Progettazione di Algoritmi **Anno Accademico 2021–2022**

Esercizi sulla Tecnica Divide et Impera.

Ugo Vaccaro

1. *Esercizio:* Sia $A = A[1] \dots A[n]$ un vettore di numeri, non necessariamente ordinato. Si scriva un algoritmo basato sulla tecnica Divide et Impera che conti il numero di elementi in A che sono ≥ 0. Si analizzi la complessità di tempo dell'algoritmo proposto, giustificando le affermazioni fatte.

$\diamond$

Si progetti progetti un algoritmo basato sulla tecnica Divide et Impera, e se ne analizzi la complessità, per il seguente problema algoritmico:

Input: Array $A = A[1...n]$ di n numeri interi positivi.
Output:

$$\min_{1 \leq i \leq j} (A[j] - A[i]).$$

(Suggerimento: si proceda come fatto a lezione per il calcolo di $\max_{1 \leq i \leq j}(A[j] - A[i])$).

$\diamond$

2. *Esercizio:* Sia $A = A[1] \dots A[n]$ un vettore di numeri, *non* necessariamente distinti e non necessariamente ordinati. Si scriva un algoritmo basato sulla tecnica Divide et Impera che, dato A e due numeri qualsiasi $L < U$, calcola quanti valori di A appartengono all'intervallo $[L, U]$. Si analizzi la complessità di tempo dell'algoritmo proposto, giustificando le affermazioni fatte.

$\diamond$

3. *Esercizio:* Sia $A = A[1] \dots A[n]$ un vettore di numeri, distinti tra di loro ed ordinati in senso crescente. Si scriva un algoritmo basato sulla tecnica Divide et Impera che, dato A e un numero arbitrario x, restituisce l'indice del più piccolo elemento in A che risulti strettamente maggiore di x. Se non esiste alcun valore di A che soddisfa tale vincolo, l'algoritmo deve restituire "non c'è". Si analizzi la complessità di tempo dell'algoritmo proposto, giustificando le affermazioni fatte.

$\diamond$

4. *Esercizio:* Sia $A = A[1] \dots A[n]$ un vettore di numeri, distinti tra di loro ed ordinati in senso crescente. Si scriva un algoritmo basato sulla tecnica Divide et Impera che, dato A e due numeri qualsiasi $L < U$, calcola quanti valori di A appartengono all'intervallo $[L, U]$. Si analizzi la complessità di tempo dell'algoritmo proposto, giustificando le affermazioni fatte.

$\diamond$

5. *Esercizio:* Sia $A = A[1] \dots A[n]$ un vettore di numeri, ordinati in senso non decrescente. Si scriva un algoritmo basato sulla tecnica Divide et Impera che per decidere se A contiene oppure no elementi duplicati. Si analizzi la complessità di tempo dell'algoritmo proposto, giustificando le affermazioni fatte. L'algoritmo proposto potrebbe funzionare anche nel caso in cui A non è ordinato? Giustificare la risposta.

$\diamond$

6. *Esercizio:* Sia $A = A[1] \dots A[n]$ un vettore di numeri, non necessariamente ordinato. Si scriva un algoritmo basato sulla tecnica Divide et Impera che conti il numero di indici i per cui valga $A[i] \times A[i+1] > 0$. Si analizzi la complessità di tempo dell'algoritmo proposto, giustificando le affermazioni fatte.

$\diamond$

7. *Esercizio:* Sia $A = A[1] \dots A[n]$ un vettore ordinato in senso crescente che è stato shiftato k posizioni a sinistra. Ad esempio, il vettore $[15, 18, 28, 30, 35, 42, 1, 7]$ è un vettore ordinato che è stato shiftato $k = 2$ posizioni a sinistra, mentre il vettore $[30, 35, 42, 1, 7, 15, 18, 28]$ è un vettore ordinato che è stato shiftato $k = 5$ posizioni a sinistra.

