# Complementi di Algoritmi e Strutture Dati

(III anno Laurea Triennale - a.a. 2017/18)

## Prova scritta 6 settembre 2018

**NB:** I punteggi sono indicativi.

**Esercizio 1 – Alberi AVL (Punti 6)** Consideriamo un un albero binario di ricerca (BST) che contenga 5 elementi con chiavi tutte distinte, per semplicità siano 10, 20, 30, 40, 50 tali chiavi.

1. Mostrare un ordine di inserimento che porta all'albero migliore per bilanciamento. Disegnare l'albero BST risultante. Tale sequenza è l'unica?
2. Mostrare un ordine di inserimento che porta invece all'albero peggiore per bilanciamento. Disegnare l'albero BST risultante. Tale sequenza è l'unica?
3. Considerare la sequenza del caso peggiore ed inserirla invece in un albero AVL inizialmente vuoto. Mostrare e spiegare che cosa succede ad ogni elemento inserito. Indicare anche i fattori di bilanciamento dei nodi.
4. Dall'albero risultante cancellare uno dopo l'altro gli elementi di chiave 50 e di chiave 10. Mostrare e spiegare che cosa succede ad ogni elemento cancellato.

**Esercizio 2 – Sorting (punti 6)** Si consideri il seguente array:

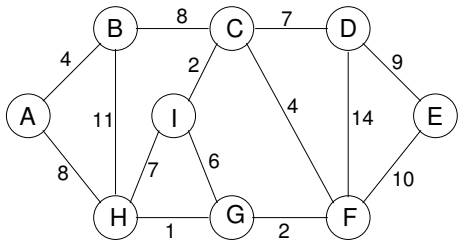
| 21 | 5 | 4 | 30 | 10 | 2 | 12 | 15 | 20 | 18 |
|---|---|---|---|---|---|---|---|---|---|

Eseguire la prima fase dell'algoritmo heapsort, cioè quella che trasforma l'array in uno heap a massimo. Si chiede di eseguirla

- PREFERIBILMENTE con la procedura *heapify*.
  Ad ogni passo disegnare tutto l'array come albero ed indicare quali sotto-parti sono già heap.
- IN ALTERNATIVA (ma allora l'esercizio vale un punto in meno) con una serie di chiamate a *insert*.
  Per ogni chiamata indicare l'elemento che viene inserito e disegnare lo heap in cui viene inserito prima e dopo l'operazione (ovviamente il "dopo" di un inserimento è il "prima" dell'inserimento seguente e non occorre ripetere il disegno).

Ricordare che deve essere uno heap *a massimo*.

**Esercizio 3 - Grafi (Punti 7)** Si esegua l'algoritmo di Kruskal sul seguente grafo pesato:




In particolare, per ogni iterazione si diano:

- l'arco estratto e il risultato (vero o falso) dell'operazione `union_by_need`
- la foresta ricoprente corrente
- la foresta union-find corrente (per brevità date solo gli alberi non singleton).

Si assuma che la `find` effettui la compressione dei cammini, e nell'unione di due alberi union-find si utilizzi la union-by-rank. Nel caso di più scelte possibili si consideri l'ordine alfabetico.

**Esercizio 4 - Tecniche algoritmiche (Punti 7)** Chiamiamo *picco* (*peak*) di una sequenza, rappresentata con un array $A[\texttt{inf}..\texttt{sup}]$ ($\texttt{inf} \leq \texttt{sup}$), un (qualunque) indice `i` tale che l'elemento a sinistra, se esiste ($\texttt{i} \neq \texttt{inf}$), è minore o uguale ($A[\texttt{i}-1] \leq A[\texttt{i}]$), e analogamente l'elemento a destra, se esiste ($\texttt{i} \neq \texttt{sup}$), è minore o uguale ($A[\texttt{i}+1] \leq A[\texttt{i}]$).

1. Esiste sempre almeno un picco?
2. Si descriva (anche solo a parole) un algoritmo brute-force che risolve il problema e se ne indichi la complessità.
3. Si descriva in pseudocodice un algoritmo divide-et-impera che risolve il problema in modo più efficiente, e se ne calcoli la complessità.
4. Se ne giustifichi la correttezza.

**Esercizio 5 - Notazioni asintotiche (Punti 6)** Usando la definizione dimostrare che:

1. se $f(n) = \Theta(g(n))$ allora anche $g(n) = \Theta(f(n))$
2. se $f(n) = O(g(n))$ e $g(n) = O(h(n))$, allora $f(n) = O(h(n))$.