

Algoritmi e Strutture Dati

Foglio 3

07/03/2025

Esercizio 1. Applicare il Teorema Principale per determinare limiti asintotici stretti per le seguenti ricorrenze:

1. $T(n) = 2T(n/4) + 1$
2. $T(n) = 2T(n/4) + \sqrt{n}$
3. $T(n) = 2T(n/4) + n$
4. $T(n) = 2T(n/4) + n^2$

Esercizio 2. Si consideri l'algoritmo CRUEL descritto di seguito. L'input è un vettore $A[p..q]$ di n interi, con n potenza di 2. Qual è il tempo di esecuzione di CRUEL?

Algorithm CRUEL(A, p, q)

```
1: if  $q - p + 1 > 1$  then
2:   CRUEL( $A, p, (q + p - 1)/2$ )
3:   CRUEL( $A, (q + p - 1)/2 + 1, q$ )
4:   CRUEL( $A, p, (q + p - 1)/2$ )
5:   MERGE-SORT( $A, p, q$ )
6: end if
```

Esercizio 3. Dato un vettore A di n interi distinti, un *minimo locale* di A è un intero x tale che:

- $x = A[i]$, per un certo $1 \leq i \leq n$;
- $A[i] < A[i - 1]$ e $A[i] < A[i + 1]$.

In parole, un minimo locale di A è un elemento di A che è più piccolo di entrambi gli elementi adiacenti (i due estremi del vettore, $A[1]$ e $A[n]$, hanno un solo elemento adiacente). Per esempio, il vettore $< 8, 9, 1, 2, 4, 3 >$ ha minimi locali 8, 1 e 3. Banalmente, è possibile trovare un minimo locale in $\Theta(n)$. Mostrare che si può fare di meglio.

Esercizio 4. Siano dati un vettore A di n interi ed un intero k . Progettare un algoritmo efficiente che trova, se esiste, una coppia di indici i, j con $i \neq j$ tali che $A[i] + A[j] = k$.

Esercizio 5. Un vettore A di dimensione n ha un *elemento preponderante* se più della metà dei suoi elementi sono uguali. Dato un vettore di dimensione n , si vuole progettare un algoritmo efficiente in grado di stabilire l'esistenza di un elemento preponderante e, nel caso, di trovarlo. Gli elementi del vettore non sono necessariamente numeri e quindi non siamo in grado di fare confronti del tipo $A[i] > A[j]$ (possiamo pensare gli elementi del vettore come dei file .gif, per esempio). Tuttavia, siamo in grado di dire se $A[i] = A[j]$ in $O(1)$ passi. Fornire un algoritmo che risolva tale problema in $O(n \log n)$ passi.

Esercizio 6. Utilizzate le seguenti idee per sviluppare un algoritmo non ricorsivo in tempo lineare per risolvere il problema del massimo sottoarray. Iniziate dall'estremità sinistra dell'array e procedete verso destra, registrando il massimo sottoarray trovato fino a quel punto. Conoscendo un massimo sottoarray di $A[1..j]$, aggiornate la soluzione considerando il massimo sottoarray che termina con l'indice $j + 1$.

sulla base della seguente osservazione: un massimo sottoarray di $A[1..j + 1]$ è un massimo sottoarray di $A[1..j]$ o un sottoarray della forma $A[i..j + 1]$, per un certo $1 \leq i \leq j + 1$. Determinate un massimo sottoarray della forma $A[i..j + 1]$ in tempo costante, supponendo di conoscere un massimo sottoarray che termina con l'indice j .

Esercizio 7. Qual è il tempo di esecuzione di QUICKSORT quando tutti gli elementi dell'array A hanno lo stesso valore?

Esercizio 8. Considerate la variante di MERGE-SORT in cui una delle due chiamate ricorsive è sostituita da una chiamata a QUICKSORT. Qual è il tempo di esecuzione di questo nuovo algoritmo (nel caso peggiore)?

