

Esercizi: ricerca, ordinamento e divide-et-impera

Esercizio 1

Progettare un algoritmo di tipo *divide-et-impera* per calcolare a^n con $O(\log n)$ moltiplicazioni. [Suggerimento: $a^n = (a^{n/2})^2$ per n pari, e $a^n = a \cdot (a^{n/2})^2$ per n dispari.]

Esercizio 2

Progettare un algoritmo per ordinare **in loco** un array a di n interi, il cui valore può essere solo 0 o 1. L'algoritmo deve richiedere tempo lineare nel caso pessimo e può solo scambiare elementi. In particolare, non può usare contatori per mantenere il numero di elementi di un certo valore.

Esercizio 3 ([CLRS] Esercizio 2.3-7)

Descrivere un algoritmo di costo in tempo $\Theta(n \log n)$ che, dato un insieme S di n numeri interi e un altro intero k , determini se esistono due elementi in S la cui somma è esattamente k .

Esercizio 4

Sia a un array di n interi distinti, tale che esiste una posizione j , $1 \leq j \leq n$, per cui:

- gli elementi nel segmento $a[1, j]$ sono in ordine crescente;
 - gli elementi in $a[j+1, n]$ sono in ordine decrescente;
 - $a[j] > a[j+1]$, se $j < n$.
1. Descrivere un algoritmo che, ricevuto in ingresso a , trova la posizione j in tempo lineare.
 2. Dimostrare che un qualunque algoritmo che risolve il problema suddetto mediante confronti richiede tempo $\Omega(\log n)$ al caso pessimo.
 3. Descrivere un algoritmo **ottimo** di tipo *divide-et-impera* per il problema precedente. Calcolare la complessità al caso pessimo dell'algoritmo indicando, e risolvendo, la corrispondente relazione di ricorrenza.

Esercizio 5

È dato un array a di n interi, alcuni dei quali possono essere ripetuti, ordinato. Si progetti un algoritmo che, ricevuto in ingresso a e un intero k , conti il numero *occ* di occorrenze di k in a .

- Descrivere un algoritmo che richiede o tempo $O(\log n + \text{occ})$ oppure tempo $O(\log n)$.
- Dimostrare che l'algoritmo di complessità $O(\log n)$ è ottimo al caso pessimo.

Esercizio 6 ([CLRS] Problema 2-1)

Dati un array a di n elementi e un valore $k < n$, si consideri una versione modificata del Merge Sort che funziona normalmente fino a quando, nella riduzione ricorsiva del problema, $n > k$; quando invece i sottoproblemi diventano sufficientemente piccoli (di dimensione minore o uguale a k) si ordinano direttamente con Insertion Sort.

1. Scrivere lo pseudocodice dell'algoritmo.
2. Valutare la complessità al caso pessimo in funzione di k e del numero n di elementi da ordinare.
3. Qual è il massimo valore asintotico di k per cui l'algoritmo ha lo stesso tempo di esecuzione asintotico del Merge Sort?
4. In pratica, come dovrebbe essere scelto il valore di k ?