

# Esercizi di Algoritmi e Strutture Dati

Ultimo aggiornamento: 9 novembre 2009

Moreno Marzolla  
marzolla@cs.unibo.it

6 Novembre 2009

## 1 Esercizi sulla complessità asintotica

### 1.1 Vero o falso?

Per ciascuna delle seguenti affermazioni, dire se è vera o falsa, fornendo una dimostrazione:

1.  $3^n = O(2^n)$
2.  $2^{2n} = O(2^n)$
3.  $2^{n+1} = O(2^n)$
4.  $\log_3 n = O(\log_2 n)$
5.  $\ln n = O(n^\alpha)$ , per ogni  $\alpha > 0$  ( $\ln$  è il logaritmo in base  $e$ ).

### 1.2 Dimostrazioni per induzione

Il seguente esercizio mostra come sia particolarmente importante prestare attenzione a come si fanno le dimostrazioni per induzione. Consideriamo la seguente relazione di ricorrenza:

$$T(n) = \begin{cases} 2T(\lfloor n/2 \rfloor) + 1 & n > 1 \\ 1 & n = 1 \end{cases}$$

È relativamente facile rendersi conto che  $T(n) = O(n)$ . Riuscite a dimostrare per induzione che  $T(n) \leq cn$ , per una opportuna costante  $c > 0$ ? Riuscite a dimostrare che  $T(n) \leq cn - b$ , per opportune costanti  $c > 0$  e  $b$  arbitraria?

### 1.3 Problema con il caso base

Il seguente esercizio mostra un esempio in cui si presenta un problema con il caso base, facilmente aggirabile. Consideriamo la seguente relazione di ricorrenza:

$$T(n) = \begin{cases} 2T(\lfloor n/2 \rfloor) + n & n > 1 \\ 1 & n = 1 \end{cases}$$

Dimostrare per induzione che  $T(n) = O(n \log n)$  (ossia, dimostrare che  $T(n) \leq cn \log n$  per una opportuna costante  $c > 0$  e per  $n > n_0$ ).

## 2 Definizione e analisi di algoritmi

**Nota:** nel caso sia richiesto di descrivere un algoritmo, è possibile usare pseudocodice, oppure un frammento di codice Java, a scelta. Il codice Java non deve necessariamente compilare, è sufficiente che descriva completamente l'algoritmo richiesto. Come suggerimento personale, consiglio di utilizzare lo pseudocodice in modo da non perdere tempo sui dettagli della sintassi del linguaggio. Nel caso si usi pseudocodice, prestare particolare attenzione a specificare in modo dettagliato *tutti* i passi eseguiti dall'algoritmo. Nel caso di Java si assume sempre che l'indice degli array parte da zero. Nel caso dello pseudocodice si può assumere che parta da uno (ma è necessario specificarlo chiaramente).

### 2.1 Caso pessimo di Quick Sort

Consideriamo l'algoritmo Quick Sort nella versione deterministica vista a lezione, in cui il valore del pivot è sempre scelto come il valore del primo elemento dell'array da partizionare. Quando si verifica il caso pessimo per questa versione dell'algoritmo?

### 2.2 La “bandiera nazionale”

(*problema 4.7 del libro di testo*). Il problema della bandiera nazionale italiana è così definito. Sia  $A$  un array di dimensione  $n$ , i cui elementi possono assumere solo uno di tre possibili colori: bianco, verde e rosso. Vogliamo ordinare l'array in modo che tutti gli elementi verdi precedano quelli bianchi, e gli elementi bianchi precedano quelli rossi. L'algoritmo deve richiedere tempo  $O(n)$  nel caso peggiore, può solo scambiare elementi e non deve usare altri array temporanei di appoggio. Non può nemmeno utilizzare contatori per tenere traccia del numero di elementi di un certo colore presenti (quindi non si può usare una variante dell'algoritmo *counting sort*, che vedremo a lezione). L'algoritmo, infine, deve ordinare gli elementi facendo una singola scansione dell'array. Assicurarsi che l'algoritmo funzioni anche se mancano elementi di qualcuno dei colori.

**Suggerimento:** ripensare a come funziona la procedura *partition* dell'algoritmo Quick Sort, e provare ad adattarla per risolvere questo problema.

### 2.3 Un problema apparentemente complicato

(*problema 4.8 del libro di testo*). Descrivere un algoritmo che dato un array  $A$  di  $n$  numeri interi, tutti compresi nell'intervallo  $[1, k]$  ( $k > 1$ ), preprocessa l'array in tempo  $O(n + k)$  in modo da generare una opportuna struttura dati che consenta di rispondere in tempo  $O(1)$  a query del tipo: “quanti elementi di  $A$  sono compresi nell'intervallo  $[a, b]$ ?” (per qualsiasi  $1 \leq a \leq b \leq k$ )

### 2.4 L'attitudine al problema vs l'attitudine alla soluzione

(*problema 4.11 del libro di testo*). Progettare un algoritmo che, dato in input un array di  $n$  numeri reali, trovi in tempo lineare un *qualsiasi numero* che **non** appartenga all'array. Dimostrare poi che  $\Omega(n)$  è un limite inferiore al numero di passi necessari per risolvere questo problema.