Esercitazione
12 novembre 2024

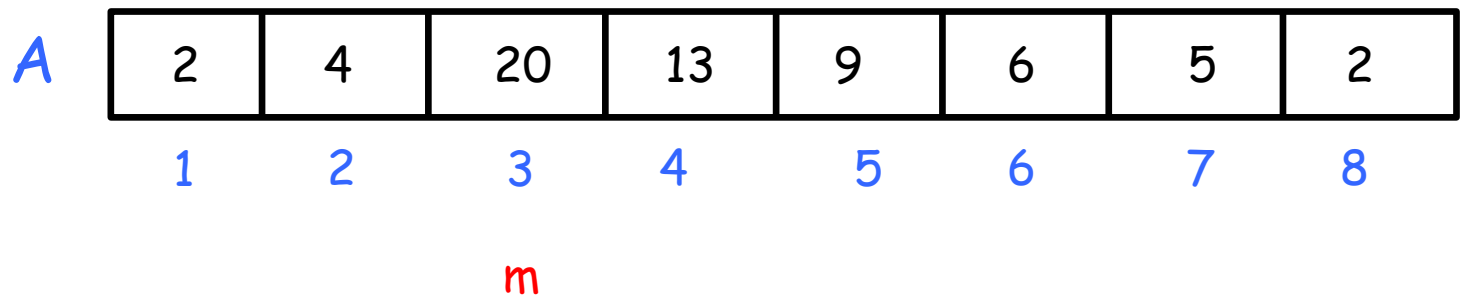
Esercizio Un array A di n elementi è detto *unimodale* se consiste di una sequenza crescente seguita da una sequenza decrescente o più precisamente se esiste un indice $m \in \{1, 2, \ldots, n\}$ tale che:

- $A[i] < A[i+1]$, per ogni $1 \leq i < m$, e
- $A[i] > A[i+1]$, per ogni $m \leq i < n$.

In particolare $A[m]$ è il massimo elemento ed è l'unico che è circondato da due elementi più piccoli ($A[m-1]$ e $A[m+1]$).

(a) Si progetti un algorimo con complessità temporale $o(n)$ che, dato un array unimodale A, restituisce l'indice dell'elemento massimo.

(b) Si progetti un algorimo con complessità temporale $o(n \log n)$ che ordina (in ordine crescente) un array unimodale A.



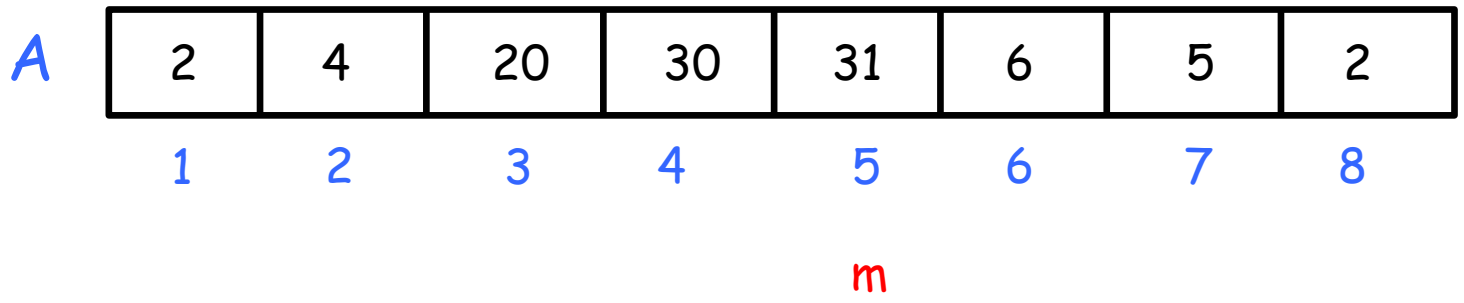
Esercizio Un array A di n elementi è detto *unimodale* se consiste di una sequenza crescente seguita da una sequenza decrescente o più precisamente se esiste un indice $m \in \{1, 2, \ldots, n\}$ tale che:

- $A[i] < A[i+1]$, per ogni $1 \leq i < m$, e
- $A[i] > A[i+1]$, per ogni $m \leq i < n$.

In particolare $A[m]$ è il massimo elemento ed è l'unico che è circondato da due elementi più piccoli ($A[m-1]$ e $A[m+1]$).

