

# Statistiche d'ordine

Moreno Marzolla  
Dip. di Scienze dell'Informazione  
Università di Bologna

marzolla@cs.unibo.it  
<http://www.moreno.marzolla.name/>

Original work Copyright © Alberto Montresor, University of Trento  
(<http://www.dit.unitn.it/~montreso/asd/index.shtml>)

Modifications Copyright © 2009, Moreno Marzolla, Università di Bologna

*This work is licensed under the Creative Commons Attribution-NonCommercial-ShareAlike License. To view a copy of this license, visit <http://creativecommons.org/licenses/by-nc-sa/2.5/> or send a letter to Creative Commons, 543 Howard Street, 5th Floor, San Francisco, California, 94105, USA.*

# Statistica

- Algoritmi statistici su vettori: Estraggono da un vettore numerico alcune caratteristiche statisticamente rilevanti
- Esempi
  - **Media:**  $\mu = (A[1] + A[2] + \dots + A[n])/n$
  - **Varianza:** (sample variance):  $\sigma^2 = \frac{1}{n} \sum_{i=1}^n (A[i] - \mu)^2$
  - **Moda:** il valore (o valori) più frequente
  - **Mediano:** il valore che occuperebbe la posizione (n/2) se l'array fosse ordinato
- **Selezione del k-esimo minimo**
  - Dato un array  $A[1..n]$  di valori distinti e un valore  $1 \leq k \leq n$ , trovare l'elemento che è maggiore di esattamente  $k-1$  elementi

# Statistiche: Classificazione

- Statistiche distributive, algebriche
  - Oltre ai dati originali di dimensione  $n$ , richiedono uno spazio  $O(1)$  per essere calcolate
  - Non richiedono ordine
  - Esempi: Media, varianza, momenti, etc.
- Statistiche d'ordine (o *olistiche*)
  - Sono basate sul concetto d'ordine
  - Richiedono uno spazio  $O(f(n))$  per essere calcolate
  - Esempi: Moda (valore piu' frequente), mediana, selezione
  - Metodo semplice: Ordinamento + estrazione:  
→ tempo  $O(n \log n)$

# Selezione del k-esimo: a che serve?

Google   [Ricerca avanzata](#)  
[Preferenze](#)

Cerca:  nel Web  pagine in Italiano  pagine provenienti da: Italia

**Web** Risultati **1 - 10** su circa **104.000** per **algoritmi e strutture dati**. (0,53 secondi)

**Algoritmi e Strutture Dati**  
Ingegneria degli **Algoritmi** (Parte I). Contenuti: In questo modulo imparerai a progettare ed analizzare **algoritmi e strutture dati**. Docente: ...  
[gauguin.info.uniroma2.it/~italiano/.../Algoritmi/](http://gauguin.info.uniroma2.it/~italiano/.../Algoritmi/) - [Copia cache](#) - [Simili](#)

**Corso di Algoritmi e Strutture Dati**  
Il corso ha lo scopo di presentare i concetti fondamentali riguardanti l'analisi e il progetto di **algoritmi e strutture dati** efficienti. ...  
[www.dit.unitn.it/~montreso/asd/index.shtml](http://www.dit.unitn.it/~montreso/asd/index.shtml) - [Copia cache](#) - [Simili](#)

**Sito "Algoritmi e strutture dati - Progetto di algoritmi e ..."**  
Progetto di **algoritmi e strutture dati** in Java. Camil Demetrescu, Irene Finocchi, Giuseppe F. Italiano, Umberto Ferraro Petrillo ...  
[www.ateneonline.it/demetrescu/](http://www.ateneonline.it/demetrescu/) - [Copia cache](#) - [Simili](#)

**Sito "Algoritmi e strutture dati 2/ed", Camil Demetrescu, Irene ...**  
Questo libro offre un'introduzione allo studio degli **algoritmi** e delle **strutture dati**, cercando di conciliare comprensibilità, chiarezza di esposizione e ...  
[www.ateneonline.it/demetrescu2e/homeA.asp](http://www.ateneonline.it/demetrescu2e/homeA.asp) - [Copia cache](#) - [Simili](#)

**Appunti di Algoritmi E Strutture Dati - Studenti.it**  
26 feb 2007 ... Tutte le risorse per **algoritmi e strutture dati**. Tantissimi riassunti e tesine appunti completamente gratis.  
[www.studenti.it/appunti/.../algoritmi-e-strutture-dati/](http://www.studenti.it/appunti/.../algoritmi-e-strutture-dati/) - [Copia cache](#) - [Simili](#)

