

Algoritmi e Strutture Dati

Foglio 4
14/03/2025

Esercizio 1. Disegnate l'albero di decisione per l'ordinamento di 3 interi corrispondente all'algoritmo MERGE-SORT.

Esercizio 2. Utilizzando un opportuno albero di decisione e l'idea vista a lezione, dimostrate che il problema della ricerca di un valore target in un array ordinato di dimensione n richiede $\Omega(\log n)$ confronti nel caso peggiore. Che conclusione possiamo trarre su BINARY-SEARCH?

Esercizio 3. Supponete che l'ultimo ciclo di COUNTING-SORT visto a lezione sia rimpiazzato da
`for j = 1 to A.length`

Dimostrate che il nuovo algoritmo opera ancora correttamente. Il nuovo algoritmo è stabile?

Esercizio 4. Descrivete un algoritmo che, dati n numeri interi in $[0, k]$, svolga un'analisi preliminare del suo input e poi risponda nel tempo $O(1)$ a qualsiasi domanda su quanti degli n interi ricadono nell'intervallo $[a, b]$. L'algoritmo deve impiegare un tempo $O(n + k)$ per l'analisi preliminare.

Esercizio 5. È possibile ordinare n interi in $[0, n^3 - 1]$ in tempo $O(n)$?

Esercizio 6. Sia A un array di $n \geq 2$ interi contenente gli interi tra 1 ed $n - 1$, uno dei quali appare due volte. Descrivete un algoritmo che trovi l'elemento ripetuto in tempo $O(n)$.

Esercizio 7. Fornite un algoritmo che, dato un array A di n interi, restituisca un intero k che *non* può essere ottenuto come somma di due elementi di A , cioè un intero k tale che non esistono $i \neq j$ tali che $k = A[i] + A[j]$. L'algoritmo deve impiegare $O(n)$ passi.

Esercizio 8. Sia A un array di n interi. Un *salto* in A è un indice i ($1 \leq i < n$) tale che $A[i+1] - A[i] \geq 2$. Osservate che, se $n \geq 2$ e $A[n] - A[1] \geq n$, l'array A ha almeno un salto. Progettate un algoritmo che, dato un array A di dimensione $n \geq 2$ e tale che $A[n] - A[1] \geq n$, trovi un salto in tempo $O(\log n)$.

Esercizio 1. Disegnate l'albero di decisione per l'ordinamento di 3 interi corrispondente all'algoritmo MERGE-SORT.

MERGE-SORT (A, l, r) :

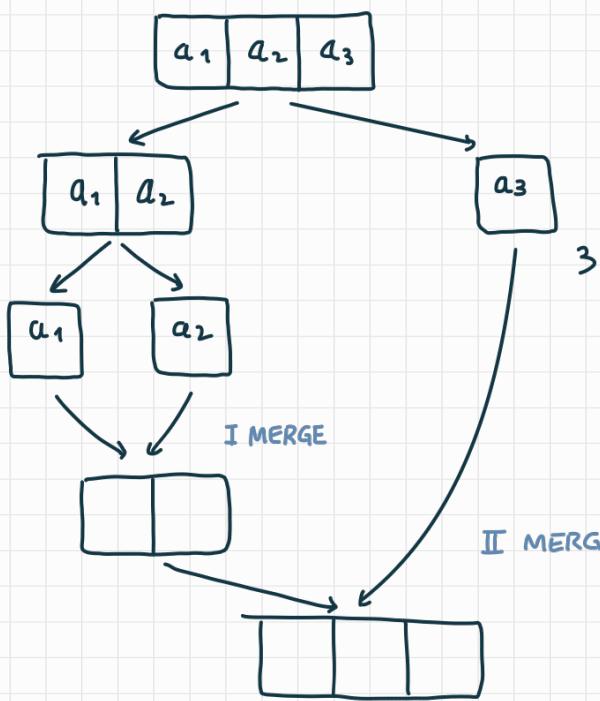
$$q = \left\lfloor \frac{l+r}{2} \right\rfloor$$