(a) Si progetti un algorimo con complessità temporale $o(n)$ che, dato un array unimodale A, restituisce l'indice dell'elemento massimo.

(b) Si progetti un algorimo con complessità temporale $o(n \log n)$ che ordina (in ordine crescente) un array unimodale A.



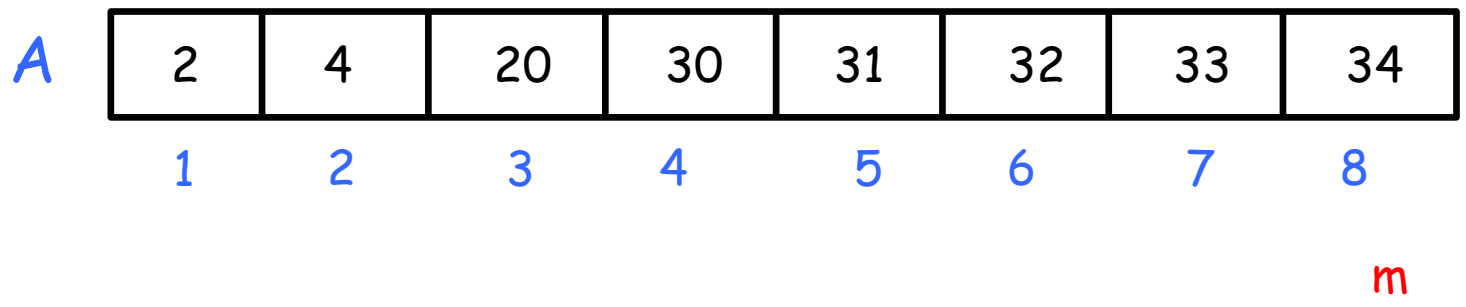
Esercizio Un array A di n elementi è detto *unimodale* se consiste di una sequenza crescente seguita da una sequenza decrescente o più precisamente se esiste un indice $m \in \{1, 2, \ldots, n\}$ tale che:

- $A[i] < A[i+1]$, per ogni $1 \leq i < m$, e
- $A[i] > A[i+1]$, per ogni $m \leq i < n$.

In particolare $A[m]$ è il massimo elemento ed è l'unico che è circondato da due elementi più piccoli ($A[m-1]$ e $A[m+1]$).

(a) Si progetti un algorimo con complessità temporale $o(n)$ che, dato un array unimodale A, restituisce l'indice dell'elemento massimo.

(b) Si progetti un algorimo con complessità temporale $o(n \log n)$ che ordina (in ordine crescente) un array unimodale A.



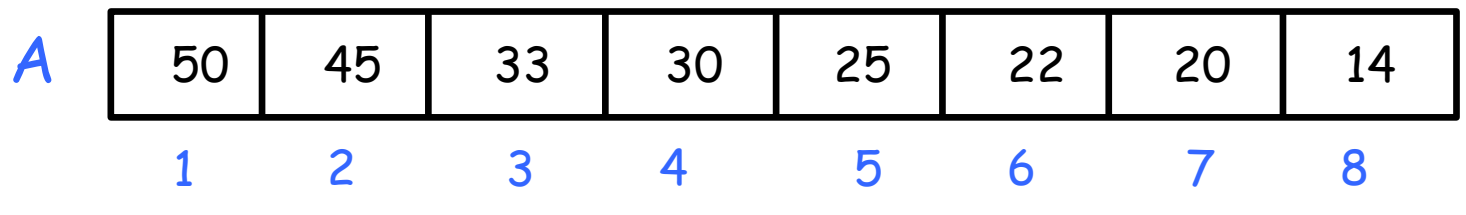
Esercizio Un array A di n elementi è detto *unimodale* se consiste di una sequenza crescente seguita da una sequenza decrescente o più precisamente se esiste un indice $m \in \{1, 2, \ldots, n\}$ tale che:

- $A[i] < A[i+1]$, per ogni $1 \leq i < m$, e
- $A[i] > A[i+1]$, per ogni $m \leq i < n$.

In particolare $A[m]$ è il massimo elemento ed è l'unico che è circondato da due elementi più piccoli ($A[m-1]$ e $A[m+1]$).

(a) Si progetti un algorimo con complessità temporale $o(n)$ che, dato un array unimodale A, restituisce l'indice dell'elemento massimo.