**Algoritmi e strutture dati**  
M. Torelli, Appunti per il corso di **Algoritmi e Strutture Dati** ... C. Demetrescu, I. Finocchi, G. Italiano **Algoritmi e strutture dati**, McGraw-Hill, 2004. ...  
[homes.dsi.unimi.it/~goldwurm/algo/](http://homes.dsi.unimi.it/~goldwurm/algo/) - [Copia cache](#) - [Simili](#)

**Corso di Algoritmi e Strutture Dati**  
Corso di **Algoritmi e Strutture Dati**. Dott. Giorgio Terracina. A.A. 2008/2009. (Ultimo aggiornamento: 15/06/2009). AVVISO: Risultati dell'ultimo appello ...  
[www.mat.unical.it/terraccina/asd/](http://www.mat.unical.it/terraccina/asd/) - [Copia cache](#) - [Simili](#)

**Lezioni di algoritmi e strutture dati - Maniezzo Vittorio; Margara ...**  
Libro di Maniezzo Vittorio; Margara Luciano, Lezioni di **algoritmi e strutture dati**, Pitagora.  
[www.ibs.it/codef/.../lezioni-algoritmi-strutture.html](http://www.ibs.it/codef/.../lezioni-algoritmi-strutture.html) - [Copia cache](#) - [Simili](#)

**Algoritmi e Strutture Dati |**  
[www.di.unipi.it/didadoc/asd1/](http://www.di.unipi.it/didadoc/asd1/) - [Copia cache](#) - [Simili](#)

104.000 risultati!

# Selezione del k-esimo: a che serve?

- I motori di ricerca producono molti risultati a fronte di una singola query
- I risultati vengono mostrati in **pagine**, in ordine decrescente di rilevanza
  - Nella prima pagina i risultati più rilevanti
  - Nella seconda quelli meno, e così via
- E' inutile ordinare tutti i risultati in base alla rilevanza
  - Quanti vanno frequentemente oltre la quarta pagina di risultati della ricerca?
- E' quindi utile selezionare i primi k risultati, e via via i successivi, se l'utente seleziona le altre pagine
  - Stay tuned

# Selezione: casi particolari

## Ricerca del minimo (massimo)

```
algorithm minimum(A)
  min := A[1]
  for i := 2 to A.length
    if (A[i] < min) then
      min = A[i]
    endif
  endfor
  return min
```

- $T(n) = n-1 = \Theta(n)$  confronti
- Possiamo dimostrare che questo algoritmo è ottimale?
  - Idea: scelta del minimo come un “torneo”
  - Tutti gli elementi (tranne il vincitore) deve perdere almeno un “partita”
  - Quindi il problema è  $\Omega(n)$

# Selezione: casi particolari

## Ricerca del minimo e massimo

```
algorithm min-max(A)
  min := +∞
  max := -∞
  for i := 1 to A.length-1 step 2 do
    if (A[i] ≤ A[i+1]) then
      if (A[i] < min) then
        min := A[i]
      endif
      if (A[i+1] > max) then
        max := A[i+1]
      endif
    else
      if (A[i] > max) then
        max := A[i]
      endif
      if (A[i+1] < min) then
        min := A[i+1]
      endif
    endif
  return (min,max)
```

- Algoritmo banale:
  - due istanze di min, max
  - Costo:  
 $T(n) = 2n-2 = \theta(n)$
- Algoritmo “ottimizzato”
  - assumiamo n pari
  - Costo:  
 $T(n) = 3n/2 = \theta(n)$

# Selezione: casi particolari

- Ricerca del secondo minimo
  - Trovare il secondo elemento più piccolo dell'array A[]
  - Costo:  $2n-3$  confronti nel caso peggiore (si verifica quando i valori sono in ordine decrescente)

```
algorithm minimum2 (A)
  min1 := A[1]
  min2 := A[2]
  if (min2 < min1) then
    swap(min1, min2)
  endif
  for i:= 3 to A.length do
    if (A[i] < min2) then
      min2 = A[i]
      if (min2 < min1) then
        swap(min1, min2)
      endif
    endif
  endif
  return min2
```