   (a) Supponendo di avere A e k in input, dare un algoritmo che determina il minimo in A in tempo $O(1)$

   (b) Supponendo di avere *solo* il vettore A in input, dare un algoritmo che determina il minimo in A in tempo $O(\log n)$

$\diamond$

8. *Esercizio:* Sia $A = A[1] \dots A[n]$ un vettore ordinato in senso crescente che è stato shiftato k posizioni a sinistra. Avendo in input il solo vettore A, progettare un algoritmo basato sulla tecnica Divide et Impera che determini il valore k di cui sopra. Si analizzi la complessità di tempo dell'algoritmo proposto, giustificando le affermazioni fatte.

$\diamond$

9. *Esercizio:* Sia A un vettore di n interi. Si dice che A è continuo se per ogni $i = 1, 2, \dots, n-1$, vale che $|A[i+1] - A[i]| \leq 1$. Si dice zero del vettore un indice k tale che $A[k] = 0$. Dato un vettore A di $n \geq 2$ interi continuo tale che $A[1] \leq 0$ e $A[n] > 0$, provare che A ha almeno uno zero. Progettare un algoritmo basato sulla tecnica Divide et Impera che, dato un vettore A di $n \geq 2$ interi continuo e tale che $A[1] < 0$ e $A[n] > 0$, trovi uno zero in tempo $O(\log n)$.

$\diamond$

10. *Esercizio:* Sia $A = A[1] \dots A[n]$ un vettore di n interi, con $0 > A[1] < A[2] < \dots < A[n] > 0$. Progettare un algoritmo basato sulla tecnica Divide et Impera che calcoli il più piccolo valore di i per cui $A[i] < 0$ e $A[i+1] > 0$. Si analizzi la complessità di tempo dell'algoritmo proposto, giustificando le affermazioni fatte.

$\diamond$

11. *Esercizio:* Sia $A = A[1] \dots A[n]$ un vettore di n interi, che possono essere sia positivi che negativi. Si supponga che $A[1] \leq \dots \leq A[n]$. Progettare un algoritmo basato sulla tecnica Divide et Impera che conti il numero di elementi > 0 nel vettore A. Si analizzi la complessità di tempo dell'algoritmo proposto, giustificando le affermazioni fatte.

$\diamond$

12. *Esercizio:* Sia $A = A[1] \dots A[n]$ un vettore di n interi. Progettare un algoritmo basato sulla tecnica Divide et Impera che conti il numero di elementi $A[i]$ nel vettore A, $1 < i < n$, per cui valga sia che $A[i] = A[i+1]$ che $A[i] = A[i-1]$. Si analizzi la complessità di tempo dell'algoritmo proposto, giustificando le affermazioni fatte.

$\diamond$

13. *Esercizio:* Sia $A = A[1] \dots A[n]$ un vettore ordinato in senso crescente tale che $A[i] \in \{0, 1\}$, per ogni $i = 1, \dots, n$. Progettare un algoritmo basato sulla tecnica Divide et Impera che conti il numero di elementi $A[i]$ nel vettore A, per cui valga che $A[i] = 1$. Se ne analizzi la complessità. (Il massimo del punteggio lo si ottiene per un algoritmo di complessità $O(\log n)$).

$\diamond$

14. *Esercizio:* Siano A ed n numeri naturali positivi. Progettare un algoritmo basato sulla tecnica Divide et Impera che calcoli nA, utilizzando $O(\log n)$ addizioni.

$\diamond$

15. *Esercizio:* Sia Sia $A = A[1] \dots A[n]$ un vettore di n interi, tale che $A[1] < \dots < A[n]$. L'indice $i \in \{2, \dots, n\}$ è detto un *salto*, se vale che $A[i-1] + 1 < A[i]$. Progettare un algoritmo che determini se il generico vettore A ha un salto, e nell'ipotesi che lo abbia, determini attraverso la tecnica Divide et Impera almeno un valore di i per cui $A[i-1] + 1 < A[i]$. Si analizzi la complessità di tempo dell'algoritmo proposto, giustificando le affermazioni fatte.

$\diamond$

16. *Esercizio:* Sia data una matrice $n \times n$ di interi in cui gli elementi di ogni riga sono ordinati in senso decrescente, e anche gli elementi di ogni colonna sono ordinati in senso decrescente. Si progetti un algoritmo per determinare se un dato intero k é presente nella matrice o meno, e se ne analizzi la complessitá.