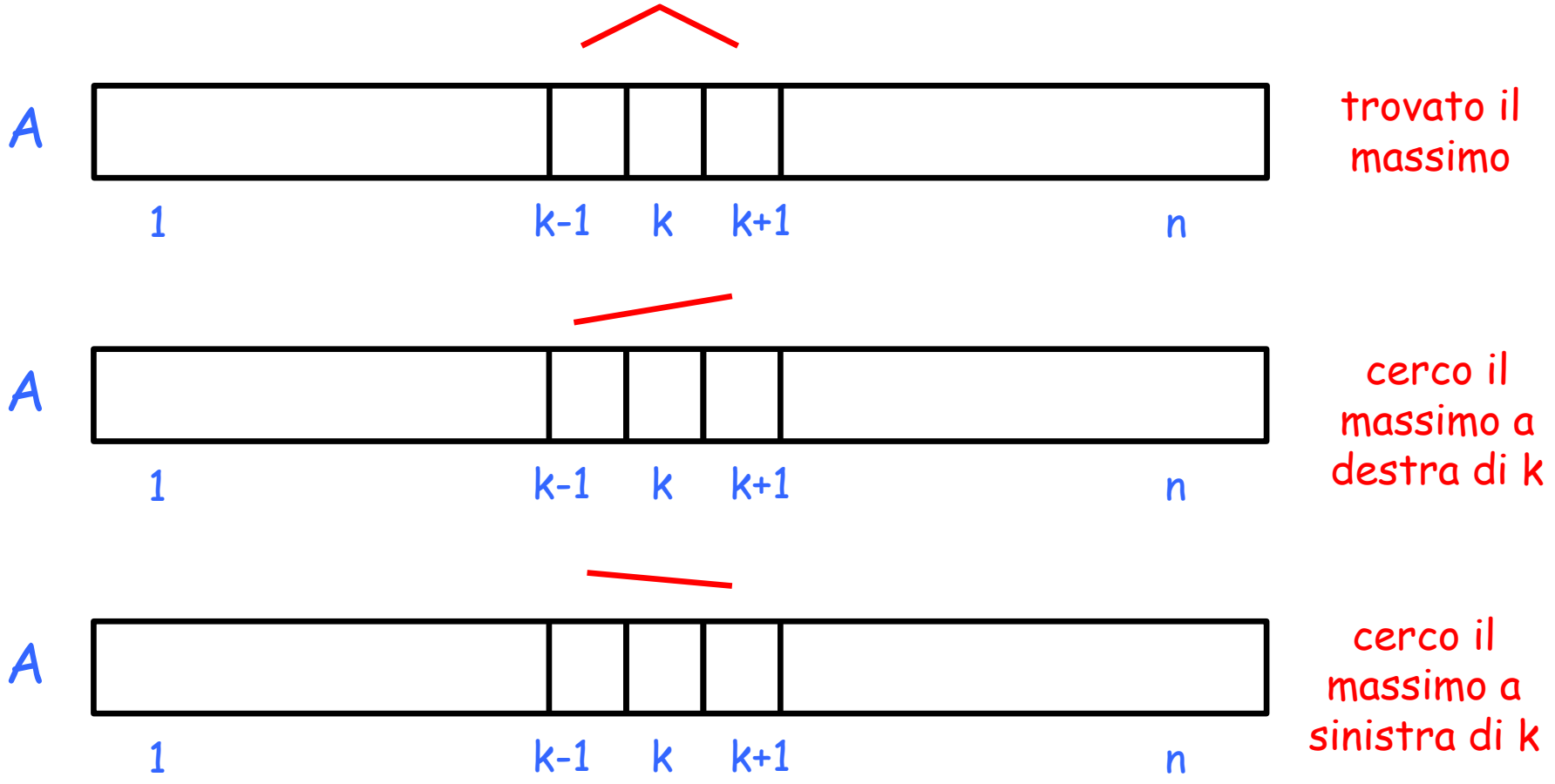
(b) Si progetti un algorimo con complessità temporale $o(n \log n)$ che ordina (in ordine crescente) un array unimodale A.



idea: uso l'approccio delle ricerca binaria.

punto (a)

osservazione cruciale: se guardo un elemento in posizione k e i due elementi adiacenti in posizione k-1 e k+1, so capire in tempo constante se A[k] è il massimo o se devo cercare il massimo a sinistra o a destra di k.



punto (a)

Algorithm 2: `MaxRic`(A, i, j)

```
if i > j then
    return −1
m = ⌊(i+j)/2⌋ ;
if A[m] > A[m − 1] e A[m] > A[m + 1] then
    return m
if  A[m] < A[m + 1] then
    return MaxRic(A, m + 1, j)
else
    return MaxRic(A, i, m − 1)
```

Algorithm 1: `Max`(A)

```
n = size of A ;
if A[1] > A[2] then
    return 1
if A[n] > A[n − 1] then
    return n
return MaxRic(A, 2, n − 1)
```

correttezza?