# Selezione: casi particolari

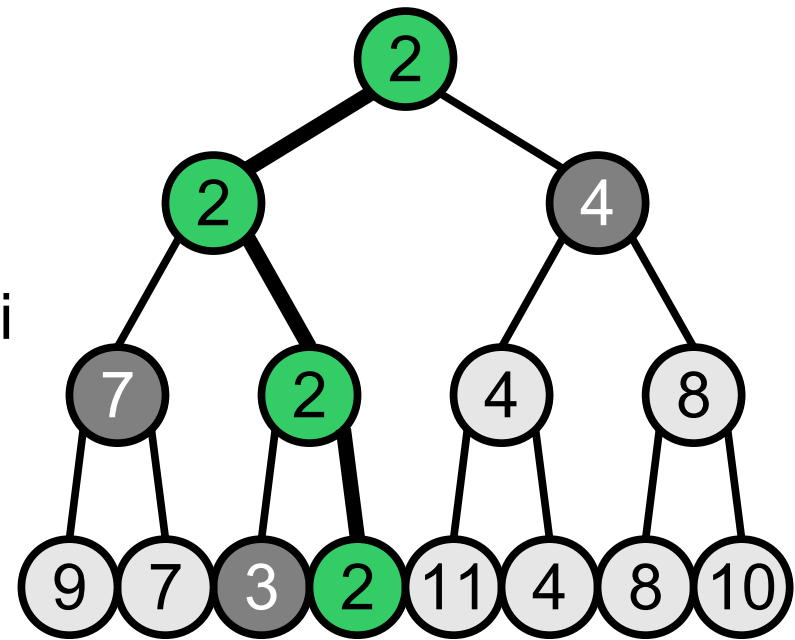
- Ricerca del secondo minimo

- L'*albero del torneo* permette di trovare il secondo minimo in  $O(n + \log n)$  confronti nel caso pessimo

- Dimostrazione

- $n$  passi necessari per la ricerca del minimo
- Siano  $M$  e  $S$  il minimo e il secondo minimo
- Sicuramente c'è stato un "incontro" fra  $M$  e  $S$ , dove  $M$  ha "vinto"
- Se così non fosse, esisterebbe un valore  $X < S$  che ha "battuto"  $S$  → assurdo dalla definizione di  $S$

- Quindi, basta cercare nei  $\log n$  valori "battuti" direttamente da  $m$  per trovare il secondo minimo. Totale:  $O(n + \log n)$



# Selezione del k-esimo elemento

```
algorithm select(A[], k)
  for i:=1 to k do
    minIndex = i
    minValue = A[i]
    for j:=i+1 to A.length do
      if (A[j] < minValue) then
        minIndex = j
        minValue = A[j]
      endif
    endfor
    swap A[i] and A[minIndex]
  endfor
  return A[k]
```

- In sostanza un Selection Sort incompleto
  - Si ferma al k-esimo elemento
- Complessità:  $O(kn)$

# Selezione per piccoli valori di $k$

- Costruisco un min-heap a partire dai valori
  - Costo  $O(n)$
- Estraggo per  $k-1$  volte il minimo
  - Costo  $O(k \log n)$
- Il  $k$ -esimo minimo è l'elemento che rimane in cima allo heap
  - Costo complessivo:  $O(n + k \log n)$

```
algorithm heapselect(A, k)
  min-heapify(A)
  for i := 1 to k-1 do
    deleteMin(A)
  return findMin(A)
```

Questa funzione deve costruire un min-heap

# Esempio

- Estrarre i primi  $k$  elementi da una query che ha fornito  $n$  match costa  $O(n + k \log n) = O(n)$  se  $k$  è  $O(n/\log n)$ 
  - Esempio:  $k=10$ ,  $n=104000$

The screenshot shows a Google search interface with the query "algoritmi e strutture dati". The search results are displayed under the "Web" tab, showing 10 results out of approximately 104,000. The results include various educational resources, course materials, and books related to algorithms and data structures.

Google   [Ricerca avanzata](#)  
[Preferenze](#)

Cerca:  nel Web  pagine in Italiano  pagine provenienti da: Italia

Web Risultati 1 - 10 su circa 104.000 per algoritmi e strutture dati. (0,53 secondi)

**Algoritmi e Strutture Dati**  
Ingegneria degli **Algoritmi** (Parte I). Contenuti: In questo modulo imparerai a progettare ed analizzare **algoritmi e strutture dati**. Docente: ...  
[gauguin.info.uniroma2.it/~italiano/.../Algoritmi/](#) - [Copia cache](#) - [Simili](#)

**Corso di Algoritmi e Strutture Dati**  
Il corso ha lo scopo di presentare i concetti fondamentali riguardanti l'analisi e il progetto di **algoritmi e strutture dati** efficienti. ...  
[www.dit.univr.it/~montreso/asd/index.shtml](#) - [Copia cache](#) - [Simili](#)

**Sito "Algoritmi e strutture dati - Progetto di algoritmi e ..."**  
Progetto di **algoritmi e strutture dati** in Java. Camil Demetrescu, Irene Finocchi, Giuseppe F. Italiano, Umberto Ferraro Petrillo ...  
[www.ateneonline.it/demetrescu/](#) - [Copia cache](#) - [Simili](#)

