5. Divide et Impera

- mergesort
- counting inversions
- closest pair of points
- mediana e selezione
- in-place partition e quicksort

La mediana ed il problema della Selezione

Selezione. Dati n elementi (es. numeri interi), trova il k-mo (nell'ordine dal più piccolo al più grande)

- Minimo: $k = 1$; massimo: $k = n$.
- Mediana: $k = \lfloor (n+1)/2 \rfloor$. (l'elemento in posizione "centrale")
- $O(n)$ confronti per trovare il minimo o il massimo.
- $O(n \log n)$ confronti se ordiniamo prima.
- $O(n \log k)$ confronti se usiamo heap binari (**esercizio**).

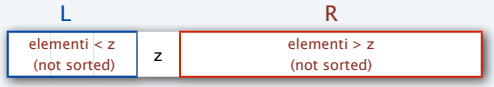
Domanda. C'è differenza tra trovare il minimo o il k-mo elemento?

- E' possibile trovare il k-mo elemento in O(n)?

Selezione

Partizioniamo l'array in tre parti:

- un elemento z.
- elementi $< z$ nel subarray L.
- elementi $> z$ nel subarray R.



facciamo una chiamata ricorsiva in uno solo dei subarray—quello che contiene il k-mo elemento più piccolo.

SELEZIONE (A, k)

può essere fatto in tempo lineare anche "in-place"

IF $|A| = 1$ RETURN $A[1]$.

Scegli $z \in A$ *uniformemente* a caso (ogni elemento ha la stessa probabilità di essere scelto).

$(L, R) \leftarrow$ PARTITION (A, z) // separa gli elementi tra $>z$ e $<z$

IF $k \leq |L|$ RETURN SELEZIONE (L, k).

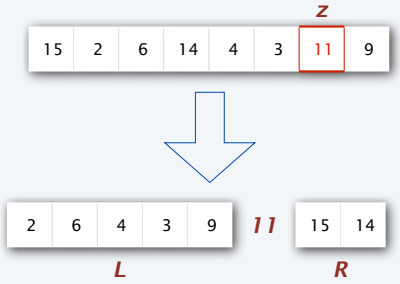
ELSE IF $k > |L| + 1$ RETURN SELEZIONE $(R, k - |L| - 1)$

ELSE RETURN z.

Partition - assumiamo che in A tutti gli elementi siano diversi

Crea da A due array L e R

- L contiene gli elementi < z
- R contiene gli elementi > z
- gli array L e R non vengono ordinati



PARTITION $(A[1...n], z)$

$i \leftarrow 1$, $\ell \leftarrow 0$, $r \leftarrow 0$

// ℓ , r contano gli elementi messi in L e R

WHILE $(i \leq n)$

  IF $(A[i] < z)$

    $\ell \leftarrow \ell + 1$

    $L[\ell] \leftarrow A[i]$

  ELSE

    $r \leftarrow r + 1$

    $R[r] \leftarrow A[i]$

  $i \leftarrow i + 1$

RETURN $(L[1...\ell], R[1...r])$

Analisi di SELEZIONE (A, k)

Intuizione. Se scelto (random) uniformemente, il valore atteso del max{|L|,|R|} è ¾.

$$T(n) \le \max\{T(|L|), T(|R|)\} + n \le T(\tfrac{3}{4}n) + n \quad \Rightarrow \quad T(n) \le 4n$$

Def. $T(n, k)$ = # atteso di confronti per selezionare il k-mo in array di taglia $\le n$.
Def. $T(n) = \max_k T(n, k)$.
Def. Per un array A e un valore x in A, $rango_A(x)$ = posizione di x nella versione ordinata di A. Es. per A[1...8] := 15, 3, 4, 7, 2, 6 1, 10, 9 abbiamo $rango_A(7) = 6$

Def. Per un array A e un valore z in A, diciamo che z è un buon partizionatore se e solo se $1/4\, n \le rango_A(z) \le 3/4\, n$.

Proposizione. In ogni array $A[1...n]$ ci sono $n/2$ elementi z che sono buoni partizionatori.
Dim. Ordiniamo l'array. Tutti i valori che sono tra la posizione 1/4 n e la posizione 3/4 n sono buoni partizionatori, per definizione. Tra la posizione 1/4 n e la posizione 3/4 n ci sono n/2 posizioni.