Segue dalle argomentazioni su come restringere la ricerca del massimo.

complessità?

O(log n)

idea: punto (b)

Sia m la posizione m del massimo. Le due "metà" A[1:m] e A[m+1:n] sono già ordinate. Allora uso il Merge per fonderle velocemente (attenzione che A[m+1:n] è ordinata in ordine decrescente).

Algorithm 3: `OrdinaUnimodale`(A)

```
n = size of A ;
m = Max(A);
if m ≠ −1 and m ≠ n then
    Inverti(A, m, n);
    if m ≠ 1 then
        Merge(A, 1, m − 1, n);
```

correttezza?
Ovvia. (Segue dalla correttezza del Merge)

complessità?

O(n)

Merge(A,i,k,f)
fonde A[i:k] e A[k+1,f] (che si aspetta ordinate in modo crescente) e mette la sequenza ordinata in A[i:f]

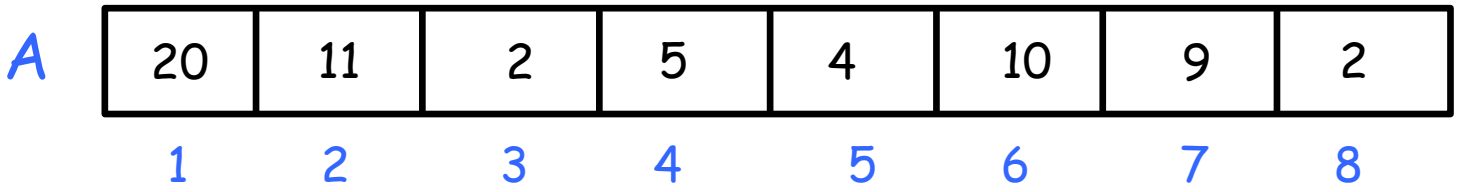
Inverti(A,i,f)
inverte l'ordine degli elementi di A[i:f]. Può essere implementata in modo richiedere tempo lineare e memoria costante (esercizio per voi)


74

Progettare un algoritmo efficiente per il seguente problema.

Input: vettore $A[1:n]$ di n numeri

Output: due indici i^* e j^*, con $i^*<j^*$, che massimizzano $A[j^*]-A[i^*]$, ovvero due indici tale che $A[j^*]-A[i^*] \geq A[j]-A[i]$, per ogni i, j con $i<j$.



$i^*=3$ $j^*=6$

Motivazione

A	20	11	2	5	4	10	9	2
	1	2	3	4	5	6	7	8

$i^*=3 \quad j^*=6$

se $A[i]$ è la quotazione di un titolo azionario a tempo i, risolvere il problema vuol dire trovare il miglior modo di acquistare e vendere il titolo per massimizzare il guadagno.

osservazioni preliminari

-algoritmo semplice con complessità $O(n^2)$

-provo tutte le coppie i,j con i<j

-prendere l'indice i* come indice del minimo e j* come indice del massimo non va bene

(per la condizione richiesta i*<j*)

goal: tempo O(n)

idea:

-dato i, il miglior j (che massimizza A[j]-A[i]) è l'indice del massimo in A[i+1,n]

-se dato i, sapessi trovare in tempo costante tale j, avrei un algoritmo con tempo O(n)

-posso precalcolare questi indici j?