**Sito "Algoritmi e strutture dati 2/ed", Camil Demetrescu, Irene ...**  
Questo libro offre un'introduzione allo studio degli **algoritmi e delle strutture dati**, cercando di conciliare comprensibilità, chiarezza di esposizione e ...  
[www.ateneonline.it/demetrescu2e/homeA.asp](#) - [Copia cache](#) - [Simili](#)

**Appunti di Algoritmi E Strutture Dati - Studenti.it**  
26 feb 2007 ... Tutte le risorse per **algoritmi e strutture dati**. Tantissimi riassunti e tesine appunti completamente gratis.  
[www.studenti.it/appunti/.../algoritmi-e-strutture-dati/](#) - [Copia cache](#) - [Simili](#)

**Algoritmi e strutture dati**  
M. Torelli, Appunti per il corso di **Algoritmi e Strutture Dati** ... C. Demetrescu, I. Finocchi, G. Italiano **Algoritmi e strutture dati**. McGraw-Hill, 2004. ...  
[homes.dsi.unimi.it/~goldwurm/algo/](#) - [Copia cache](#) - [Simili](#)

**Corso di Algoritmi e Strutture Dati**  
Corso di **Algoritmi e Strutture Dati**. Dott. Giorgio Terracina. A.A. 2008/2009. (Ultimo aggiornamento: 15/06/2009). AVVISO: Risultati dell'ultimo appello ...  
[www.mat.unical.it/terracina/asd/](#) - [Copia cache](#) - [Simili](#)

**Lezioni di algoritmi e strutture dati - Maniezzo Vittorio, Margara ...**  
Libro di Maniezzo Vittorio, Margara Luciano. Lezioni di **algoritmi e strutture dati**. Pitagora.  
[www.ibs.it/codaf.../lezioni-algoritmi-strutture.html](#) - [Copia cache](#) - [Simili](#)

**Algoritmi e Strutture Dati |**  
[www.di.unipi.it/didacoc/asd1/](#) - [Copia cache](#) - [Simili](#)

# Esempio

- Purtroppo le cose vanno meno bene se  $k$  è funzione di  $n$
- Esempio: voglio calcolare il valore mediano
  - Cioè il valore che occuperebbe la posizione centrale se l'array fosse ordinato
- In questo caso  $k = n/2$ , e per valori sufficientemente grandi di  $n$  il costo è

$$\begin{aligned} O(n + k \log n) &= O(n + (n/2) \log n) \\ &= O(n \log n) \end{aligned}$$

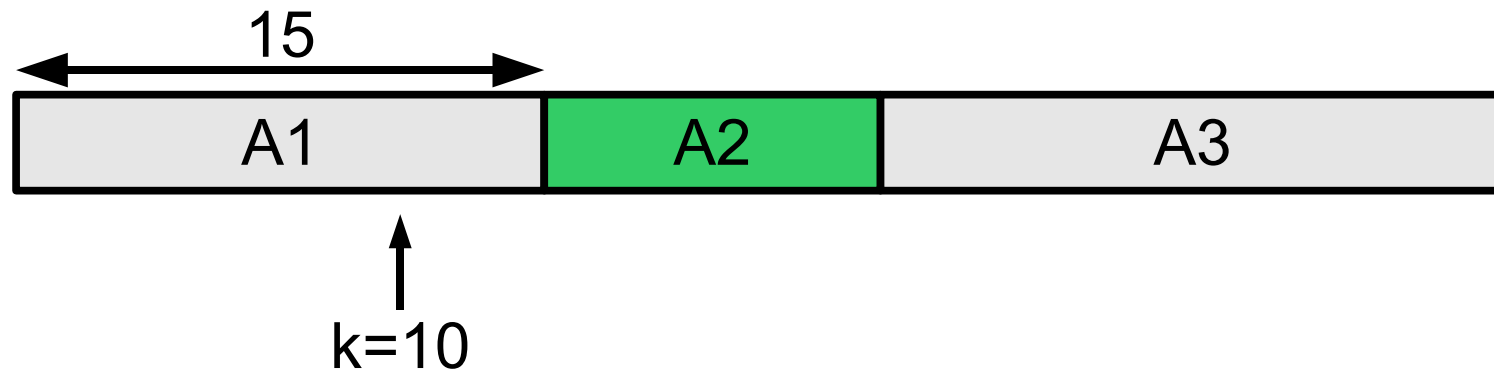
# Adattamento di quicksort al problema della selezione

```
algoritmo select1( Array A, int k ) -> elem
  scegli un elemento x in A
  A1 = {y in A: y < x }
  A2 = {y in A: y = x }
  A3 = {y in A: y > x }
  quicksort(A1);
  quicksort(A3);
  ritorna il k-esimo elemento della concatenazione di A1, A2, A3
```