$$A = (a_1, a_2, a_3)$$

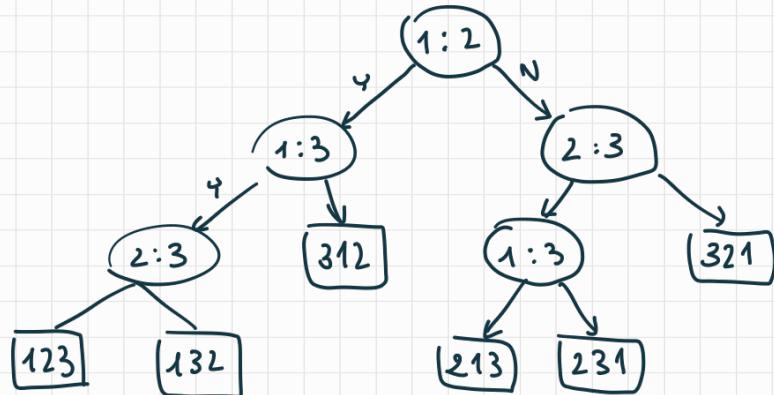
$$\text{// } q = \frac{1+3}{2} = 2$$

MERGE-SORT (A, l, q)
 MERGE-SORT ($A, q+1, r$)
 MERGE (A, l, q, r)

// MERGE-SORT ($A, 1, 2$)
 // MERGE-SORT ($A, 3, 3$)
 // MERGE ($A, 1, 2, 3$)



...
 if $L[i] \leq R[j]$
 ...



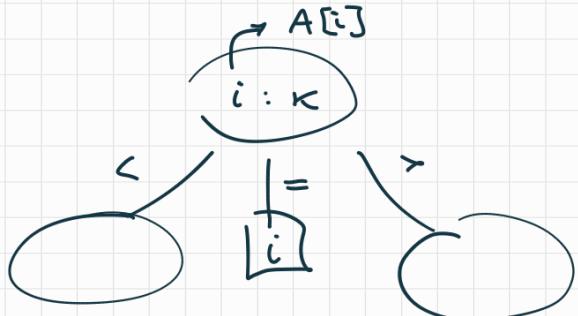
Esercizio 2. Utilizzando un opportuno albero di decisione e l'idea vista a lezione, dimostrate che il problema della ricerca di un valore target in un array ordinato di dimensione n richiede $\Omega(\log n)$ confronti nel caso peggiore. Che conclusione possiamo trarre su BINARY-SEARCH?

- Nella RICERCA BINARIA, dove avvengono i confronti?

```

...
 $q \leftarrow \left\lfloor \frac{l+r}{2} \right\rfloor$ 
if  $A[q] = \text{key}$ 
...
else if  $A[q] < \text{key}$ 
...
else

```



Ogni nodo interno dell'albero di decisione è associato a un indice sul quale faccio il confronto con la chiave.

Le possibili foglie sono gli indici $i \in [1..n]$ oppure il valore L nel caso in cui la chiave non appaia in A.

$\Rightarrow n+1$ risultati che mi interessano.

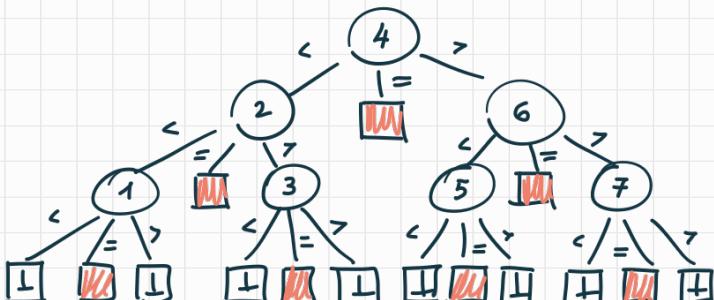
- Quanti nodi? Ogni nodo ha 3 figli, ma soltanto da 2 genero altri figli (confronti).
 \Rightarrow al livello j ($j \geq 0$) ci sono $N_j = 2^j + 2^{j-1} = 2^{j-1}(2+1) = 3 \cdot 2^{j-1}$ nodi
- Impongo numero di foglie $f \geq n+1$. Un albero di questo tipo ha una foglia per ogni nodo interno;

