[7.1 - Selezione e Sotto Problemi]
[7.2 - Selezione in Tempo Lineare]

Selezione

> **Problema**
>
> - **Input**: un insieme A di n numeri distinti e un numero i compreso tra $1..n$
> - **Output**: l'elemento $x \in A$ maggiore di $i-1$ elementi di A e minore di $n-i$ elementi di A. In altre parole, restituire l'elemento di A in **posizione i nell'ordinamento**

Minimo, Massimo e Mediano

Tutti e tre i problemi sono in realtà **sottoproblemi della selezione**

$$\begin{aligned} \text{Min}(A) &= \text{Select}(A, 1) \\ \text{Max}(A) &= \text{Select}(A, n) \\ \text{Median}(A) &= \text{Select}\left(A, \frac{n}{2}\right) \end{aligned}$$

Possiamo facilmente implementare la **ricerca del massimo e del minimo**

MIN(A)

```
min = A[1]
for i = 2 to n
        if min > A[i]
                min = A[i]

return min
```

Parametri: *A=vettore*
Return: *valore minimo in A*

> **Costo Computazionale MIN**
>
> $\Theta(n)$

MAX(A)

```
max = A[1]
for i = 2 to n
        if max < A[i]
                max = A[i]

return max
```

Parametri: *A=vettore*
Return: *valore massimo in A*

> **Costo Computazionale MAX**
>
> $\Theta(n)$

Per implementare però la ricerca del mediano, bisogna **passare dalla funzione select** per la ricerca di un generico i.

L'approccio più semplice è quello di **ordinare il vettore** e prendere l'elemento in posizione i, come da definizione del problema

SELECT-SORT(A, i)

```
S = SORT(A)
return S[i]
```

Parametri: *A=vettore, i=indice elemento da trovare*
Return: *elemento di A in posizione i nell'ordinamento*

> **Costo Computazionale SELECT SORT**
>
> Utilizzando un algoritmo ottimo per il sort (es. merge sort)

$$\Theta(n\log(n))$$

Abbiamo quindi un **upper-bound** del problema di select, ma si può arrivare fino ad una selezione in tempo lineare

Selezione Con Partition

Idea

Se invece che ordinare il vettore interamente utilizziamo il partizionamento, possiamo trovare quanti valori sono minori e maggiori del pivot in tempo lineare

Ogni volta che eseguiamo il partizionamento di un vettore, troviamo gli elementi maggiori e minori del pivot selezionato.

Sapendo a che indice si trova il pivot e che indice i devo trovare, posso capire in quale metà di vettore chiamare ricorsivamente per trovare l'elemento che cerco

Osservazione: Pivot Randomizzato

La partizione ha un costo computazionale molto variabile in base allo sbilanciamento diverso ad ogni chiamata. Per questo, possiamo utilizzare un **pivot casuale** per ricadere nel caso medio

RANDOMIZED-SELECT(A, p, r, i)

```
if p == r
        return A[p]

q = RANDOMIZED-PARTITION(A, p, r)
k = q - p + 1 // distanza tra q e p
if i == k
        return A[q]
if i < k
        return RANDOMIZED-SELECT(A, p, q - 1, i)
if i > k
        return RANDOMIZED-SELECT(A, q + 1, r, i - k)
```

Parametri: *A=vettore, p=inizio vettore, r=fine vettore, i=indice elemento da trovare in A[p..r]*
Return: *elemento di A in posizione i nell'ordinamento*

Costo Computazionale

Avendo randomizzato il partizionamento, possiamo solo considerare il caso medio

$$\Theta(n)$$

Tempo Lineare

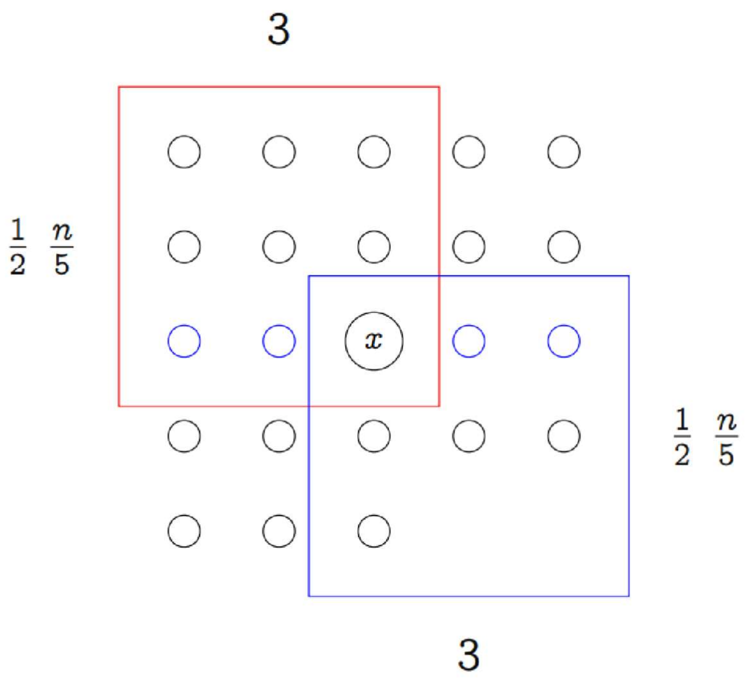
Idea

Selezionando un pivot non casuale ma con un criterio, possiamo determinare quanti elementi vengono scartati ogni partizionamento

Immaginiamo di avere il vettore ordinato.