- **Idea**

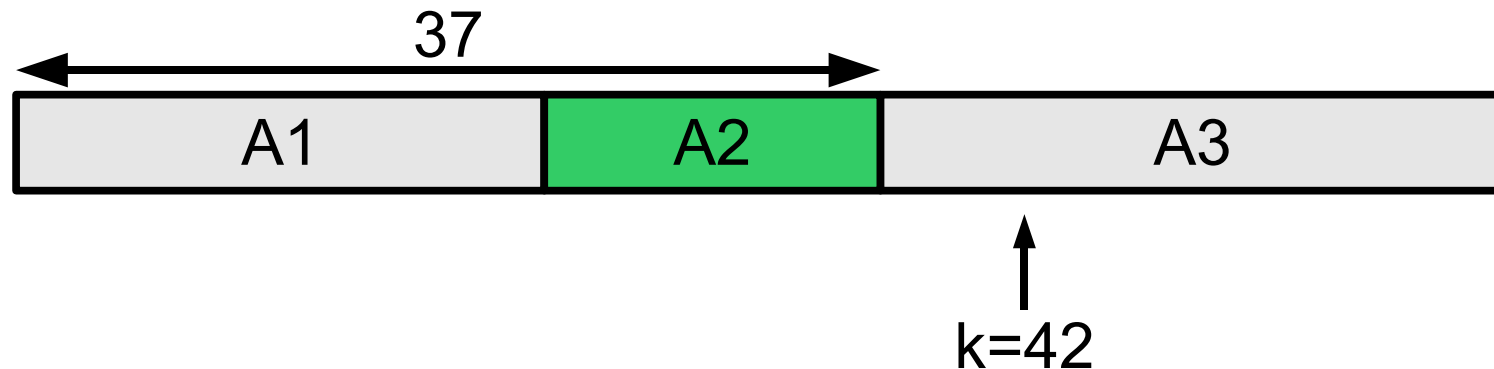
- Approccio divide-et-impera simile al quicksort
- Ma essendo un problema di ricerca, non è necessario cercare in entrambe le partizioni, basta cercare in una sola di esse

# Adattamento di quicksort al problema della selezione



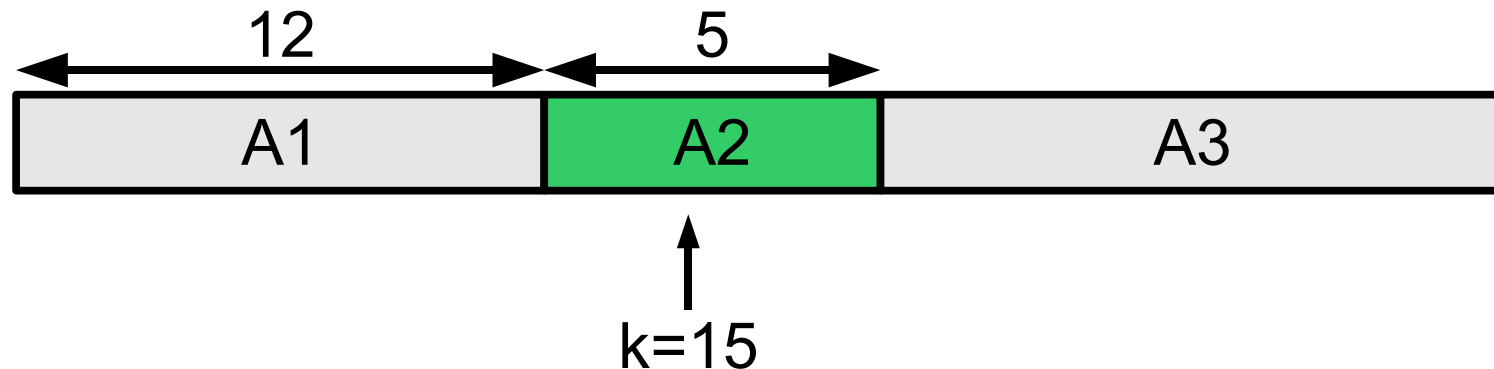
- In realtà non serve considerare tutto il vettore!
  - Supponiamo di cercare il 10mo minimo;
  - Supponiamo che A1 abbia 15 elementi
  - Il valore cercato sicuramente sta in A1

# Adattamento di quicksort al problema della selezione



- In realtà non serve considerare tutto il vettore!
  - Supponiamo di cercare il 42mo minimo;
  - Supponiamo che A1 e A2 abbiano 37 elementi
  - Il valore cercato è il  $(42-37)=5$  di A3