Analisi di SELEZIONE (A, k)

Caso 1. Se ogni chiamata ricorsiva di SELEZIONE sceglie uno z che è un buon partizionatore allora vale la ricorrenza

$$T(n) \le \max\{T(|L|), T(|R|)\} + c\,n = \max\{T(rango(z)), T(n\text{-}rango(z))\} + c\,n$$
$$\le T(\tfrac{3}{4}n) + c\,n \quad \Rightarrow \quad T(n) = O(n)$$

Caso 2. Se ogni chiamata ricorsiva di SELEZIONE sceglie uno z che non è un buon partizionatore (cioè $rango_A(z) \le 1/4\, n$ *oppure* $3/4\, n \le rango_A(z)$) allora vale la ricorrenza

$$T(n) \le \max\{T(|L|), T(|R|)\} + c\,n = \max\{T(rango(z)), T(n\text{-}rango(z))\} + c\,n$$
$$\le T(n) + c\,n$$

In termini di probabilità, nella metà dei casi vale il caso 1 e nella metà dei casi vale il caso due, quindi possiamo stimare il tempo medio di SELEZIONE come la media dei casi 1 e 2:

$$T(n) \le 1/2\ (\text{Caso } 1) + 1/2\ (\text{Caso } 2) = 1/2\ T(n) + 1/2\ T(\tfrac{3}{4}n) + c\,n$$
$$\Rightarrow \quad T(n) \le T(\tfrac{3}{4}n) + c\,n \quad \Rightarrow \quad T(n) = O(n)$$

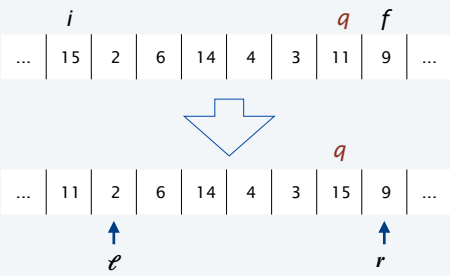
5. DIVIDE ET IMPERA

- mergesort
- counting inversions
- closest pair of points
- mediana e selezione
- in-place partition e quicksort

Partition in place - assumiamo che in A tutti gli elementi siano diversi

Riordina la parte di A dall'indice i a f

- prima tutti gli elementi < A[q]
  - *non necessariamente ordinati*
- poi A[q]
- poi tutti gli elementi > A[q]
  - *non necessariamente ordinati*

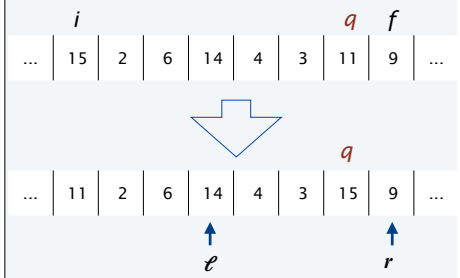


PARTITION-IN-PLACE (A,i,f,q)

$z \leftarrow A[q]$,
Scambia $A[q] \leftrightarrow A[i]$
$\ell \leftarrow i+1,\ r \leftarrow f$
// ℓ , r *contano gli elementi messi in L e R*
WHILE $(\ell \le r)$
  WHILE $(A[\ell] < z)$
    $\ell \leftarrow \ell+1$
  WHILE $(A[r] > z)$
    $r \leftarrow r-1$
  IF $(\ell < r)$
    Scambia $A[\ell] \leftrightarrow A[r]$
    $\ell \leftarrow \ell+1,\ r \leftarrow r-1$
Scambia $A[i] \leftrightarrow A[r]$
RETURN

Partition - assumiamo che in A tutti gli elementi siano diversi

Riordina la parte di A dall'indice i a f
- prima tutti gli elementi < A[q]
  - *non necessariamente ordinati*
- poi A[q]
- poi tutti gli elementi > A[q]
  - *non necessariamente ordinati*

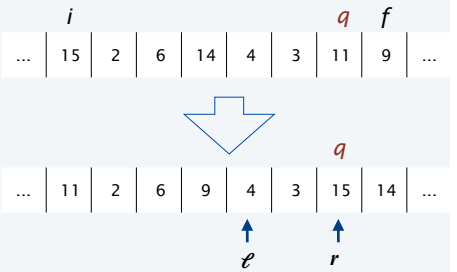


PARTITION-IN-PLACE (A,i,f,q)

