COGNOME: **NOME:** **MATRICOLA:**

Intermedio di Algoritmi e Strutture Dati (9CFU), 4 Aprile 2024.

Esercizio 1 [5 punti]
Quale delle seguenti sequenze di funzioni è tale che ogni funzione è O della successiva?

(A) $\sqrt{n}$, $n \log n$, $n^{\log n}$, $2^{\log n^2}$, 2^n

(B) $n^{4/3}$, $\binom{n}{2}$, $n(\log n)^3$, $2^{\sqrt{\log n}}$, $n^{\log n}$

(C) $n^{4/3}$, $n(\log n)^3$, $\binom{n}{2}$, $2^{\sqrt{\log n}}$, $n^{\log n}$

(D) $\sqrt{n}$, $\log n!$, $n^2 \log n$, $\binom{n}{n-3}$, $2^{\sqrt{n}}$

Esercizio 2 [5 punti]
Si consideri l'algoritmo USELESS descritto di seguito, il cui input è un intero $n \geq 0$ potenza di 4. Sia $T(n)$ il tempo di esecuzione nel caso peggiore di USELESS con input n. Assumendo che il costo di sommare due interi sia $O(1)$, indipendentemente dagli interi considerati, quale delle seguenti affermazioni è vera?

Algorithm USELESS(n)

```
if n ≤ 1 then
    return n
end if
sum = 0
for i = 0 to n^3 − 1 do
    sum = sum + 1
end for
return USELESS(3n/4) + USELESS(3n/4) + sum
```

(A) $T(n) = \Theta(n^3)$

(B) $T(n) = \Theta(n^{\log_{4/3} 2})$

(C) Nessuna delle altre

(D) $T(n) = \Theta(n^{\log_{3/4} 2})$

Esercizio 3 [5 punti]
Indicare tutte e sole le affermazioni vere.

(A) È possibile ordinare una sequenza arbitraria di 6 interi tramite al più 9 confronti.

(B) Siano $A[1..n]$ e $B[1..n]$ due array, entrambi di n interi, tali che ciascuno dei loro elementi è compreso tra 1 e n. Sia $C[1..n]$ un array tale che $C[i] = A[i] \cdot B[i]$ per ogni $i = 1, \ldots, n$. È possibile ordinare l'array C in $O(n)$.

(C) Sia $A[1..n]$ un array di n interi distinti tali che $A[1] < A[2] < \cdots < A[n]$ e $A[1] > 0$. Un qualsiasi algoritmo che trovi un indice i tale che $A[i] = i$ oppure indichi che tale indice non esiste richiede $\Omega(\log n)$ confronti nel caso peggiore.

(D) BINARY-SEARCH è un algoritmo ottimo.

Esercizio 4 [5 punti]
Indicare quale delle seguenti affermazioni è vera.

VERO, basta aumentare la priorita' della chiave dell'oggetto ad ogni inserimento (CODA PRIOR.)

(A) È possibile implementare una struttura dati del tipo LIFO tramite un min-heap.

(B) Possiamo utilizzare un heap per ordinare in tempo lineare un insieme di numeri. F

(C) La procedura di max-heapify può essere lanciata su qualsiasi albero binario quasi completo. F

(D) Utilizziamo sempre un max-heap per implementare una coda di priorità. F

Esercizio 5 [5 punti]
Un albero binario è:

(A) Un albero tale che ogni nodo ha al più due figli.

(B) Un albero per la memorizzazione e ricerca di numeri binari.

(C) Un albero tale che, dato un nodo x, il figlio sinistro ha un valore minore del valore associato al nodo x e il figlio destro ha un valore maggiore del valore associato al nodo x.

(D) Un albero tale che ogni nodo è associato ad un valore maggiore di tutti i suoi figli.