Inoltre ci sono 3 foglie per ogni nodo del penultimo livello (due L e una con indice trovato) quindi

$$3 \cdot N_{h-1} = 3^2 \cdot 2^{h-2}$$

Imponendo

$$3^2 \cdot 2^{h-2} \geq n+1 \Leftrightarrow h-2 \geq \log\left(\frac{n+1}{3^2}\right) \Leftrightarrow h = \Omega(\log n).$$



0	$N_0 = 1$
1	$N_1 = 3$
2	$N_2 = 6$
3	$N_3 = 12$

Esercizio 3. Supponete che l'ultimo ciclo di COUNTING-SORT visto a lezione sia rimpiazzato da

for $j = 1$ to $A.length$

Dimostrate che il nuovo algoritmo opera ancora correttamente. Il nuovo algoritmo è stabile?

o Algoritmo

COUNTING-SORT(A) :

for $i = 1$ to K
 $C[i] = \emptyset$

for $j = 1$ to n
 $C[A[j]] += 1$

for $i = 2$ to K
 $C[i] = C[i] + C[i-1]$ // $C[i]$ # elem di $A \leq i$

(*) for $j = 1$ to n
 $B[C[A[j]]] = A[j]$
 $C[A[j]] = C[A[j]] - 1$

return B

o Correttezza

- Utilizziamo come invarianto di (*) :

"All'inizio di ogni ciclo, ogni elemento $C[i]$, $i \in [0..K]$ indica la posizione finale in B dell'elemento " i ".
Inoltre, i $j-1$ elementi $A[1..j-1]$ sono già nella posiz. corretta in B ".

- Inizializzazione : $j = 1$

Per quanto visto a lez., nella prima iter., $C[i]$ contiene gli elem. $\leq i$ presenti; quindi ogni $C[i]$ indica correttamente la posiz. finale dell'elemento i . Non ho ancora posizionato elementi di A . ✓

- Conservazione : passo $1 < j < n$

La prima istruzione scrive l'elemento $A[j]$ in posizione $C[A[j]]$ nell'array B . Per ipotesi, $C[A[j]]$ è la posizione finale e corretta.

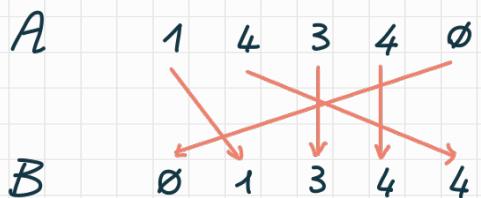
Successivamente decremento di 1 $C[A[j]]$: se ci fosse un duplicato verrebbe posizionato, correttamente, a sx dell'elemento appena posizionato.

Incremento j di 1; quindi ho ordinato $A[1..j-1]$ e ho ristabilito l'invariante in C decrementando di 1 il count del valore appena posizionato.

- Terminazione : $j: n+1$

In uscita dal ciclo, $j=n+1$. Quindi ho ordinato $A[1..n]$ in B .

- E' STABILE ? NO !



C

\emptyset	1	2	2	3	4
1	2	2	3	5	

Esercizio 4. Descrivete un algoritmo che, dati n numeri interi in $[0, k]$, svolga un'analisi preliminare del suo input e poi risponda nel tempo $O(1)$ a qualsiasi domanda su quanti degli n interi ricadono nell'intervallo $[a, b]$. L'algoritmo deve impiegare un tempo $O(n + k)$ per l'analisi preliminare.

- Idea: l'array C di counting-sort ha tutto quello che ci serve ...