# Adattamento di quicksort al problema della selezione

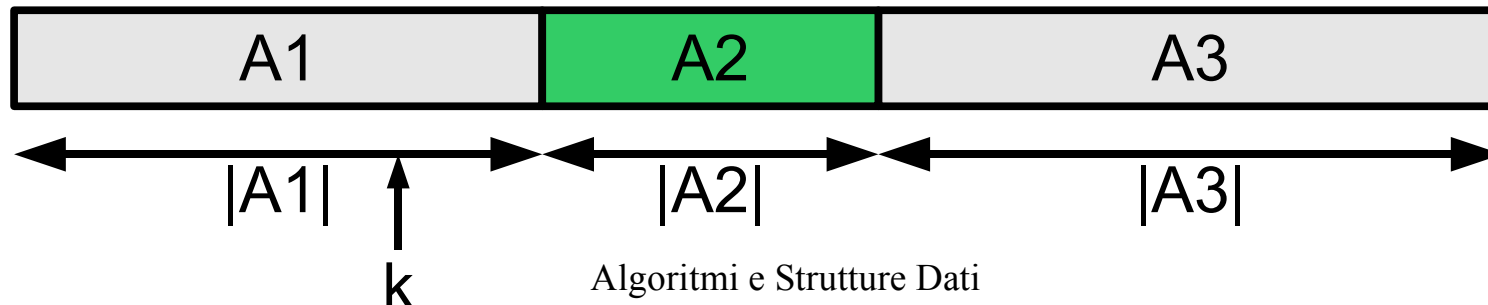


- In realtà non serve considerare tutto il vettore!
  - Supponiamo di cercare il 15mo minimo;
  - Supponiamo che A1 abbia 12 elementi e A2 ne abbia 5
  - Il valore cercato si trova in A2

# Algoritmo quickSelect()

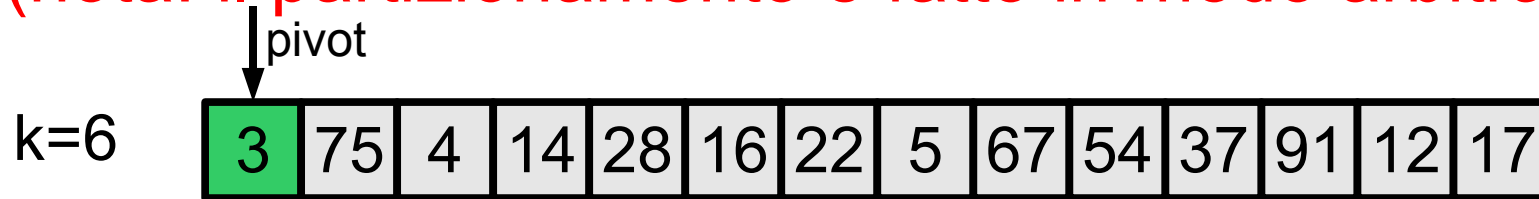
```
algoritmo quickSelect( Array A, int k ) -> elem
scegli un elemento x in A
A1 = {y in A: y < x }
A2 = {y in A: y = x }
A3 = {y in A: y > x }
if ( k <= |A1| ) then
    return quickSelect( A1, k )
else
    if ( k > |A1| + |A2| ) then
        return quickSelect( A3, k - |A1| - |A2| );
    else
        return x;
    endif
endif
```

A1, A2, e A3 possono essere determinati con l'algoritmo della "bandiera nazionale"

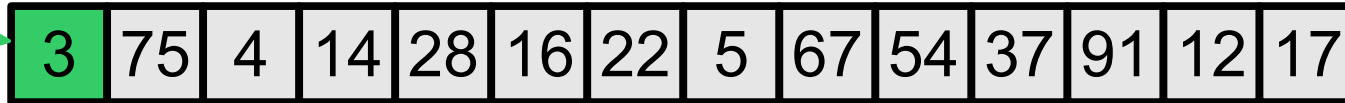


# Esempio

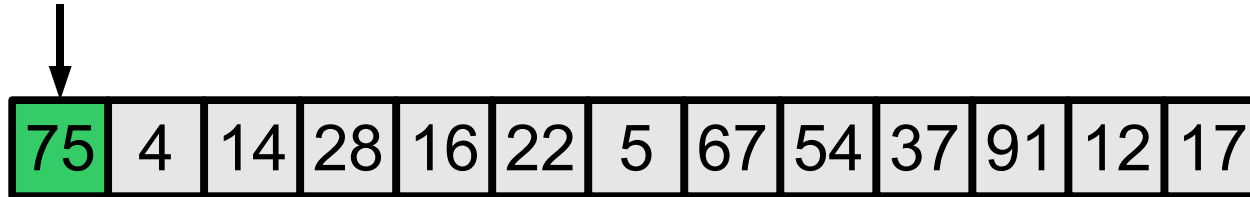
(nota: il partizionamento è fatto in modo arbitrario)