Esercizio 1. Appicare il Teorema Principale per determinare limiti asintotici stretti per le seguenti ricorrenze:

1. $T(n) = \underline{2T(n/4)} + 1$
2. $T(n) = \underline{2T(n/4)} + \sqrt{n}$
3. $T(n) = \underline{2T(n/4)} + n$
4. $T(n) = \underline{2T(n/4)} + n^2$

Theorem 4.1 (Master theorem)

Let $a > 0$ and $b > 1$ be constants, and let $f(n)$ be a driving function that is defined and nonnegative on all sufficiently large reals. Define the recurrence $T(n)$ on $n \in \mathbb{N}$ by

$$T(n) = aT(n/b) + f(n), \quad (4.17)$$

where $aT(n/b)$ actually means $a'T(\lfloor n/b \rfloor) + a''T(\lceil n/b \rceil)$ for some constants $a' \geq 0$ and $a'' \geq 0$ satisfying $a = a' + a''$. Then the asymptotic behavior of $T(n)$ can be characterized as follows:

1. If there exists a constant $\epsilon > 0$ such that $f(n) = O(n^{\log_b a - \epsilon})$, then $T(n) = \Theta(n^{\log_b a})$.
2. If there exists a constant $k \geq 0$ such that $f(n) = \Theta(n^{\log_b a} \lg^k n)$, then $T(n) = \Theta(n^{\log_b a} \lg^{k+1} n)$.
3. If there exists a constant $\epsilon > 0$ such that $f(n) = \Omega(n^{\log_b a + \epsilon})$, and if $f(n)$ additionally satisfies the **regularity condition** $af(n/b) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n , then $T(n) = \Theta(f(n))$. ■

$$T_1(n) = 2T(n/4) + 1$$

$$1. \quad b=4, a=2 \Rightarrow n^{\log_4 2} = n^{1/2} = \sqrt{n}, \quad f(n) = 1$$

$$\circ \text{ oss. che } f(n) = O(n^{1/2 - \epsilon}) \quad \forall \epsilon > 0 : 0 < \epsilon < \frac{1}{2}; \text{ infatti}$$

$$1 \leq n^{1/2 - \epsilon} = n^\alpha, \quad 0 < \alpha < \frac{1}{2}$$

$$\Leftrightarrow \sqrt[4]{1} = 1 \leq n$$

$$\Leftrightarrow n \geq 1 \quad \checkmark$$

(1)

$$\Rightarrow T_1(n) = \Theta(n^{1/2})$$

$$2. f(n) = \sqrt{n} = n^{1/2}$$

- o poiché $f(n) = \Theta(n^{1/2})$, oss che per $K=0$ si ha

$$f(n) = \Theta(n^{1/2} \cdot \log^k(n)) = \Theta(n^{1/2})$$

$$\stackrel{(2)}{\Rightarrow} T_2(n) = \Theta(n^{1/2} \cdot \log n)$$

$$3. f(n) = n, a=2, b=4 \rightarrow \log_b(a) = \frac{1}{2}$$

- o si ha $f(n) = \Omega(n^{1/2+\varepsilon}) \forall \varepsilon > 0 : 0 < \varepsilon < \frac{1}{2}$; infatti

$$n^{1/2+\varepsilon} \leq n$$

$$\Leftrightarrow n^{\varepsilon - 1/2} \leq 1, -\frac{1}{2} < \varepsilon - \frac{1}{2} < 0$$

$$\Leftrightarrow n^{-\beta} \leq 1, \beta := -(\varepsilon - \frac{1}{2}) \Rightarrow \frac{1}{2} > \beta > 0$$

$$\Leftrightarrow \frac{1}{n^\beta} \leq 1 \quad \left[\frac{1}{n^{1/4}} \leq 1 \right]$$

chiaramente vero $\forall n \geq 1$ in quanto $0 < \beta < \frac{1}{2}$.

