Esercizi

<u>Esercizio 1</u>
Progettare un algoritmo di tipo *divide-et-impera* per contare il numero di elementi pari in un array di interi, e analizzarne la complessità.

<u>Esercizio 2</u>
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 $a^n = a\,(a^{n/2})^2$ per n dispari.]

<u>Esercizio 3</u>
Progettare un algoritmo efficiente per verificare se in un array a di n interi positivi esistono i e j tali che $a[j] = 2*a[i]$, e analizzarne la complessità.

<u>Esercizio 4</u>
Dato un array **ordinato** di n interi positivi, progettare un algoritmo efficiente per verificare se esistono due elementi nell'array la cui somma è k.

<u>Esercizio 5</u>
Dato un array **non ordinato** di n interi positivi, progettare un algoritmo efficiente per verificare se esistono due elementi nell'array la cui somma è k.

<u>Esercizio 6</u>
Sia a un array di n interi distinti, tale che esiste una posizione j, $0 \leq j < n$, per cui:

- gli elementi nel segmento $a[0, j]$ sono in ordine crescente;
- gli elementi in $a[j+1, n-1]$ sono in ordine decrescente;
- $a[j] > a[j+1]$, se $j < n - 1$.

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.

<u>Esercizio 7</u>
Si consideri un array a di n elementi e un valore $k < n$. In una versione modificata dell'algoritmo MERGESORT, l'algoritmo funziona normalmente fino a quando, nella riduzione ricorsiva del problema, $n > k$. Quando invece i sottoproblemi diventano sufficientemente piccoli si ordinano direttamente con una strategia diversa. In particolare gli n/k sottoinsiemi di lunghezza k si ordinano con INSERTIONSORT.
- Si definisca lo pseudocodice dell'algoritmo.
- Se ne studi la complessità (suggerimento: per semplificare lo studio si consideri $n=k^r$).

<u>Esercizio 8</u>
Dati due array a e b, di n e m interi distinti, progettare un algoritmo efficiente per costruire l'array c che rappresenta l'insieme intersezione di a e b.

<u>Esercizio 9</u>
Dati due array a e b, di n e m interi distinti, progettare un algoritmo efficiente per costruire l'array c che rappresenta l'insieme unione di a e b.

Esercizio 10
Progettare un algoritmo efficiente di tipo divide et impera per determinare se un array non ordinato di n interi, contenente solo 0 e 1, contiene più 0 di 1. Analizzare la complessità dell'algoritmo trovato.

Esercizio 11
È dato un array a di n interi, alcuni dei quali possono essere ripetuti, ordinato in modo non decrescente. Si progetti un algoritmo che, ricevuto in ingresso a e un intero k, conta il numero occ di occorrenze di k in a.

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