- -vettore Max[1:n]
- -Max[k]: indice del massimo in A[k,n]

calcolo del vettore Max[]

sia Max[1:n] vettore di dimensione n

Max[n]=n

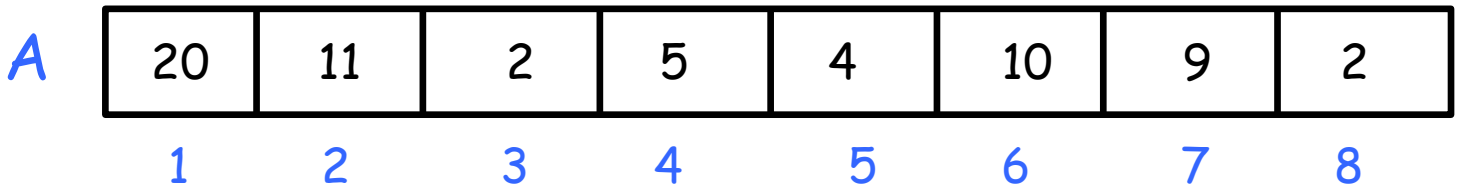
for k=n-1 down to 2 **do**

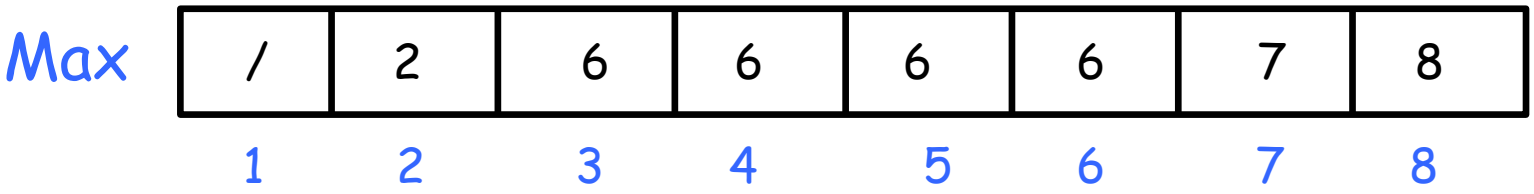
  if (A[k]>A[Max[k+1]]) **then** Max[k]=k

  else Max[k]=Max[k+1]

complessità?

$\Theta(n)$





L'algoritmo

```
MaxDiff(A)
sia Max[1:n] vettore di dimensione n
Max[n]=n
for k=n-1 down to 2 do
    if A[k]>A[Max[k+1]]  then Max[k]=k
                         else Max[k]=Max[k+1]
i*=1; j*=Max[2]; M=A[j*]-A[i*];
for i=2 to n-1 do
    if (A[Max[i+1]]-A[i])>M then i*=i;
                                 j*=Max[i+1];
                                 M=A[j*]-A[i*];
return (i*,j*)
```

correttezza?

Sì. Provo tutte le coppie (i,migliore j per i)

complessità?

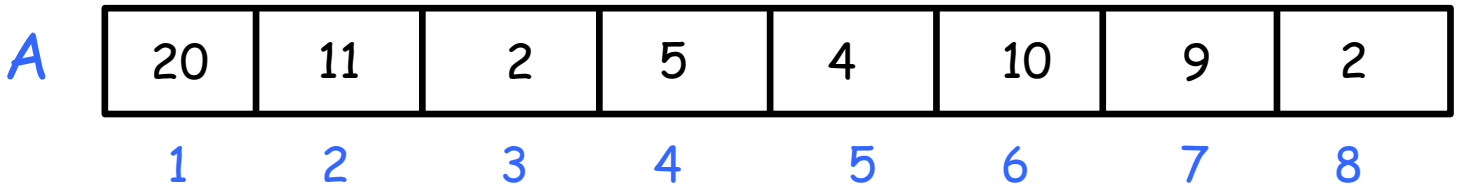
$\Theta(n)$

Qualche altro problema

Progettare un algoritmo efficiente per il seguente problema.

Input: vettore $A[1:n]$ di n numeri

Output: due indici i^* e j^*, con $i^*<j^*$, che massimizzano $A[j^*]-A[i^*]$, ovvero due indici tale che $A[j^*]-A[i^*] \geq A[j]-A[i]$, per ogni i, j con $i<j$.



$i^*=3$ $j^*=6$

goal: tempo $O(n)$ e memoria ausiliaria costante?

Problema

Sia $V[1 : n]$ un vettore di n caratteri, dove ogni posizione può contenere un carattere nell'insieme $\{Y, E, S\}$. Il vettore è organizzato in modo che, se letto da sinistra a destra, si ottiene prima una sequenza non vuota di Y, poi una sequenza non vuota di E, e poi una sequenza non vuota di S. Si progetti un algoritmo con complessità $o(n)$ che calcoli il numero di Y, di E e di S contenute nel vettore. Si fornisca lo pseudocodice dell'algoritmo.