- o verifico cond. di regolarità

$$2f\left(\frac{n}{4}\right) \leq c f(n) \Leftrightarrow 2 \cdot \frac{n}{4} \leq cn$$

$$\Leftrightarrow \frac{n}{2} \leq cn$$

$$\Leftrightarrow \frac{1}{2} \leq c < 1$$

✓

(3)

$$\Rightarrow \text{segue che } T_3(n) = \Theta(n).$$

$$4. f(n) = n^2$$

- o potrebbe essere $f(n) = \Omega(n^{1/2+\varepsilon})$? Si, prendo $\varepsilon = \frac{1}{4} \Rightarrow n^2 = \Omega(n^{3/4})$

- o cond. di regolarità?

$$2 \cdot f\left(\frac{n}{4}\right) \leq c f(n) \Leftrightarrow 2 \cdot \frac{n^2}{16} \leq c \cdot n^2$$

$$\Leftrightarrow \frac{1}{8} \leq c$$

[vero $\forall n \in \mathbb{N}$]

$$\rightarrow \text{scalo } c \geq \frac{1}{8} \text{ e } n_0 = 1$$

$$\Rightarrow T(n) = \Theta(n^2)$$

$a \vee b \rightarrow b - a + 1$ elementi

Esercizio 2. Si consideri l'algoritmo CRUEL descritto di seguito. L'input è un vettore $A[p..q]$ di n interi, con n potenza di 2. Qual è il tempo di esecuzione di CRUEL?

Algorithm CRUEL(A, p, q)

```

1: if  $q - p + 1 > 1$  then  $n := q - p + 1$ 
2:   CRUEL( $A, p, (q + p - 1)/2$ )  $\Theta(n)$   $T(n/2)$ 
3:   CRUEL( $A, ((q + p - 1)/2 + 1) \circ q$ )  $T(n/2)$ 
4:   CRUEL( $A, p, (q + p - 1)/2$ )  $T(n/2)$ 
5:   MERGE-SORT( $A, p, q$ )  $\Theta(n \log n)$ 
6: end if

```

$$q - \frac{(q+p-1)}{2} + 1 = \frac{n}{2}$$

- come confronto la dimensione dell'input delle sottochiamate con quella attuale?

- $n := q - p + 1$

- $\underbrace{\frac{q+p-1}{2}}_{\text{dimensione input sottochiamata (1)}} - p + 1 = \frac{q+p-1 - 2p + 2}{2} = \frac{q-p+1}{2} = \frac{n}{2}$

dimensione input sottochiamata (1)

- le altre sono analoghe ...

- scriviamo eq. di ricorrenza

$$T(n) = 3T(n/2) + \Theta(n \log n)$$

- quindi $n^{\log_2 3} \approx 1.585$, il che implica

$$n < n^{\log_2 3} < n^2$$

- OSS. ora che

$$\begin{aligned} &\xrightarrow{\text{sse}} n \log n < n^{\log_2 3} \log n < n^{\log_2 3 - \varepsilon} \quad \forall n > 1 \\ &\log n < n^{-\varepsilon} \rightarrow -\varepsilon > 0 \rightarrow \varepsilon < 0 \end{aligned}$$

NOTA

$$\begin{aligned} \lim_{n \rightarrow \infty} \frac{f(n)}{n^{\log_2 3 - \varepsilon}} &= \lim_{n \rightarrow \infty} \frac{n \log n}{n^{\log_2 3 - \varepsilon}} = \lim_{n \rightarrow \infty} \frac{\log n}{n^{\alpha - \varepsilon}}, \quad \alpha := \log_2 3 - 1 \\ &= \begin{cases} 0 & \text{se } \alpha - \varepsilon > 0 \Leftrightarrow \varepsilon < \alpha \\ +\infty & \text{altrimenti} \end{cases} \end{aligned}$$

Quindi basta prendere $0 < \varepsilon < \alpha =: \log_2 3 - 1$, che esiste poiché $\frac{1}{2} < \alpha < 1$; ad esempio

$$\varepsilon = \alpha - \frac{1}{2} \approx \log_2(3) - 1 - \frac{1}{2} = 0.085$$

verifica

$$f(n) = O(n^{\log_2 3 - \varepsilon}) \Rightarrow f(n) = O(n^{\log_2 3 - \varepsilon}) \quad \checkmark$$

\Rightarrow segue che $T(n) = \Theta(n^{\log_2 3})$.

