Cognome: **Nome**: **# Matricola**: **Riga**: **Col**:

Algoritmi e Strutture Dati – 26/07/2021 – Parte A

A1 – Complessità – Punti ≥ 8

Trovare i limiti superiore e inferiore più stretti possibili per la seguente equazione di ricorrenza:

$$T(n) = \begin{cases} 1 & n < 4 \\ 3T(\lfloor n/2 \rfloor) + 3T(\lfloor n/4 \rfloor) + n^2 & n \geq 4 \end{cases}$$

A2 – Vettori k-ordinati – Punti ≥ 10

Un vettore è k-ordinato se ogni elemento è spostato di non più di k posizioni rispetto alla posizione corretta.

Per esempio, $A = [1, 4, 5, 2, 3, 7, 8, 6, 10, 9]$ è 2-ordinato.

Scrivere un algoritmo

$$\mathsf{sortK}(\textbf{int}[\,]\ A,\ \textbf{int}\ n,\ \textbf{int}\ k)$$

che prenda in input un vettore A di dimensione n, k-ordinato con k noto a priori e molto più piccolo di $\log n$, e lo ordini totalmente.

È possibile risolvere questo problema in tempo:

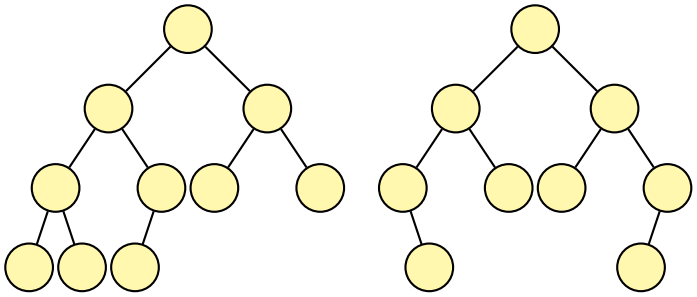
- $O(n \log n)$ o superiori; tali soluzioni non verranno considerate;
- $O(nk)$; tale soluzione darà origine ad un punteggio pari al 60%;
- $O(n \log k)$; tale soluzione darà origine ad un punteggio pari al 100%;
- È ovviamente possibile che esistano altre soluzioni, che verranno valutate a seconda della complessità.

Discutere informalmente la correttezza dell'algoritmo e la sua complessità computazionale, argomentando bene quest'ultima.

A3 – Alberi ben bilanciati – Punti ≥ 12

Un nodo t è *ben-bilanciato* se il numero di nodi nel sottoalbero destro di t e il numero di nodi nel sottoalbero sinistro di t differiscono al più di 1. Un albero T è *ben-bilanciato* se tutti i suoi nodi sono ben-bilanciati (definizione inventata, ovviamente).

Negli esempi successivi, l'albero binario a sinistra non è ben-bilanciato, mentre lo è quello di destra.



Scrivere un algoritmo

$$\textbf{boolean}\ \mathsf{wellBalanced}(\textsc{Tree}\ T)$$

che prenda in input un albero binario T e restituisca **true** se T è ben-bilanciato, **false** altrimenti.

Discutere informalmente la correttezza dell'algoritmo e la sua complessità computazionale.

Cognome: **Nome**: **# Matricola**: **Riga**: **Col**:

Algoritmi e Strutture Dati – 26/07/2021 – Parte B

B1 – Punto d'incontro – Punti ≥ 8

Sia $G = (V, E)$ un grafo orientato rappresentante una rete di trasporto. Il grafo è pesato tramite una funzione $w : V \times V \to \mathbb{R}$, dove $w(u, v)$ corrisponde al *costo* dell'arco (u, v).

Due persone che si trovano nei nodi $s, t \in V$ vogliono incontrarsi in un nodo neutro x; una volta scelto il nodo, entrambi voglio incorrere nel minor costo totale possibile per raggiungerlo (dato dalla somma del costo degli archi attraversati), ma contemporaneamente non vogliono pagare un costo maggiore rispetto all'altra persona. Il problema è capire se tale nodo esiste.

Scrivere un algoritmo

boolean meetingPoint(Graph G, Node s, Node t)

che restituisca **true** se esiste un nodo che può essere raggiunto con lo stesso costo minimo da entrambi, **false** altrimenti.

Discutere informalmente la correttezza dell'algoritmo e la sua complessità computazionale.

B2 – PrintBits – Punti ≥ 10

Scrivere un algoritmo

printBits(**int** n)

che dato un intero n, stampi tutte le stringhe binarie di n bit che non contengono bit 1 consecutivi.

Discutere informalmente la correttezza dell'algoritmo e la sua complessità computazionale.

Ad esempio, per $n = 5$ le stringhe da stampare sono le seguenti:

```
00000 00001 00010 00100 00101
01000 01001 01010 10000 10001
10010 10100 10101
```

B3 – Sottosequenze k-contigue – Punti ≥ 12

Dato un vettore V, una sottosequenza di V è *k-contigua* se non deriva dalla cancellazione di più di k elementi consecutivi dalla sequenza originale. Il *valore* di una sottosequenza è la somma dei suoi valori. Una sottosequenza k-contigua di V è *massimale* se ha il valore massimo fra tutte le sottosequenze k-contigue di V.

Scrivere un algoritmo

int maxSumK(**int**[] V, **int** n, **int** k)

che prenda in input un vettore V contenente n interi e restituisca il valore della sottosequenza k-contigua massimale.

Discutere informalmente la correttezza dell'algoritmo e la sua complessità computazionale.

Per esempio, dato $V = [1, 2, 3, 4, 5, 6]$, $[1, 3, 5]$ è una sottosequenza 1-contigua di V, ma non è 0-contigua; $[1, 4, 6]$ è una sottosequenza 2-contigua di V, ma non è 1-contigua perché sono stati cancellati 2 valori consecutivi (2 e 3); $[1, 5]$ è una sottosequenza 3-contigua di V, ma non è 2-contigua.