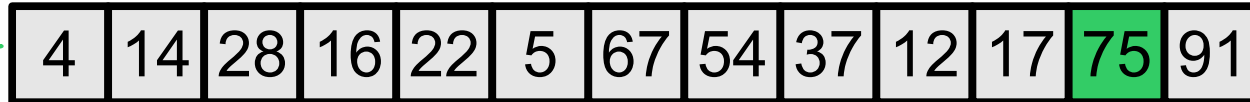
Cerchiamo il k=5 in A3



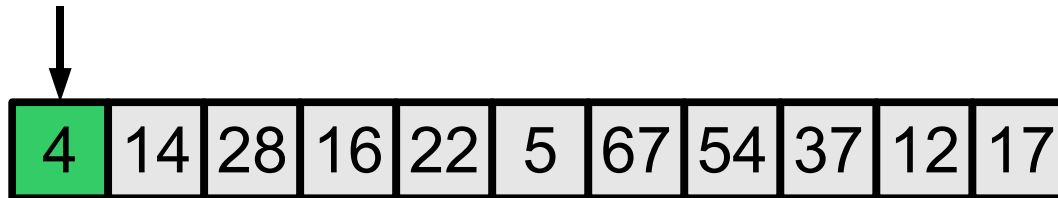
k=5



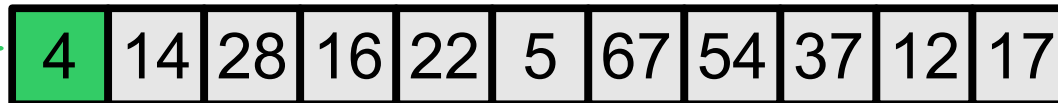
Cerchiamo il k=5 in A1



k=5

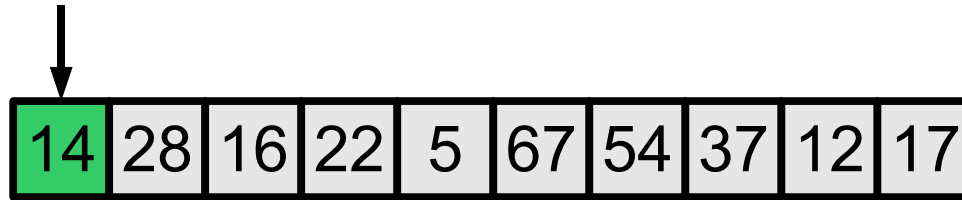


Cerchiamo il k=4 in A3

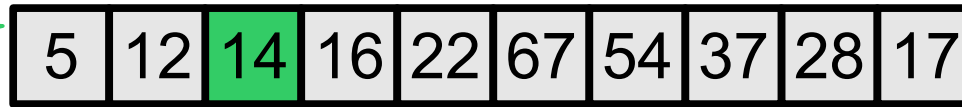


# Esempio (cont.)

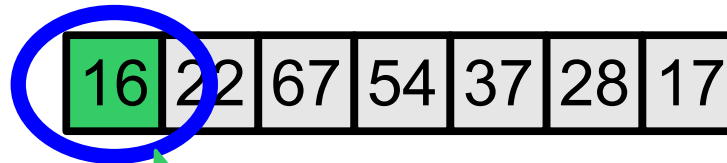
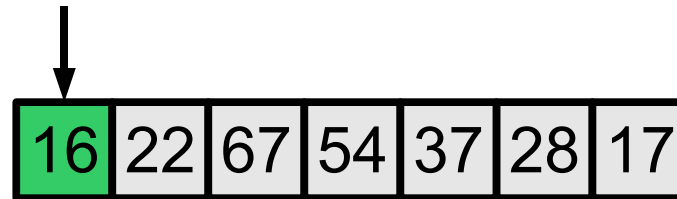
k=4