$\diamond$

17. *Esercizio:* Dato un vettore *ordinato* di interi $A[1 \dots n]$ ed un intero N, si progetti e si analizzi un algoritmo basato sulla tecnica Divide et Impera che, preso in input il vettore A ed il numero N determini se esistono o meno due indici i e j per cui $A[i] \times A[j] = N$.

$\diamond$

18. *Esercizio:* Si consideri il seguente problema: la Ditta ACME è stata quotata in borsa, ed il valore delle sue azioni sono state tabulate per tutto l'anno. Sia A il valore di una azione di ACME al primo Gennaio, e sia B il corrispondente valore al 31 Dicembre.

    (a) Se $A > B$, argomentare che c'è stato un giorno dell'anno in cui l'azione di ACME è stata quotata ad un valore inferiore al giorno precedente;

    (b) formalizzando opportunamente il problema in termini algoritmici (cioè definendo con precisione chi sono gli input e gli output al problema), sia dia un algoritmo di complessità logaritmica nella taglia dell'input che, sotto l'ipotesi che $A > B$, determini un giorno dell'anno in cui l'azione di ACME è stata quotata ad un valore inferiore al giorno precedente.

$\diamond$

19. *Esercizio:* Si consideri il seguente problema:

    Input: array `a=a[1]...a[n]` contenente valori interi ordinati in senso non decrescente (possono essere presenti valori duplicati), intero `x` .

    Output: l'indice (la posizione) dell'**dell'ultima** occorrenza di `x` in `a=a[1]...a[n]`, oppure `0` se il valore `x` non è presente in `a=a[1]...a[n]`.

    Si progetti e si analizzi un algoritmo basato sulla tecnica Divide et Impera che risolve il problema.

$\diamond$

20. *Esercizio:* Consideriamo il seguente problema algoritmico:

    Input: vettore `a=a[1]a[2]...a[n]`, ordinato in senso crescente, di n numeri tutti distinti tra di loro tranne che per una coppia, con $a[i] \in \{1, 2, \ldots, n-1\}$, per ogni $i = 1, \ldots, n$
    Output: l'unico intero $k \in \{1, 2, \ldots, n-1\}$ che compare due volte in `a`

$\diamond$

21. *Esercizio:* Si descriva l'algoritmo `QuickSelect (a,sinistra,k,destra)` visto a lezione (non è necessario descrivere in dettaglio l'algoritmo `Distribuzione(a, sinistra, pivot, destra)`, si derivi l'equazione di ricorrenza che descrive il tempo medio di esecuzione di `QuickSelect`.

$\diamond$

22. *Esercizio:* Consideriamo il seguente problema. Data una sequenza di numeri $a = a[0]a[1]\ldots a[n-1]$, un minimo relativo di a è un elemento di a che è *minore o uguale* dei suoi adiacenti (o di un **solo** adiacente, se stiamo considerando $a[0]$ o $a[n-1]$.

    Si derivi e si analizzi un algoritmo basato sulla tecnica Divide et Impera per il seguente problema algoritmico.

    Input: array $a = a[0]a[1]\ldots a[n-1]$
    Output: un (qualsivoglia) minimo relativo di a.

$\diamond$

23. *Esercizio:* Si ricordi l'equazione di ricorrenza che descrive il tempo medio di esecuzione $T(n)$ di `QuickSelect` (non è necessario derivarla) e si provi che $T(n) = O(n)$.

$\diamond$

24. *Esercizio:* Data una arbitraria sequenza di numeri $a = a[1]a[2]\ldots a[n]$, si provi che ogni algoritmo di ordinamento per $a = a[1]a[2]\ldots a[n]$ (basato su confronti) deve eseguire necessariamente $\Omega(n \log n)$ confronti nel caso peggiore.

$\diamond$

25. *Esercizio:* Sia $a = a[1]a[2]\ldots a[n]$ sequenza arbitraria di numeri, $n > 100$, di cui si sà che i primi 10 elementi e gli ultimi 20 elementi sono *correttamente* ordinati. Si provi che ogni algoritmo di ordinamento per $a = a[1]a[2]\ldots a[n]$ (basato su confronti) deve eseguire necessariamente $\Omega(n \log n)$ confronti nel caso peggiore.