Prova intermedia di Algoritmi (12 Aprile 2018)

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 di $O(f(n))$ e mostrare che la ricorrenza

$$T(n) = T\left(\frac{n}{2}\right) + T\left(\frac{n}{4}\right) + 2n$$

ha soluzione $O(n)$.

Soluzione: Con $O(f(n))$ si indica l'insieme delle fuzioni che hanno come limite asintotico superiore $f(n)$, ovvero

$$O(f(n)) = \{g(n) \mid \exists c > 0.\ \exists n_0 \in \mathbb{N}.\ \forall n \geqslant n_0.\ 0 \leqslant g(n) \leqslant cf(n)\}$$

Si scrive spesso $g(n) = O(f(n))$ anziché $g(n) \in O(f(n))$.

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

$$\begin{array}{rcl} T(n) & = & T(n/2) + T(n/4) + 2n \\ & \leqslant & n/2\,c + n/4\,c + 2n \\ & = & (2 + 3/4\,c)n \\ & \leqslant & cn \end{array}$$

quando $2 + 3/4\,c \leqslant c$, ovvero $c \geqslant 8$.

Domanda 2 Dare la definizione di max-heap. Dato un array `A[1..12]` con sequenza di elementi `[94, 69, 58, 26, 67, 51, 25, 46, 18, 47, 60, 32]` si indichi il risultato della procedura `BuildMaxHeap` applicata ad `A`. Si descriva sinteticamente come si procede per arrivare al risultato.

Soluzione: Un max-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 maggiore o uguale a quella di x, o equivalentemente ogni nodo x ha chiave maggiore o uguale a quella dei suoi successori.

Per ottenere un max-heap a partire da un array si usa la procedura `BuildMaxHeap(A)` (vedi libro) e si ottiene: [94, 69, 58, 46, 67, 51, 25, 26, 18, 47, 60, 32]

Domanda 3 Realizzare una funzione `Cube(A,n)` che, dato un array `A[1,n]` ordinato in senso crescente, verifica se esiste una coppia di indici `i, j` tali che `A[j]` = `A[i]`^{3}. Restituisce la coppia se esiste e (0,0) altrimenti. Scrivere lo pseudocodice e valutare la complessità.

Soluzione:

Il codice può essere:

```
cube(A,n) {                         // assume A[1,n] ordinato
                                    // ritorna una coppia i, j
   i=1                              // tale che A[i]^3 = A[j]
   j=1
   while (i<=n) and (j<=n) and (A[i]^3 <> A[j])
      if (A[i]^3 > A[j])
         j++
      else
         i++

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

È facile vedere che si mantiene l'invariante $\forall i' \in [0, i). \ \forall j' \in [0, j). \ A[i]^3 \neq A[j]$. Da questo la correttezza segue immediatamente. Il costo è lineare (il numero di iterazioni è pari ad al più $2n$, ed ogni iterazione ha costo costante), quindi $T(n) = O(n)$.