$z \leftarrow A[q]$,
Scambia $A[q] \leftrightarrow A[i]$
$\ell \leftarrow i+1, r \leftarrow f$
// ℓ , r *contano gli elementi messi in L e R*
WHILE $(\ell \le r)$
  WHILE $(A[\ell] < z)$
    $\ell \leftarrow \ell+1$
  WHILE $(A[r] > z)$
    $r \leftarrow r-1$
  IF $(\ell < r)$
    Scambia $A[\ell] \leftrightarrow A[r]$
    $\ell \leftarrow \ell+1,\ r \leftarrow r-1$
Scambia $A[i] \leftrightarrow A[r]$
RETURN

Partition - assumiamo che in A tutti gli elementi siano diversi

Riordina la parte di A dall'indice i a f
- prima tutti gli elementi < A[q]
  - *non necessariamente ordinati*
- poi A[q]
- poi tutti gli elementi > A[q]
  - *non necessariamente ordinati*

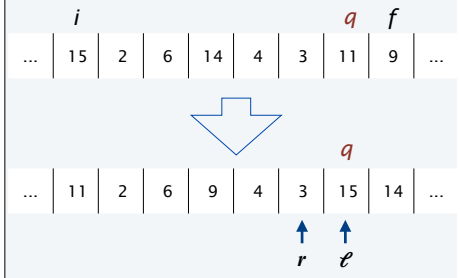


PARTITION-IN-PLACE (A,i,f,q)

$z \leftarrow A[q]$,
Scambia $A[q] \leftrightarrow A[i]$
$\ell \leftarrow i+1, r \leftarrow f$
// ℓ , r *contano gli elementi messi in L e R*
WHILE $(\ell \le r)$
  WHILE $(A[\ell] < z)$
    $\ell \leftarrow \ell+1$
  WHILE $(A[r] > z)$
    $r \leftarrow r-1$
  IF $(\ell < r)$
    Scambia $A[\ell] \leftrightarrow A[r]$
    $\ell \leftarrow \ell+1,\ r \leftarrow r-1$
Scambia $A[i] \leftrightarrow A[r]$
RETURN

Partition - assumiamo che in A tutti gli elementi siano diversi

Riordina la parte di A dall'indice i a f
- prima tutti gli elementi < A[q]
  - *non necessariamente ordinati*
- poi A[q]
- poi tutti gli elementi > A[q]
  - *non necessariamente ordinati*

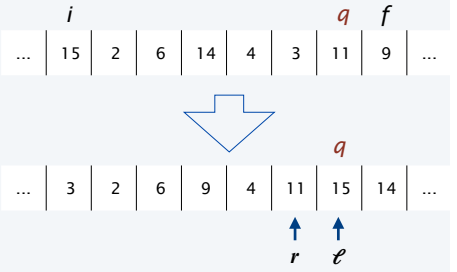


PARTITION-IN-PLACE (A,i,f,q)

$z \leftarrow A[q]$,
Scambia $A[q] \leftrightarrow A[i]$
$\ell \leftarrow i+1, r \leftarrow f$
// ℓ , r *contano gli elementi messi in L e R*
WHILE $(\ell \le r)$
  WHILE $(A[\ell] < z)$
    $\ell \leftarrow \ell+1$
  WHILE $(A[r] > z)$
    $r \leftarrow r-1$
  IF $(\ell < r)$
    Scambia $A[\ell] \leftrightarrow A[r]$
    $\ell \leftarrow \ell+1,\ r \leftarrow r-1$
Scambia $A[i] \leftrightarrow A[r]$
RETURN

Partition - assumiamo che in A tutti gli elementi siano diversi

Riordina la parte di A dall'indice i a f
- prima tutti gli elementi < A[q]
  - *non necessariamente ordinati*
- poi A[q]
- poi tutti gli elementi > A[q]
  - *non necessariamente ordinati*



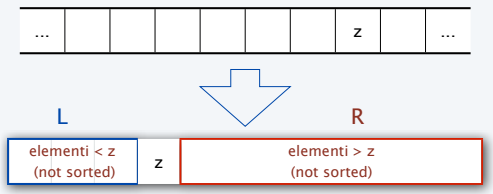
PARTITION-IN-PLACE (A,i,f,q)

