Corso di Laurea in Informatica *A.A. 2008-2009*

Soluzione Problemi 2.4, 2.8 e 2.9

Esame di Algoritmi 1 *Docente: Irene Finocchi*

Problema 2.4. Per ogni coppia di funzioni $f(n)$ e $g(n)$, si dica se $f(n) = O(g(n))$, $f(n) = \Omega(g(n))$, e $f(n) = \Theta(g(n))$ completando la seguente tabella:

$f(n)$	$9^{\log_3 n}$	$n \log\log n$	$2^{n/2}$	1	$n^2 + 8n$	n^n	2^n	$4^{\log_2 n}$
$g(n)$	$n\sqrt{n}$	$n^{1+\epsilon}$, $\epsilon > 0$	$n^{\log\log n}$	2^9	$(n \log n)^2$	$n!$	$2^{n/4}$	$n^{\log_2 4}$
O								
Ω								
Θ								

Soluzione. La soluzione è illustrata nella seguente tabella:

$f(n)$	$9^{\log_3 n}$	$n \log\log n$	$2^{n/2}$	1	$n^2 + 8n$	n^n	2^n	$4^{\log_2 n}$
$g(n)$	$n\sqrt{n}$	$n^{1+\epsilon}$, $\epsilon > 0$	$n^{\log\log n}$	2^9	$(n \log n)^2$	$n!$	$2^{n/4}$	$n^{\log_2 4}$
O		•		•	•			•
Ω	•		•	•		•	•	•
Θ				•				•

◁ ◇ ▷

Problema 2.8. L'algoritmo seguente, dati due array A e B di n elementi ciascuno, restituisce `true` se A e B hanno un elemento in comune, `false` altrimenti:

```
    algoritmo elementoInComune(array A e B, interi n, i e j) → booleano
1.      if ( i = j ) then
2.          for k = 0 to n do
3.              if ( A[k] = B[i] ) then return true
4.          return false
5.      else
6.          m ← (i + j)/2
7.          if ( elementoInComune(A, n, B, i, m) ) then return true
8.          else return elementoInComune(A, n, B, m + 1, j)
```

La chiamata iniziale è `elementoInComune`$(A,n,B,0,n-1)$. Qual è il tempo di esecuzione del passo base? Qual è il tempo di esecuzione totale? Si discutano il caso migliore e quello peggiore, assumendo che il passaggio di parametri costi tempo $O(1)$.

Soluzione. Nel caso peggiore, il tempo di esecuzione del caso base ($i = j$) è $O(n)$. Sia $b = j - i + 1$ la cardinalità della porzione di array B considerata in una generica chiamata dell'algoritmo `elementoInComune`, e sia $T(n, b)$ il tempo di esecuzione di una generica chiamata dell'algoritmo nel caso peggiore. Notiamo che per la chiamata iniziale si avrà $b = n$.

$$T(n,b) = \begin{cases} 2T\left(n, \frac{b}{2}\right) + c & \text{se } b \geq 2 \\ O(n) & \text{se } b = 1 \end{cases}$$

per qualche costante positiva c. Questo implica

$$T(n,b) = 2^i \cdot T\left(n, \frac{b}{2^i}\right) + c\sum_{j=0}^{i-1} 2^j \leq b \cdot T(n,1) + cb = b \cdot O(n) + cb = O(bn)$$

Dato che nella chiamata iniziale si ha $b = n$, nel caso peggiore l'algoritmo richiederà tempo $O(n^2)$. Nel caso migliore invece (che si ha se $A[0] = B[0]$)

$$T(n,b) = \begin{cases} T\left(n, \frac{b}{2}\right) + c & \text{se } b \geq 2 \\ O(1) & \text{se } b = 1 \end{cases}$$

per qualche costante positiva c. Questo implica

$$T(n,b) = T\left(n, \frac{b}{2^i}\right) + ci = T(n,1) + c \log b = O(\log n)$$

◁ ◇ ▷

Problema 2.9. Definiamo il problema della ricerca di un indice speciale come segue:

> data una sequenza ordinata di n interi distinti, sia positivi che negativi, $a_1 < a_2 < \ldots < a_n$, determinare se esiste un indice i tale che $a_i = i$.

Progettare un algoritmo che risolve il problema della ricerca di un indice speciale in tempo $O(\log n)$.

Soluzione. È sufficiente simulare la ricerca binaria, distinguendo i seguenti tre casi:

- Se $a_{n/2} = n/2$ restituisci `true`
- Se $a_{n/2} < n/2$ prosegui a destra, ovvero sugli elementi $a_{n/2+1} \ldots a_n$. Poiché gli elementi sono tutti distinti, $a_{n/2-1} \leq a_{n/2} - 1 < n/2 - 1$. Proseguendo in questo modo, si può dimostrare facilmente che $a_i < i$ per ogni $i < n/2$.
- Se $a_{n/2} > n/2$ prosegui a sinistra, ovvero sugli elementi $a_1 \ldots a_{n/2-1}$. Poiché gli elementi sono tutti distinti, $a_{n/2+1} \geq a_{n/2} + 1 > n/2 + 1$. Proseguendo in questo modo, si può dimostrare facilmente che $a_i > i$ per ogni $i > n/2$.