-Prova intermedia di Algoritmi-

(11 Aprile 2019)

Cognome .... **Nome** .... **Matricola** ....

Note

1. La leggibilità è un prerequisito: parti difficili da leggere potranno essere ignorate.
2. Quando si presenta un algoritmo è fondamentale spiegare l'idea soggiacente il suo funzionamento e motivarne la correttezza.

Domanda 1 Dare la definizione della classe $\Theta(f(n))$. Mostrare che la ricorrenza

$$T(n) = \frac{1}{3}\,T(n-1) + 3n^2$$

ha soluzione in $\Theta(n^2)$.

Soluzione: Per provare che $T(n) = O(n^2)$ dobbiamo dimostrare che $T(n) \leqslant cn^2$, per un'opportuna costante $c > 0$. Procediamo per induzione:

$$\begin{aligned} T(n) &= 1/3\,T(n-1) + 3n^2 \\ &\leqslant 1/3\,c(n-1)^2 + 3n^2 \\ &\leqslant 1/3\,cn^2 + 3n^2 \\ &\leqslant cn^2 \end{aligned}$$

dove, per la validità dell'ultima disuguaglianza $1/3\,cn^2 + 3n^2 \leqslant cn^2$, occorre che

$$2/3\,cn^2 \geqslant 3n^2$$

ovvero, $c \geqslant 9/2$, con n qualunque.

Per provare $T(n) = \Omega(n^2)$ dobbiamo dimostrare che $T(n) \geqslant dn^2$, per un'opportuna costante $d > 0$. La prova per induzione non usa neppure l'ipotesi induttiva. Infatti

$$T(n) = 1/3\,T(n-1) + 3n^2 \geqslant dn^2$$

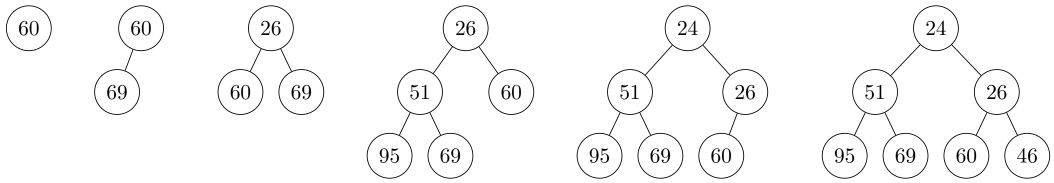
purché $d \leqslant 3$.

Domanda 2 Dare la definizione di min-heap. Data la sequenza di elementi `60, 69, 26, 95, 51, 24, 46, 80, 60, 38, 12, 70` si specifichi il min-heap ottenuto, inserendo uno alla volta questi elementi nell'ordine indicato, a partire da uno heap vuoto. Si descriva sinteticamente come si procede per arrivare al risultato.

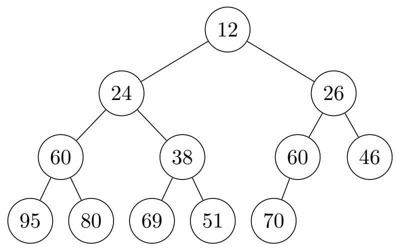
Soluzione: Un min-heap è un albero binario ordinato quasi-completo con la proprietà che per ogni nodo x, se x non è radice, il genitore di x ha chiave minore o uguale a quella di x, o equivalentemente ogni nodo x ha chiave minore o uguale a quella dei suoi successori.

Si procede inserendo nel min-heap i vari elementi con la procedura `HeapInsert` a partire da heap vuoto. La procedura inserisce l'elemento come prima foglia utile (ultimo elemento dell'array) e richiama la procedura `MinHeapifyUp` per ripristinare la proprietà di min-heap.

I primi passi producono gli heap indicati in figura.



Il risultato finale è quindi:



che in forma di array, è [12, 24, 26, 60, 38, 60, 46, 95, 80, 69, 51, 70].

Domanda 3 Realizzare una funzione `Diff(A,k)` che, dato un array `A[1,n]` ordinato in senso crescente, verifica se esiste una coppia di indici `i, j` tali che `A[i]` $-$ `A[j]` $=$ `k`. Restituisce la coppia di indici se esiste e (0,0) altrimenti. La funzione non deve alterare l'input e deve operare in spazio costante. Scrivere lo pseudocodice e valutarne la complessità.

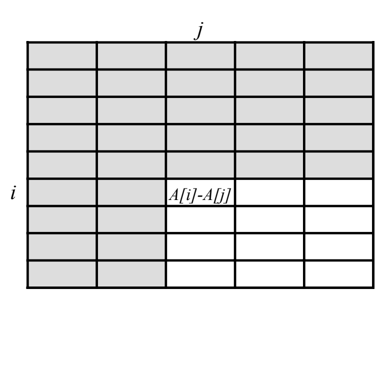
Soluzione:

Il codice può essere:

```
diff(A, n, k):
    i=1
    j=1
    while (i<=n) and (j<=n) and (A[i]- A[j] <> k)
        if (A[i]- A[j] < k)
            i++
        else
            j++

    if (i <= n) and (j<=n)
        return (i,j)
    else
        return (0,0)
```

È facile vedere che si mantiene l'invariante $\forall(i',j') \in [1,n].(i' < i) \vee (j' < j) \Rightarrow A[i] - A[j] \neq k$, ovvero una coppia (i',j') tale che $A[i] - A[j] = k$ può esistere solo tra le coppie ancora esplorabili ($i' \geqslant i$ e $j' \geqslant j$), ovvero, graficamente nella parte non grigia:



Infatti, inizialmente, con $i = j = 1$, l'invariante è vacuamente vero.
Ad ogni iterazione, se entro nel ciclo, ci sono due possibilità:

- Se $A[i] - A[j] < k$, allora incremento i. In questo modo, escludo dall'esplorazione le coppie (i, j') con $j' \geqslant j$ per le quali, dato che l'array è crescente e quindi $A[j] \leqslant A[j']$, vale

$$A[i] - A[j'] \leqslant A[i] - A[j] < k.$$

  Dunque non escludo coppie utili, l'invariante continua a valere.

- Dualmente, $A[i] - A[j] > k$, allora incremento j. In questo modo, escludo dall'esplorazione le coppie (i', j) con $i' \geqslant i$ per le quali, dato che l'array è crescente e quindi $A[i] \leqslant A[i']$, vale

$$A[i'] - A[j] \geqslant A[i] - A[j] > k.$$

  Dunque, anche in questo caso, non escludo coppie utili, l'invariante continua a valere.

Quando esco dal ciclo, se $A[i] - A[j] = k$, ho concluso con successo. Altrimenti deve essere $i > n$ o $j > n$, che unitamente all'invariante, mi permettono di concludere che per ogni $i, j \in [1, n]$, $A[i] \neq A[j]$, come desiderato.

Da questo la correttezza segue immediatamente. La complessità è lineare. Il numero di iterazioni è pari al più a $2n - 1$, dato che i e j partono da 1, sono limitate da n ed ogni iterazione aumenta una delle due. Dato che ciascuna iterazione ha costo costante, ottengo $T(n) = O(2n - 1) = O(n)$.