- Ogni cella indica $\#$ naturali $\leq i$

$$\rightarrow \begin{cases} C[b] : \#n \leq b \\ C[a] : \#n \leq a \end{cases}$$

- Oss: se considero $C[b] - C[a]$, sto considerando quanti numeri in $[a, b]$.

\rightarrow prendo $C[a-1]$

$\rightarrow \#$ elem di A in $[a, b] = C[b] - C[a-1]$, con $C[-1] = \emptyset$.



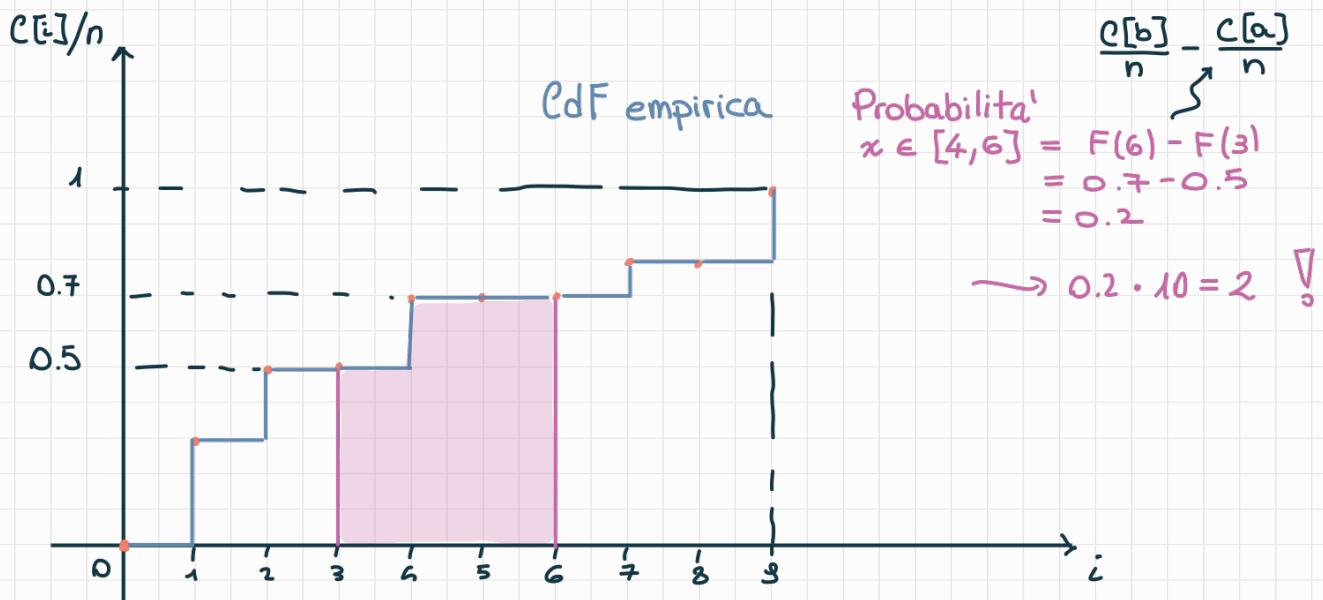
NOTA : $C[i]$ ci dice quanti elementi $\leq i$ e cresce fino al valore massimo, n .

Potrei graficare i punti $(i, \frac{C[i]}{n}) \dots$

$$A : [1, 4, 7, 9, 1, 1, 2, 2, 9, 4]$$

$$C : [\emptyset, 3, 5, 5, 7, 7, 7, 8, 8, 10]$$

0 1 2 3 4 5 6 7 8 9



$$\underline{n} \cdot \underline{n} \cdot \underline{n} \rightarrow \log_n(n^3) = 3$$

Esercizio 5. È possibile ordinare n interi in $[0, n^3 - 1]$ in tempo $O(n)$?

- o OSS: $n^3 - 1$ è il max. numero rappresentabile con 3 cifre in base n .
 ↳ COUNTING-SORT non applicabile ... $O(n^3)$
 ↳ RADIX-SORT ∇
- o converto in base n , poi applico RADIX-SORT sulle 3 cifre.
 $T(n) = \Theta(3 \cdot (n+n)) = \Theta(n)$
 ↓
 k rappresenta i valori $0..k$ che può assumere la singola cifra; n in questo caso.

