del relativo carattere.



   Seguendo la convenzione che associa il bit 0 ad archi verso il sottoalbero di sinistra e il bit 1 ad archi verso il sottoalbero di destra, la codifica ottenuta risulta essere: $a = 00$, $b = 01$, $c = 11$, $d = 101$, $e = 100$.

4. Progettare un algoritmo che dato un grafo orientato $G = (V, E)$ restituisce un valore booleano: *true* se esiste un ciclo in G, *false* altrimenti.

   Soluzione. La ricerca di cicli in grafi orientati si può effettuare utilizzando una visita in profondità (DFS) in quanto esiste un ciclo se e solo se durante la DFS si visita un nodo che ha tra

le sue adiacenze un nodo già visitato, ma non ancora chiuso (quindi marcato "grey" secondo la terminologia usata a lezione).

Tale soluzione è riportata come Algoritmo 2. Si noti che la visita viene interrotta nel caso in cui venga trovato il ciclo, restituendo subito il valore booleano *true*. Il costo computazionale di tale algoritmo, nel caso pessimo, 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 2: VerificaCicloInGrafoOrientato(Graph (V, E)) $\rightarrow$ Bool

```
// Inizializzazione marcatura dei vertici
for v ∈ V do
    v.mark ← white
// Esecuzione della DFS
for v ∈ V do
    if (v.mark = white) then
        if DFSciclo(v) then
            // La chiamata a DFSciclo ha trovato un ciclo
            return true
// Non si sono trovati cicli
return false

// Effettua la DFS di un nodo restituendo true se si incontra un ciclo, false altrimenti
DFSciclo(Vertex u) → Bool
u.mark ← grey
for v ∈ u.adjacents do
    if v.mark = grey then
        // Si è trovato un ciclo
        return true
    if (v.mark = white) then
        if DFSciclo(v) then
            // La chiamata a DFSciclo ha trovato un ciclo
            return true
u.mark ← black
return false
```

The End.