$z \leftarrow A[q]$,
Scambia $A[q] \leftrightarrow A[i]$
$\ell \leftarrow i+1, r \leftarrow f$
// ℓ , r *contano gli elementi messi in L e R*
WHILE $(\ell \le r)$
  WHILE $(A[\ell] < z)$
    $\ell \leftarrow \ell+1$
  WHILE $(A[r] > z)$
    $r \leftarrow r-1$
  IF $(\ell < r)$
    Scambia $A[\ell] \leftrightarrow A[r]$
    $\ell \leftarrow \ell+1,\ r \leftarrow r-1$
Scambia $A[i] \leftrightarrow A[r]$
RETURN r // *ritorna il rango del pivot*

Quicksort (randomizzato)

Idea. Partizioniamo l'array in tre parti:

- un elemento z.
- elementi $< z$ nel subarray L.
- elementi $> z$ nel subarray R.



l'elemento z è ora nella posizione corretta (è "ordinato") rispetto agli altri
Per ordinare il resto facciamo una chiamata ricorsiva in entrambi i subarray.

Quicksort (randomizzato) - usando PARTITION-IN-PLACE

Idea. Partizioniamo la parte dell'array da ordinare A[i...j] in tre parti:

- il pivot z - va nella posizione "ordinata" cioè $s = rango_A(z)$
- elementi $< z$ nel subarray $A[i...s-1]$.
- elementi $> z$ nel subarray $A[s+1...j]$.

l'elemento z è ora nella posizione corretta (è "ordinato") rispetto agli altri
Per ordinare il resto facciamo una chiamata ricorsiva in entrambi i subarray.

QUICKSORT (A, i, j)

IF $(j \le i)$ RETURN

Scegli $q \in \{i, ..., j\}$ *uniformemente* a caso (ogni indice ha la stessa probabilità di essere scelto).

$s \leftarrow$ PARTITION-IN-PLACE (A, i, j, q)

QUICKSORT $(A, i, s\text{-}1)$

QUICKSORT $(A, s+1, j)$

Analisi di QUICKSORT (simile a selection)

Caso 1. Se ogni chiamata ricorsiva di QUICKSORT sceglie uno z che è un buon partizionatore allora vale la ricorrenza

$$T(n) \le T(|L|) + T(|R|) + c\,n = T(rango(z)) + T(n\text{-}rango(z)) + c\,n$$
$$\le T(\tfrac{1}{4}n) + T(\tfrac{3}{4}n) + c\,n$$

Caso 2. Se ogni chiamata ricorsiva di SELEZIONE sceglie uno z che non è un buon partizionatore (cioè $rango_A(z) \le 1/4\,n$ *oppure* $3/4\,n \le rango_A(z)$)
allora vale la ricorrenza

$$T(n) \le T(|L|) + T(|R|) + c\,n = T(rango(z)) + T(n\text{-}rango(z)) + c\,n$$
$$\le T(n\text{-}1) + T(0) + c\,n \le T(n) + c\,n \qquad (assumiamo\ T(n\text{-}1) \le T(n)\ e\ T(0)=0)$$

In termini di probabilità, nella metà dei casi vale il caso 1 e nella metà dei casi vale il caso 2, quindi possiamo stimare il tempo medio di QUICKSORT come la media dei casi 1 e 2:

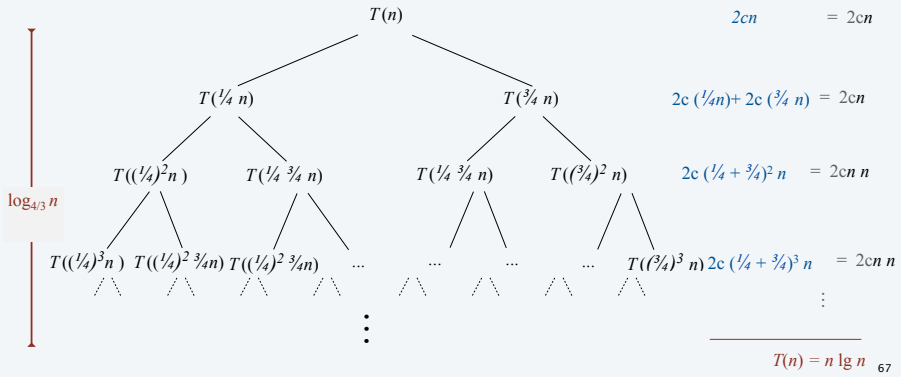
$$T(n) \le 1/2\ (\text{Caso 2}) + 1/2\ (\text{Caso 1}) = 1/2\ T(n) + 1/2\ (T(\tfrac{1}{4}n) + T(\tfrac{3}{4}n)) + c\,n$$
$$\Rightarrow\ T(n) \le T(\tfrac{1}{4}n) + T(\tfrac{3}{4}n) + 2cn \ \Rightarrow\ T(n) = O(n \log n)$$