SBARRAMENTO - 15 punti sugli esercizi precedenti

Esercizio 6 [10 punti]
Un array $A[1..n]$ di n interi distinti è *quasi ordinato* se ciascuno dei suoi elementi è fuori posto rispetto all'ordinamento crescente di al più una posizione. In altre parole, un qualsiasi elemento che nell'array ordinato occuperebbe la posizione i, in un array quasi ordinato può occupare una qualsiasi delle posizioni $i-1, i, i+1$. Per esempio, l'array $< 4, 1, 7, 10, 8 >$ è quasi ordinato, mentre l'array $< 1, 7, 8, 4, 10 >$ non lo è.

Vogliamo sviluppare un algoritmo efficiente che, dati in input un array $A[1..n]$ quasi ordinato ed un intero k, restituisca l'indice di k in A, se esiste, o il valore -1, se k non appare in A.

1. Spiegare brevemente perchè un algoritmo che risolve il nostro problema in $\Theta(\log n)$ è ottimo. **[4 punti]**

2. Siano $A[1..n]$ un array quasi ordinato di n interi distinti, k un intero e m un indice tale che $1 < m < n$. Dimostrare che, se k appare in A e $A[m] > k$, allora: o $A[m-1] = k$, o $A[m+1] = k$, oppure k appare in $A[1..m-2]$. **[3 punti]**

3. Usando l'osservazione del punto 2, scrivere lo pseudo-codice di un algoritmo Divide et Impera che risolve il nostro problema in $\Theta(\log n)$. **[3 punti]** $m \leftarrow \left\lfloor \frac{l+r}{2} \right\rfloor$

Ex 6

1. Sappiamo che l'algo. BINARY-SEARCH e' ottimo poichè impiega un num. di passi pari a $O(\log n)$ nel caso peggiore per stabilire se la chiave K e' contenuta o meno nell'array ordinato.

2. Scriviamo il nostro enunciato da dimostrare: → essenzialmente ci dice di guardare le posiz. $[1..m+1] \setminus [m]$

" $A[1..n]$ quasi ordinato, n interi distinti, $K, m \in \mathbb{N}$: $1 < m < n$.
Se $K \in A$ e $A[m] > K$

$\Rightarrow$ $A[m-1] = K$ o $A[m+1] = K$ o $K \in A[1..m-2]$ "

OSSERVAZIONE 1: Se $A[i]$, $i \in [1..n]$ e' in pos. corretta e $A[i] > K$, allora sicuramente gli elementi $A[1..i-1]$ sono $\leq K$ mentre quelli $A[i+1..n]$ sono sicuramente $> K$.

OSSERVAZIONE 2: Se $A[i]$, $i \in [1..n]$ non è in pos. corretta, allora vuol dire che sono possibili due casi mutuamente esclusivi:

- $i-1$ è pos. corretta per $A[i]$ $\Leftrightarrow$ i è pos. corretta per $A[i-1]$
- $i+1$ è pos. corretta per $A[i]$ $\Leftrightarrow$ i è pos. corretta per $A[i+1]$

OSSERVAZIONE 3: Se $i \in 1..n$ allora $A[i] < A[i+K]$, $2 \leq K \leq n-i$ (Dim. banale per assurdo) (*)

- Dim per assurdo l'enunciato. Supp. quindi che A quasi ordinato, $K \in A$, $1 < m < n$, $A[m] > K$ ma $(A[m-1] \neq K$, $A[m+1] \neq K$, $K \notin A[1..m-2]$ → $K \in A[m+2..n]$
- Quindi necessariamente $K \in A[m+2..n]$. Per l'osservazione 2 sappiamo pero' che $A[m] < A[m+K]$ $2 \leq K \leq n-m$
- Inoltre $K < A[m]$ quindi K non puo' appartenere a $A[m+2..n]$ ↯

(*) Per assurdo supp. $i \in 1..n$ e che $\exists K \in [2..n-i]$: $A[i] > A[i+K]$ (non $\geq$ perchè elementi distinti).

Quindi vorrebbe dire che l'elemento in posizione i dovrebbe stare in una posizione a destra di $A[i+K]$, diciamo $A[f]$, $f > i+K$

Ma quindi

$$f - i > i + K - i = K \geq 2 \quad ↯$$

assurdo poichè A e' quasi ordinato.