Cerchiamo  
il k=1 in A3



k=1

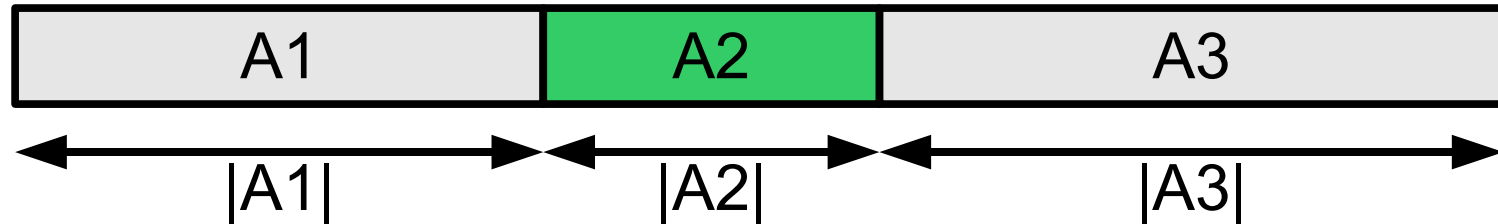


Trovato! il sesto  
elemento e' 16

# Analisi dell'algoritmo quickSelect()

- Costo nel caso ottimo
  - $T(n) = T(n/2) + \Theta(n) = \Theta(n)$
  - (Caso (3) del Master Theorem)
- Costo nel caso pessimo
  - $T(n) = T(n-1) + \Theta(n) = \Theta(n^2)$
  - (Dimostrazione come nel caso di Quick Sort)
- ...e il costo nel caso medio?

# Analisi del caso medio



- Ad ogni iterazione si elimina, nell'ipotesi peggiore, una parte di vettore di lunghezza  $|A2| + \min(|A1|, |A3|)$ 
  - Eliminiamo A2 perché altrimenti l'algoritmo sarebbe terminato;
  - Nell'ipotesi peggiore, scartiamo il sottovettore più corto tra A1 e A3
- Il numero di elementi scartati è un qualsiasi numero compreso tra 1 e  $n/2$
- La probabilità di ricorrere su un sottovettore di lunghezza  $i$  è  $\approx 1/(n/2) = 2/n$  per  $i=n/2, n/2+1, \dots, n-1$

# Analisi del caso medio

- Si ha la seguente relazione di ricorrenza per esprimere il numero  $T(n)$  di confronti richiesti:

$$T(n) = n - 1 + \frac{2}{n} \sum_{i=n/2}^{n-1} T(i)$$

- **Teorema:** la soluzione all'equazione di ricorrenza di cui sopra è  $T(n) \leq 4n$

Costo del partizionamento  
usando la funzione `partition()`,  
la stessa di Quick Sort

# Dimostrazione

- Per sostituzione, dimostriamo che  $T(n) \leq cn$  per una opportuna costante  $c$

$$\begin{aligned} T(n) &= n - 1 + \frac{2}{n} \sum_{i=n/2}^{n-1} T(i) \\ &\leq n - 1 + \frac{2}{n} \sum_{i=n/2}^{n-1} ci \\ &= n - 1 + \frac{2c}{n} \left( \sum_{i=1}^{n-1} i - \sum_{i=1}^{n/2-1} i \right) \end{aligned}$$

# Dimostrazione

$$T(n) \leq n - 1 + \frac{2c}{n} \left( \sum_{i=1}^{n-1} i - \sum_{i=1}^{n/2-1} i \right)$$

$$= n - 1 + \frac{2c}{n} \left( \frac{n^2}{2} - \frac{n^2}{8} - \frac{n}{4} \right)$$

$$= \left( 1 + \frac{3c}{4} \right) n - 1 - \frac{c}{2}$$

$$\leq \left( 1 + \frac{3c}{4} \right) n \leq cn$$

Ricordiamo che  
 $\sum_{i=1}^n i = n(n+1)/2$

- L'induzione funziona quando  $(1+3c/4) \leq c$ , ossia  $c \geq 4$

# Un algoritmo di selezione “nel mondo reale”

```
// Versione semplificata dell'implementazione dell'algoritmo di
// selezione usato nella Standard Template Library del compilatore
// g++ (/usr/include/c++/3.4.6/bits/stl_algo.h)
template<typename _RandomAccessIterator>
void nth_element(_RandomAccessIterator __first,
                _RandomAccessIterator __nth,
                _RandomAccessIterator __last)
{
    while (__last - __first > 3) {
        _RandomAccessIterator __cut =
            std::__unguarded_partition(__first, __last,
                _ValueType(std::__median(*__first,
                    *(__first+(__last-__first)/2),
                    *(__last-1) )));

        if (__cut <= __nth)
            __first = __cut;
        else
            __last = __cut;
    }
    std::__insertion_sort(__first, __last);
}
```

Riarrangia gli elementi nell'intervallo  $[\_first, \_last-1]$  in modo che al termine l'elemento che si viene a trovare in posizione  $\_nth$  è quello che occuperebbe quella posizione se il range fosse ordinato