Risoluzione della ricorrenza mediante l'Albero di ricorsione

Proposizione. Se $T(n)$ soddisfa la seguente ricorrenza, allora $T(n) = O(n \log_2 n)$

$$T(n) = \begin{cases} 0 & \text{se } n \le 1 \\ T(\tfrac{1}{4}n) + T(\tfrac{3}{4}n) + 2cn & \text{altrimenti} \end{cases}$$

Dim 1.



5. Divide et Impera

- mergesort
- counting inversions
- closest pair of points
- mediana e selezione
- "in-place" partition e quicksort
- selezione in O(n) deterministicamente

Selezione in tempo lineare (nel caso pessimo)

Scopo. Troviamo un elemento pivot z che divide l'array A di n elementi in due parti garantendo che la maggiore contiene $\leq 7/10\, n$ elements.

Selezione deterministico - SEL-DET (A, k)

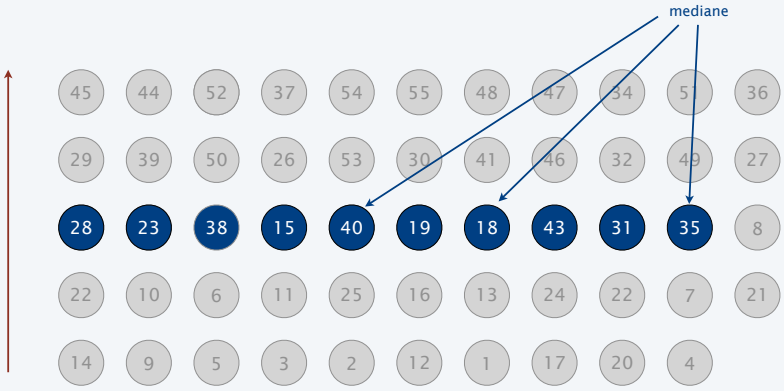
1. Dividiamo gli n elementi in $\lfloor n / 5 \rfloor$ gruppi di 5 elementi (più al massimo un gruppo di taglia inferiore).

29	10	38	37	2	55	18	24	34	35	36
22	44	52	11	53	12	13	43	20	4	27
28	23	6	26	40	19	1	46	31	49	8
14	9	5	3	54	30	48	47	32	51	21
45	39	50	15	25	16	41	17	22	7	

N = 54

Selezione deterministico - SEL-DET (A, k)

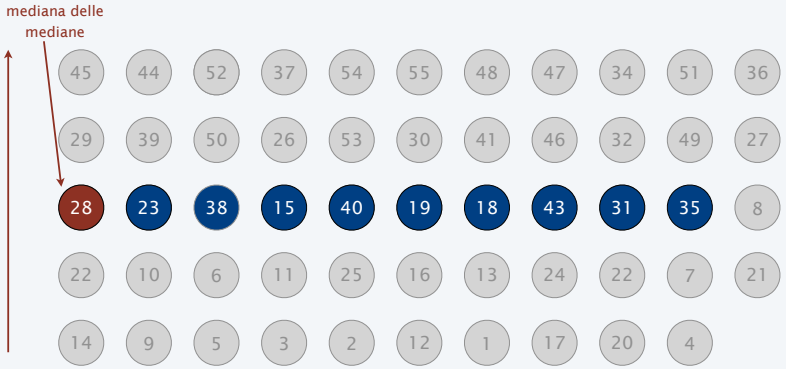
1. Dividi gli n elementi in $\lfloor n / 5 \rfloor$ gruppi di 5 elementi (più al massimo un gruppo di taglia inferirore).
2. Ordina ogni gruppo da 5 e trova la mediana.