A	Y	Y	E	E	E	E	S	S
	1	2	3	4	5	6	7	8

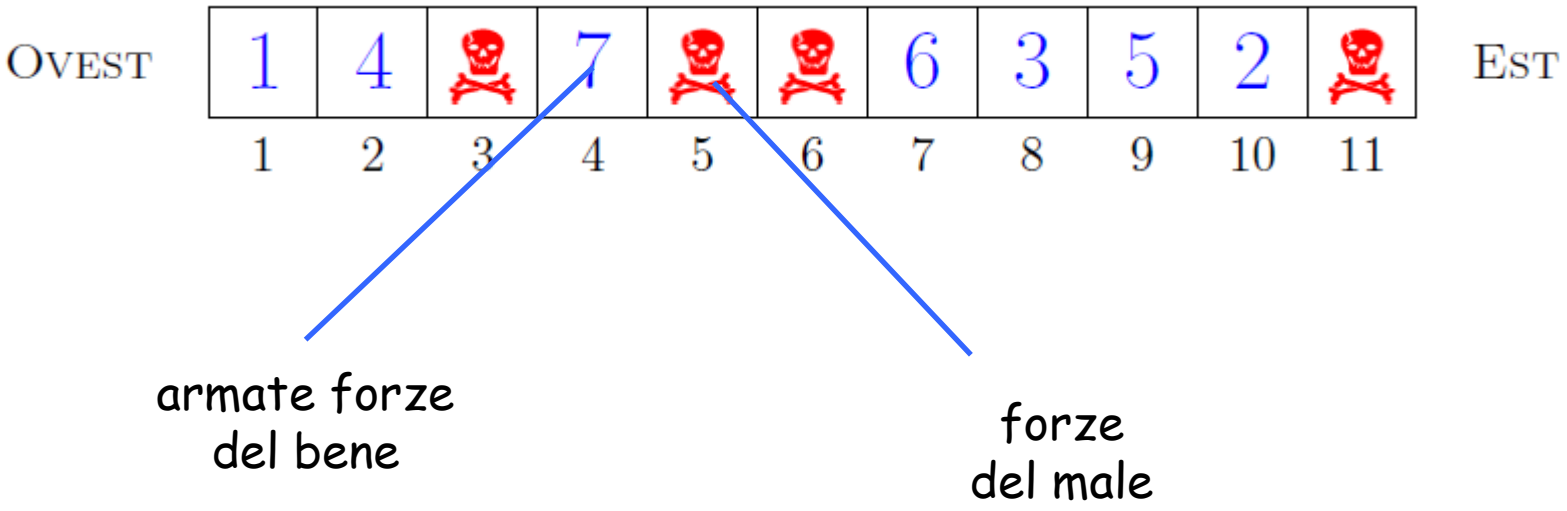
#Y=2
#E=4
#S=2

Problema

Nella Striscia Di Mezzo si combatte una guerra sanguinaria fra le Forze Del Bene e le Forze Del Male. La Striscia si estende da ovest a est ed è composta da n territori, numerati (da ovest a est) da 1 a n. Il territorio i-esimo confina con il territorio (i – 1)-esimo e (i + 1)- esimo. Ogni territorio è controllato da uno dei due schieramenti e, per ogni territorio controllato dalle Forze Del Bene, sono noti il numero di armate che difendono tale territorio.

Se le Forze Del Male di un territorio i vogliono conquistare un territorio j controllato dalle Forze Del Bene, devono farsi strada sconfiggendo tutte le armate delle Forze del Bene che si trovano nei territori tra i e j (j compreso). Per ogni territorio j controllato dalle Forze Del Bene, definiamo il fattore di difesa δ_j come il numero minimo di armate che le Forze Del Male devono sconfiggere per conquistare il territorio j. Progettate un algoritmo che calcoli in tempo O(n) tutti i fattori di difesa dei territori delle forze del bene.

Striscia di Mezzo



δ_j (fattore di difesa territorio j)
numero minimo di armate che le Forze Del Male devono sconfiggere per conquistare il territorio j

$\delta_1 = 5$

$\delta_8 = 9$

$\delta_9 = 7$

goal: calcolare in tempo O(n) tutti i δ_j