Esercizio 3. Dato un vettore A di n interi distinti, un *minimo locale* di A è un intero x tale che:

- $x = A[i]$, per un certo $1 \leq i \leq n$;
- $A[i] < A[i-1]$ e $A[i] < A[i+1]$.

In parole, un minimo locale di A è un elemento di A che è più piccolo di entrambi gli elementi adiacenti (i due estremi del vettore, $A[1]$ e $A[n]$, hanno un solo elemento adiacente). Per esempio, il vettore $< 8, 9, 1, 2, 4, 3 >$ ha minimi locali 8, 1 e 3. Banalmente, è possibile trovare un minimo locale in $\Theta(n)$. Mostrare che si può fare di meglio.

1	2	3	4	5	6
4	5	7	2	8	9
$i : A[i] < A[i-1] \in A[i] < A[i+1]$					(1)

o scelgo un i

o se verifica (1) \rightarrow trovato

o altrimenti, $A[i] > A[i-1] \stackrel{(a)}{\text{e}} A[i] > A[i+1] \stackrel{(b)}{\text{e}}$

o se (a) è vera, esiste un minimo locale a sx

Dim: per assurdo, supp. che non esista un minimo locale a sx di $A[i]$, nel caso (a).

Quindi è vero che $j = i-1 \quad A[j] > A[i]$

$\forall j < i \quad A[j] > A[j+1] \stackrel{(a)}{\text{e}} A[j] > A[j-1]$ [negazione (1)]

ma per (a), $i-1 < i$ verifica $A[i-1] < A[i]$ quindi necessariamente

$A[i-1] > A[i-2] \Rightarrow A[i-2] > A[i-3] \Rightarrow \dots \Rightarrow A[2] > A[1]$

ma allora esiste il minimo locale $A[1]$

□

o se (b) è vera, esiste un minimo locale a dx

Dim. analoga al caso (a)

- pseudocodice;

LOCAL-MIN(A, l, r) :

if $l > r$
raise "error"

if $l = r$
return l

$q \leftarrow \left\lfloor \frac{l+r}{2} \right\rfloor$

if $q = 1$ $\&\&$ $A[q] < A[q+1]$
return 1
else if $q = n$ $\&\&$ $A[q] < A[q-1]$
return n

// arrivo a sx

// arrivo a dx

if $A[q] < A[q+1]$ $\&\&$ $A[q] < A[q-1]$
return q

else if $A[q] > A[q-1]$

return LOCAL-MIN(A, l, q-1)

else

return LOCAL-MIN(A, q+1, r)

- complessità

$$T(n) = T\left(\frac{n}{2}\right) + \Theta(1)$$

$$\circ n^{\log_2 1} = n^0 = 1 \rightarrow f(n) = \Theta(1) = \Theta(n^0 \cdot \log^0 n)$$

$$\stackrel{(2)}{\Rightarrow} T(n) = \Theta(\log n)$$

$$K = \emptyset$$

- o correttezza

- o CASO BASE

✓

Se $n=1$, il minimo locale è l'unico elemento nell'array \rightarrow ritorno quell'indice

- o PASSO IND.: Algo funziona fino a un certo $\frac{n}{2} \in \mathbb{N}$

Supponiamo di avere un input di dimensione n .

Se l'elemento in pos. q è $1 \leq n$ + rispetto condiz., ritorno q . ✓

Altrimenti, sfrutto (a) e (b) per richiamare la procedura su due porzioni diverse dell'array iniziale; oss. che

$$\begin{cases} (q-1) - l + 1 = q - l \leq \frac{r+l}{2} - l \leq \frac{r-l}{2} < n \\ r - (q+1) + 1 = r - q \leq r - l = n \end{cases}$$

In entrambi i casi posso applicare l'hyp induttiva

$$r-l+1 = n+1 \Rightarrow r-l = n$$

Quindi per ipotesi induttiva, la sottochiamata selezionata ritorna il valore cercato.

