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

Algoritmi e Strutture Dati – 1/2/2024 – Parte A

Esercizio -1 Iscriversi allo scritto entro la scadenza. In caso di inadempienza, -1 al voto finale.

Esercizio 0 Scrivere correttamente nome, cognome, numero di matricola, riga e colonna su tutti i fogli consegnati. Consegnare foglio A4 e foglio protocollo di bella. In caso di inadempienza, -1 al voto finale.

A1 – Complessità – Punti ≥ 8

Si consideri questo algoritmo:

```
int fun(int[] A1, int[] A2, int n)
    return fr(A1, A2, 1, n, 1, n)
```

```
int fr(int[] A1, int[] A2, int s1, int e1, int s2,
int e2)
    if s1 == e1 then
        return iif[A1[s1] > A2[s1], 1, −1)
    else
        sum = 0
        int m1 = ⌊(s1 + e1)/2⌋
        int m2 = ⌊(s2 + e2)/2⌋
        shuffle(A1, s1, e1)
        shuffle(A2, s2, e2)
        sum += fr(A1, A2, s1, m1, s2, m2)
        sum += fr(A1, A2, s1, m1, m2 + 1, e2)
        mergeSort(A1, s1, e1)
        mergeSort(A2, s2, e2)
        sum += fr(A1, A2, m1 + 1, e1, s2, m2)
        sum += fr(A1, A2, m1 + 1, e1, m2 + 1, e2)
        return sum
```

dove $\mathsf{shuffle}(A, s, e)$ è una funzione di costo $O(n)$ che mescola casualmente i valori del sottovettore $A[s \dots e]$. Per semplicità, si consideri solo il caso in cui n sia una potenza esatta di 2.

Scrivere l'equazione di ricorrenza associata a questo algoritmo e calcolare la complessità computazionale che ne deriva.

A2 – Tree Repair Shop 2 – Punti ≥ 10

Come nell'esercizio del mese scorso, avete memorizzato un grafo non orientato G che rappresenta un albero non radicato, pronto per essere utilizzato come testcase per il corso di Algoritmi. Il vostro compagno di appartamento burlone vi ha fatto un altro scherzo: questa volta, ha rimosso un certo numero di archi dal grafo e ora vi ritrovate con una foresta!
Scrivete un algoritmo

$$\mathsf{repairTree}(\textsc{Graph}\ G)$$

che prenda in input il grafo modificato e lo renda nuovamente un albero di n nodi (non necessariamente quello originale), aggiungendo uno o più archi con l'operazione $G.\mathsf{insertEdge}(x, y)$ e conservando gli archi rimasti.

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

A3 – Un singolo – Punti ≥ 12

Suggerimento: leggere bene il testo!
Si consideri un vettore A contenente n interi con le seguenti caratteristiche:

- tutti i numeri presenti nel vettore compaiono un numero pari di volte, tranne un numero x che compare una sola volta;
- se $A[i] = y \neq x$, allora si verifica una e una sola di queste condizioni:
  - $A[i-1] = y \wedge A[i+1] \neq y$
  - $A[i-1] \neq y \wedge A[i+1] = y$

Scrivere un algoritmo

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

che restituisca il valore x.

Discutere informalmente la correttezza dell'algoritmo e la sua complessità computazionale. Soluzioni in tempo lineare o superiore non verranno considerate.

Per esempio, per il vettore $A = [2, 2, 3, 4, 4, 2, 2, -1, -1]$, la risposta è 3.

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

Algoritmi e Strutture Dati – 1/2/2024 – Parte B

Esercizio -1 Iscriversi allo scritto entro la scadenza. In caso di inadempienza, -1 al voto finale.

Esercizio 0 Scrivere correttamente nome, cognome, numero di matricola, riga e colonna su tutti i fogli consegnati. Consegnare foglio A4 e foglio protocollo di bella. In caso di inadempienza, -1 al voto finale.

B1 – Metà o meno – Punti ≥ 8

Scrivere un algoritmo

$$\mathsf{lessThanHalf}(\textbf{int}\ n)$$

che prenda in input un intero n e stampi tutte le sequenze $A = a_1, \ldots, a_k$ di interi positivi che iniziano con il valore n ($a_1 = n$), tali per cui ogni elemento A_i con $i > 1$ sia tale che $A_{i-1} \geq 2 \cdot A_i$.

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

Per esempio, se $n = 7$, queste sono le sequenze che devono essere stampate, non necessariamente in quest'ordine: [7], [7, 1], [7, 2], [7, 2, 1], [7, 3], [7, 3, 1]

B2 – Alberibelli – Punti ≥ 10

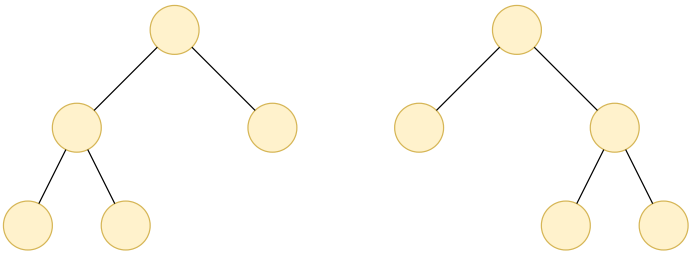
Un *alberobello* è un albero binario in cui tutti i nodi hanno 0 o 2 figli; in altre parole, nessuno nodo ha 1 figlio.
Scrivere un algoritmo

$$\textbf{int}\ \mathsf{countAB}(\textbf{int}\ n)$$

che prenda in input un intero n e restituisca il numero di alberibelli strutturalmente diversi contenenti n nodi.

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

I due alberi seguenti sono gli alberibelli strutturalmente diversi di dimensione $n = 5$.



B3 – LCNCS – Punti ≥ 12

Una stringa S è una *non-consecutive subsequence* (sottosequenza non consecutiva) di una stringa T se S è ottenuta da T rimuovendo alcuni caratteri e mantenendo l'ordine, senza però conservare caratteri consecutivi. Per esempio, `"A"`, `"B"`, `"C"` e `"AC"` sono sottosequenze non consecutive di `"ABC"`, mentre `"AB"`, `"BC"`, `"ABC"` non lo sono perchè contengono caratteri consecutivi della stringa originale.

Una *longest common non-consecutive subsequence* (LCNCS, sottosequenza comune massimale non-consecutiva) di due stringhe T e U è una sottosequenza non-consecutiva sia di T che di U, che ha lunghezza massima fra tutte le stringhe che sono LCNCS sia di T che di U. Per esempio, `"BD"` è una LCNCS di `"ABCD"` e `"BCDC"`.

Scrivere un algoritmo

$$\textbf{int}\ \mathsf{lcncs}(\textsc{Item}[\,]\ T,\ \textsc{Item}[\,]\ U,\ \textbf{int}\ n,\ \textbf{int}\ m)$$

che prenda in input due stringhe T, U di lunghezza n, m e restituisca la lunghezza delle LCNCS di T e U. Nell'esempio precedente, l'algoritmo deve restituire 2.

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