Esercizio 6. Sia A un array di $n \geq 2$ interi contenente gli interi tra 1 ed $n - 1$, uno dei quali appare due volte. Descrivete un algoritmo che trovi l'elemento ripetuto in tempo $O(n)$.

- o ordino con COUNTING-SORT $\rightarrow \Theta(n+n) = \Theta(n)$
 - o scorro l'array finché due posizioni $i, i+1$ contengono lo stesso elemento $\rightarrow O(n)$
- $$\Rightarrow T(n) = \Theta(n) + O(n) = \Theta(n)$$

Esercizio 7. Fornite un algoritmo che, dato un array A di n interi, restituisca un intero k che *non* può essere ottenuto come somma di due elementi di A , cioè un intero k tale che non esistono $i \neq j$ tali che $k = A[i] + A[j]$. L'algoritmo deve impiegare $O(n)$ passi.

- o Idea : posso scegliere K a piacere ... perché non prendere
 - $K = 2 \cdot \max(A) + 1$
 - $K = \min(A) - 1$
 - ecc ...
- o $\min(A) \leq \max(A) \rightarrow \Theta(n)$
 $\Rightarrow T(n) = \Theta(n)$
- o perché funziona ?
 - sto sommando due interi $A[i], A[j]$ con $i \neq j$. Per entrambi

$$\begin{cases} A[i] \leq \max(A) \\ A[j] \leq \max(A) \end{cases} \Rightarrow A[i] + A[j] \leq 2\max(A) < 2\max(A) + 1 .$$

Esercizio 8. Sia A un array di n interi. Un salto in A è un indice i ($1 \leq i < n$) tale che $A[i+1] - A[i] \geq 2$. Osservate che, se $n \geq 2$ e $A[n] - A[1] \geq n$, l'array A ha almeno un salto. Progettate un algoritmo che, dato un array A di dimensione $n \geq 2$ e tale che $A[n] - A[1] \geq n$, trovi un salto in tempo $O(\log n)$.

- Idea: sfrutto

"se $n \geq 2 \Leftrightarrow A[n] - A[1] \geq n$ allora esiste almeno un salto"

per un algo. divide et impera.

- se spezzo in 2 parti A tramite $q = \left\lfloor \frac{l+r}{2} \right\rfloor$, controllo la condizione sui sottoarray: cerco in uno tra i due che la soddisfi (c'e' sicuramente poiché il salto e' sta a sx e a dx in quanto esiste)

SALTO (A, l, r):

```
if  $r-l = 1$  dd  $A[r]-A[l] \geq 2$ 
    return  $l$ 
else if  $l-r = 1$ 
    return  $-1$ 
```

$$q = \lfloor (l+r)/2 \rfloor$$

if $A[q] - A[1] \geq (q-l+1)$

return SALTO (A, l, q)

else

return SALTO (A, q, r)

// due elementi: se arrivo al caso base
// non serve fare controlli aggiuntivi ...
// non entreremo mai qui ... perché?

$$n = r-l+1$$

// oss. SALTO () viene chiamata
// con input di dimensione ≥ 2 :
// se arrivo qui, $r-l+1 \geq 3$ altrimenti
// caso base sopra;
// quindi

$$\begin{aligned} r+l-1 + 1 - l &\geq 3 + l - 1 \\ q &\geq 2l + 1 \end{aligned}$$

$$\left\{ \begin{array}{l} q-l+1 \geq 2l-l+1+1 \geq l+2 \geq 2 \\ r-q+1 = (r-l+1)-q+1+l-1 \\ \geq 3-q+l \end{array} \right.$$

$$= 4 - 1 - q + l = 4 - (q - l + 1)$$

$$\geq 4 - 2 = 2$$

- complessita':

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