■

Esercizio 4. Siano dati un vettore A di n interi ed un intero k . Progettare un algoritmo efficiente che trova, se esiste, una coppia di indici i, j con $i \neq j$ tali che $A[i] + A[j] = k$.

o IDEA: ordino A e applico ricerca binaria all'elemento $k - A[i]$

FIND-PAIR(A, k) :

// Ordino A con MERGE-SORT $\Theta(n \log n)$

for $i = 1$ to n

$J \leftarrow \text{BINARY-SEARCH}(A, 1, n, \underbrace{k - A[i]}_{A[j]})$ $O(\log n)$

if $J \neq -1$ $\&$ $J \neq i$
return (i, J)

return $(-1, -1)$ // nessuna coppia trovata

o costo: $T(n) = \Theta(n \log n)$.

$\underbrace{\Theta(n \log n)}_{\text{MERGE-SORT}} + \underbrace{O(n \log n)}_{\text{CICLO FOR}} \rightarrow \Theta(n \log n)$

Esercizio 5. Un vettore A di dimensione n ha un *elemento preponderante* se più della metà dei suoi elementi sono uguali. Dato un vettore di dimensione n , si vuole progettare un algoritmo efficiente in grado di stabilire l'esistenza di un elemento preponderante e, nel caso, di trovarlo. Gli elementi del vettore non sono necessariamente numeri e quindi non siamo in grado di fare confronti del tipo $A[i] > A[j]$ (possiamo pensare gli elementi del vettore come dei file .gif, per esempio). Tuttavia, siamo in grado di dire se $A[i] = A[j]$ in $O(1)$ passi. Fornire un algoritmo che risolva tale problema in $O(n \log n)$ passi.

- Concetto chiave: un elemento x è preponderante sse $\#x \geq \frac{n}{2} + 1$, dove n è la dimensione di A .
- Idea: uso Divide et Impera. Spezzo il problema in due sottoproblemi di dimensione $\left\lceil \frac{n}{2} \right\rceil \leq \left\lfloor \frac{n}{2} \right\rfloor$:



- Oss. che se x preponderante in A , allora x prep. in A_1 o A_2

Dim: per assurdo, supponiamo che x non sia prep. in A_1 e A_2 ; allora, vuol dire che

$$\begin{cases} \#x_{A_1} \leq \frac{1}{2} \cdot \left\lceil \frac{n}{2} \right\rceil \\ \#x_{A_2} \leq \frac{1}{2} \cdot \left\lfloor \frac{n}{2} \right\rfloor \end{cases} \quad \xrightarrow{\text{se fosse prep. in } A_1, \text{ ci sarebbero almeno}} \quad \frac{1}{2} \cdot \left\lceil \frac{n}{2} \right\rceil + 1$$

occorrente

il che implica

$$\#x_{A_1} + \#x_{A_2} \leq \frac{1}{2} \left(\left\lceil \frac{n}{2} \right\rceil + \left\lfloor \frac{n}{2} \right\rfloor \right) = \frac{1}{2} \cdot n = \frac{n}{2}$$

assurdo, in quanto per hp x è preponderante in A . ■

- quindi posso procedere controllando le occorrenze degli elem. preponderanti dei due sottovettori (se esistono).

- pseudocodice

FIND-P-ELEM(A, l, r) : // ritorna coppia (elem, count)

if $r < l$ raise "error"

$$q \leftarrow \lceil \frac{r+l}{2} \rceil$$

count \leftarrow COUNT(A, A[q])

// $\Theta(n)$

if count $\geq \frac{A.length}{2} + 1$

return (A[q], count)

if $r = l$

return (NULL, \emptyset)

// 1 elemento

lhs \leftarrow FIND-P-ELEM(A, l, q-1)

rhs \leftarrow FIND-P-ELEM(A, q+1, r)

if lhs.Count $\neq \emptyset$

return lhs

// al piu' restituisco
// (NULL, \emptyset) se non
// esiste.

else

return rhs

// altrimenti e' indifferente
// prendere lhs o rhs

- Costo : $T(n) = 2T\left(\frac{n}{2}\right) + \Theta(n) \Rightarrow T(n) = \Theta(n \log n)$.