N = 54

Selezione deterministico - SEL-DET (A, k)

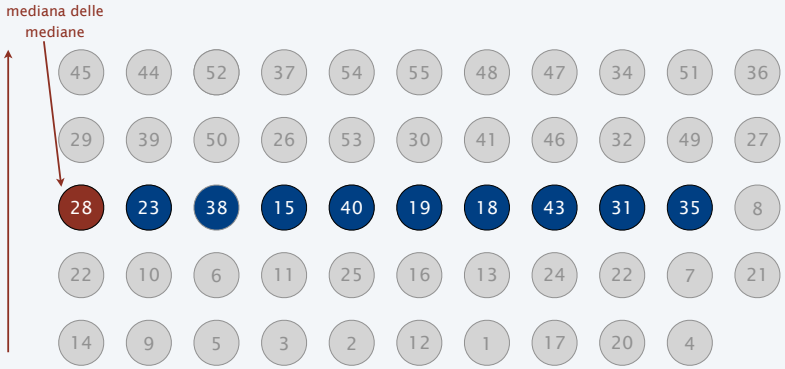
1. Dividi gli n elementi in $\lfloor n / 5 \rfloor$ gruppi di 5 elementi (più al massimo un gruppo di taglia inferirore).
2. Ordina ogni gruppo da 5 e trova la mediana.
3. trova la mediana delle n/5 mediane ricorsivamente usando SEL-DET()



N = 54

Selezione deterministico - SEL-DET (A, k)

1. Dividi gli n elementi in $\lfloor n / 5 \rfloor$ gruppi di 5 elementi (più al massimo un gruppo di taglia inferirore).
2. Ordina ogni gruppo da 5 e trova la mediana.
3. trova la mediana delle n/5 mediane ricorsivamente usando SEL-DET()
3. usa la mediana delle mediane come pivot, continua come SELEZIONE()



N = 54

Median-of-medians selection algorithm

SEL-DET (A, k)

$n \leftarrow |A|$.

IF $n < 70$ RETURN k^{th} smallest of element of A via mergesort.

// caso base --- 70 è scelto per ragioni tecniche

Dividi A in $\lfloor n / 5 \rfloor$ gruppi di 5 elementi ciascuno (più un gruppo < 5).

Sia B l'array delle mediane di ogni gruppo da 5.

$z \leftarrow$ SEL-DET$(B, \lfloor |B|+1 / 2 \rfloor)$ ← mediana delle mediane

$(L, R) \leftarrow$ PARTITION (A, z) // separa gli elementi tra >z e <z

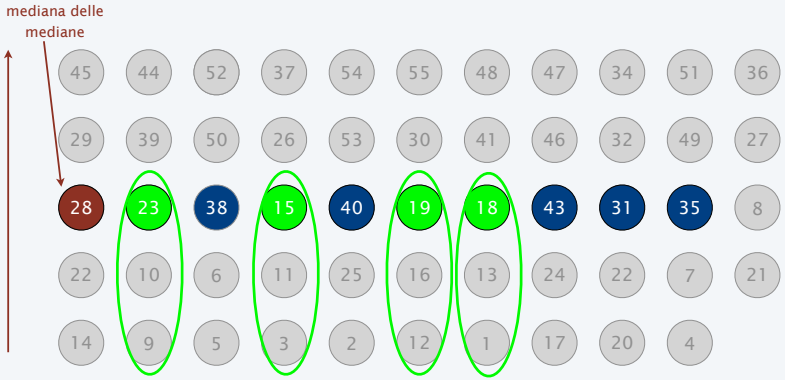
IF $k < |L| + 1$ RETURN SELEZIONE (L, k).

IF $k > |L| + 1$ RETURN SELEZIONE $(R, k - |L| - 1)$

IF $k = |L| + 1$ RETURN z.

Analisi di Selezione deterministico - SEL-DET (A, k)

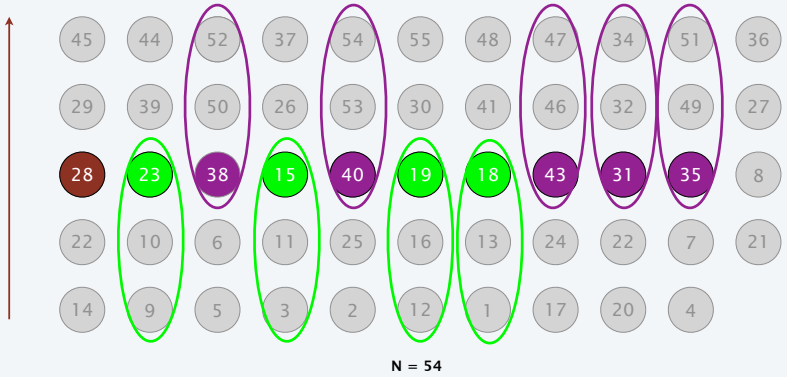
- almeno metà delle mediane dei gruppi da 5-element $\leq z$.
- nei gruppi di tali mediane, ci sono quindi almeno 3 elementi $\leq z$
- *quindi il numero di elementi* $\leq z$ *è almeno* $3\ (1/2)\ (n/5) = \mathbf{3n/10}$