Dividiamolo in gruppi di 5 e posizioniamo i gruppi su righe diverse per formare una matrice.

- La colonna centrale conterrà tutti i **mediani dei gruppi**
- Il valore centrale x sarà il **mediano dei mediani**



Osservazione: Quadranti

- **Blu**: valori **maggiori uguali** di x
- **Rosso**: valori **minori uguali** di x

Sappiamo che quindi, scegliendo x come pivot della partizione, **possiamo scartare alla peggio $\frac{3}{10}n$ elementi ogni volta che selezioniamo una delle due metà**

SELECT(A, p, r, i)

```
1  if r - p <= 5
2          return manually calculated result
3  sia M un nuovo arrai di n = (r - p + 1) / 5 elementi,
4  dividi gli elementi in A[p..r] in gruppi di 5,
5  calcola i mediani di tutti gruppi e inseriscili in M
6  x = SELECT(M, 1, n, n/2)
7  q = PARTITION(A, p, r, x)
8  k = q - p + 1 // distanza tra q e p
9  if i == k
10         return A[q]
11 if i < k
12         return SELECT(A, p, q - 1, i)
13 if i > k
14         return SELECT(A, q + 1, r, i - k)
```

***Parametri**: A=vettore, p=inizio vettore, r=fine vettore, i=indice elemento da trovare in A[p..r]*
***Return**: elemento di A in posizione i nell'ordinamento*

Sia n il numero di elementi di A

- **1-2**: calcolare la soluzione esplicita su 5 elementi avrà un costo costante $\Theta(1)$
- **3-5**: calcolare il mediano di un vettore di 5 elementi avrà un costo costante, il quale verrà ripetuto massimo $\Theta(n)$ volte
- **6**: la chiamata ricorsiva avrà un costo $T\left(\frac{n}{5}\right)$
- **7**: il partizionamento ha sempre costo lineare $\Theta(n)$
- **12,14**: le chiamate ricorsive a questi passi, grazie alla scelta del mediano dei mediani come pivot, sappiamo avere un costo $T\left(\frac{7}{10}n\right)$

Otteniamo quindi la formula

$$T(n) = T\left(\frac{1}{5}n\right) + T\left(\frac{7}{10}n\right) + \Theta(n)$$

Costo Computazionale

Possiamo dimostrare per induzione che il costo computazionale è

$$O(n)$$

Dimostrazione

Assumiamo che

$$\forall n' > n, T(n') \leq cn'$$

Calcoliamo

$$\begin{aligned}T(n) &= T\left(\frac{1}{5}n\right) + T\left(\frac{7}{10}n\right) + \Theta(n)\\ &\le c\frac{1}{5}n + c\frac{7}{10}n + kn\\ &\le c\frac{9}{10}n + kn\end{aligned}$$

Troviamo per quali valori di c la disequazione è verificata

$$\begin{aligned}c\frac{9}{10}n + kn \le cn\\ kn \le c\frac{1}{10}n\\ k \le c\frac{1}{10}\\ c \ge 10k\end{aligned}$$

La disequazione è verificata asintoticamente

Grandezza dei Gruppi

La grandezza dei gruppi pari a 5 non è scelta casualmente, è invece il **valore più piccolo per cui abbiamo un costo lineare

Prendiamo un generico numero dispari $P = 2k+1$

- Saranno presenti $\frac{n}{P}$ gruppi
- I due quadranti della matrice avranno altezza $\frac{n}{2P}$ e base $k+1$

La formula quindi diventerà

$$T\left(\frac{n}{P}\right) + T\left(n - \frac{n(k+1)}{2(P)}\right) + n$$
$$T\left(\frac{n}{2k+1}\right) + T\left(\frac{3k+1}{2(2k+1)}n\right) + n$$

Sappiamo che

$$T(an) + T((1-a)n) + n = \Theta(n\log(n))$$

Dobbiamo trovare quindi un valore di k per il quale

$$\frac{1}{2k+1} + \frac{3k+1}{2(2k)+1} < 1$$

Provando i valori

$$\begin{aligned}k = 1 &\implies 1\\ k = 2 &\implies \frac{9}{10}\end{aligned}$$

Quindi il valore più piccolo è

$$k = 2 \implies P = 5$$