Esercizio 7. Qual è il tempo di esecuzione di QUICKSORT quando tutti gli elementi dell'array A hanno lo stesso valore?



- se tutti gli elementi sono uguali, PARTITION crea una partizione completamente sbilanciata :

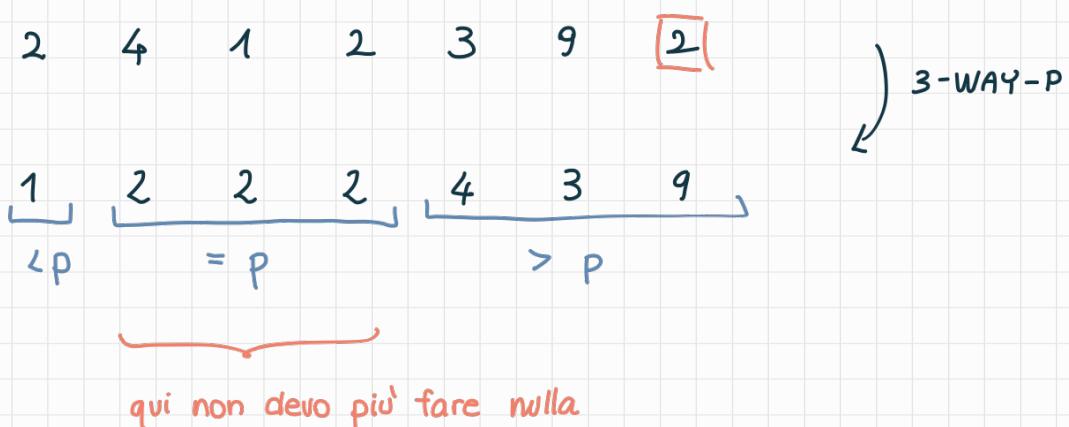
$$T(n) = T(n-1) + T(\emptyset) + \Theta(n)$$

$$\Rightarrow T(n) = \Theta(n^2)$$



- come possiamo risolvere ? **3-WAY-PARTITION** ...

L'idea è di separare in una zona centrale tutti gli elementi uguali al pivot. In questo modo, richiamiamo la procedura soltanto sugli elementi minori e maggiori del pivot !



- se gli elementi sono tutti uguali allora

$$T(n) = 2T(\emptyset) + \Theta(n) = \Theta(n)$$

!

Esercizio 8. Considerate la variante di MERGE-SORT in cui una delle due chiamate ricorsive è sostituita da una chiamata a QUICKSORT. Qual è il tempo di esecuzione di questo nuovo algoritmo (nel caso peggiore)?

- nel caso peggiore, QUICKSORT è $\Theta(n^2)$; quindi

$$T(n) = \underbrace{T\left(\frac{n}{2}\right)}_{\text{sottochiamata MERGE-SORT}} + \underbrace{\Theta\left(\left(\frac{n}{2}\right)^2\right)}_{\text{sottochiam. QUICK-SORT}} + \underbrace{\Theta(n)}_{\text{MERGE}}$$

$$= T\left(\frac{n}{2}\right) + \Theta(n^2)$$

- oss. che $n^{\log_2(1)} = n^0 = 1$; inoltre $f(n) = n^2$, quindi

$$f(n) = \Omega(n^{0+\varepsilon}) \quad \text{per } \varepsilon = 1$$

- inoltre

$$\left(\frac{n}{2}\right)^2 \leq cn^2 \Leftrightarrow \frac{1}{4} \leq c \quad \forall n \in \mathbb{N}$$

→ scelgo $c = \frac{1}{4}$, $n_0 = 1$ per la condiz. di regolarità.

$$\Rightarrow T(n) = \Theta(n^2).$$