N = 54

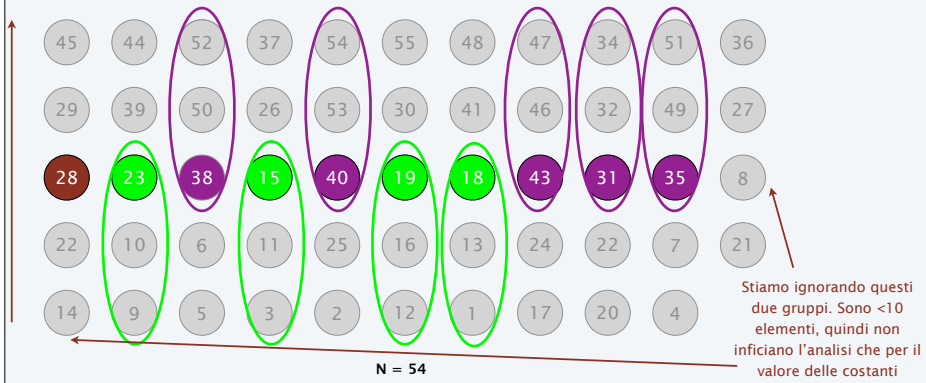
Analisi di Selezione deterministico - SEL-DET (A, k)

- almeno metà delle mediane dei gruppi da 5-element $\leq z$.
- nei gruppi di tali mediane, ci sono quindi almeno 3 elementi $\leq z$
- *quindi il numero di elementi* $\leq z$ *è almeno* $3\ (1/2)\ (n/5) = 3n/10$
- almeno metà delle mediane dei gruppi da 5-element $\geq z$.
- nei gruppi di tali mediane, ci sono quindi almeno 3 elementi $\geq z$
- *quindi il numero di elementi* $\geq z$ *è almeno* $3\ (1/2)\ (n/5) = 3n/10$



Analisi di Selezione deterministico - SEL-DET (A, k)

- almeno metà delle mediane dei gruppi da 5-element $\leq z$.
- nei gruppi di tali mediane, ci sono quindi almeno 3 elementi $\leq z$
- *quindi il numero di elementi* $\leq z$ *è almeno* $3\ (1/2)\ (n/5) = 3n/10$
- almeno metà delle mediane dei gruppi da 5-element $\geq z$.
- nei gruppi di tali mediane, ci sono quindi almeno 3 elementi $\geq z$
- *quindi il numero di elementi* $\geq z$ *è almeno* $3\ (1/2)\ (n/5) = 3n/10$



Analisi di SEL-DET (A, k) - la ricorrenza

Come otteniamo la relazione di ricorrenza.

- SEL-DET è chiamato ricorsivamente sulle $n / 5$ mediane per determinare la mediana delle mediane z.
- Almeno $3\ n / 10$ elementi $\leq z$.
- Almeno $3\ n / 10$ elements $\geq z$.
- Quindi SEL-DET viene chiamato su al più $(n - 3\ n / 10) = 7n/10$ elementi.

$$T(n) \leq \begin{cases} \mathrm{O}(1) & \text{se } n \leq 70 \\ T(n/5) + T(7n/10) + cn & \text{altrimenti} \end{cases}$$

costo della chiamata ricorsiva per trovare la mediana delle mediane

costo della chiamata ricorsiva. Per avere un upper bound, assumiamo che venga fatta nella parte più grande

costo di:
- dividere in gruppi
- ordinare e trovare la mediana in ogni gruppo
- partition

Risolviamo usando l'albero di ricorsione.

Risoluzione della ricorrenza mediante l'Albero di ricorsione

Proposizione. Se $T(n)$ soddisfa la seguente ricorrenza, allora $T(n) = \mathrm{O}(n)$

$$T(n) = \begin{cases} \mathrm{O}(1) & \text{se } n \leq 70 \\ T(2n/10) + T(7n/10) + cn & \text{altrimenti} \end{cases}$$

Dim 1.