

## PRIMA PARTE

1. [3 punti] Si consideri un ciclo

**for**  $i \leftarrow 1$  **to**  $n$  **do** {*corpo del ciclo*}.

Dire brevemente cosa bisogna fare per provare che alla fine del ciclo vale una certa proprietà  $\mathcal{L}$ .

2. [4 punti] Si consideri la visita in *preorder* di un albero  $T$  di  $n$  nodi eseguita invocando `preorder(T, T.root())`. Per ogni nodo  $u \in T$ , sia  $c_u$  il numero di figli di  $u$ , e sia  $t_u$  il costo della visita di  $u$ . Si ricordi che  $\sum_{u \in T} c_u = n - 1$ .
- (a) Dire quanti nodi ha l'albero della ricorsione associato a `preorder(T, T.root())`, e qual è il costo di ciascun nodo.
  - (b) Sia  $t_u \in \Theta(\log n)$  per ogni  $u \in T$ . In base al punto precedente, determinare la complessità di `preorder(T, T.root())`.
3. [4 punti] Si consideri l'array di entry:  $P = [(10, *), (8, *), (6, *), (7, *), (5, *), (2, *)]$ , e l'approccio *top-down* per la costruzione di uno *heap* a partire da  $P$ . Mostrare l'albero associato all'array alla fine di ogni iterazione dell'approccio *top-down*. (È sufficiente mostrare le chiavi delle entry.)
4. [5 punti] Si consideri un grafo  $G = (V, E)$ . Dimostrare che `DFS(G, s)` visita tutti i nodi della componente connessa di  $s$ .

## SECONDA PARTE

1. [8 punti] Sia  $T$  un albero binario di ricerca che implementa una mappa le cui entry sono coppie  $(t, i_t)$ , dove  $t$  (la chiave) rappresenta un istante di tempo in cui è avvenuto un terremoto, e  $i_t$  (il valore) rappresenta l'intensità del terremoto. Per ogni nodo  $v \in T$  esiste un campo  $v.\text{max}$  che riporta la massima intensità dei terremoti rappresentati dalle entry di  $T_v$  (sottoalbero con radice  $v$ ).
  - (a) Progettare un algoritmo ricorsivo **CheckGG** che, dati un tempo  $t$  e un valore di intensità  $i$ , restituisce **yes** se c'è stato un terremoto di intensità  $\geq i$  in un istante di tempo  $\geq t$ , altrimenti restituisce **no**. Specificare chiaramente l'input e l'output dell'algoritmo.
  - (b) Analizzare la complessità dell'algoritmo del punto precedente.
2. [8 punti] Siano  $S_1[0 \div n - 1]$  and  $S_2[0 \div n - 1]$  due array ordinati di  $n$  chiavi intere ciascuno. In ogni array le chiavi sono distinte, ma una chiave può comparire in entrambi gli array.
  - (a) Progettare un algoritmo **CountNeg**, **contenente un solo ciclo**, che determina il *numero di chiavi negative distinte* che compaiono in  $S_1$  e/o  $S_2$ . Ad esempio, se  $S_1 = [-3, -2, 0, 34]$  e  $S_2 = [-7, -2, 34, 45]$  l'algoritmo restituisce 3 (chiavi negative distinte: -7, -3, -2). Per avere punteggio pieno, l'algoritmo deve utilizzare spazio costante, oltre a  $S_1$  e  $S_2$ .
  - (b) Specificare un opportuno invariante che serva per provare la correttezza dell'algoritmo.