

Divide et impera

Vittorio Maniezzo - Università di Bologna

1

1

Divide et Impera

Divide et impera:

- **Dividi:**
se l'istanza del problema da risolvere è troppo complicata per essere risolta direttamente, dividila in due o più parti
- **Risolvi** (*ricorsivamente*):
usa la stessa tecnica divide et impera per risolvere le singole parti (sottoproblemi)
- **Combina:**
combina le soluzioni trovate per i sottoproblemi in una soluzione per il problema originario.

Vittorio Maniezzo - Università di Bologna

2

2

Merge sort

- Algoritmo di **ordinamento** basato sulla tecnica *Divide et Impera*.
- Ideato da *John von Neumann* nel 1945.
- Implementato come **algoritmo di ordinamento standard** (con alcune ottimizzazioni) nelle librerie di alcuni linguaggi (Perl, Java) e alcune versioni di Linux.



Vittorio Maniezzo - Università di Bologna

3

3

MergeSort: Algoritmo

Dividi: se S contiene almeno due elementi (un solo elemento è banalmente già ordinato), rimuovi tutti gli elementi da S e inseriscili in due vettori, S_1 e S_2 , ognuno dei quali contiene circa la metà degli elementi di S .

(S_1 contiene i primi $n/2$ elementi e S_2 contiene i rimanenti $n/2$ elementi).

Risolvi: ordina gli elementi in S_1 e S_2 usando MergeSort (ricorsione).

Combina: metti insieme gli elementi di S_1 e S_2 ottenendo un unico vettore S ordinato (merge)

Vittorio Maniezzo - Università di Bologna

4

4

Merge Sort: Algoritmo

```
MergeSort(A,p,r)
  if p < r then
    q=(p+r)/2
    MergeSort(A,p,q)
    MergeSort(A,q+1,r)
    Merge(A,p,q,r)
```

Merge(A,p,q,r)

1. Rimuovi il più piccolo dei due elementi affioranti in $A[p..q]$ e $A[q+1..r]$ e inseriscilo nel vettore in costruzione.
2. Continua fino a che i due vettori sono svuotati.
3. Copia il risultato in $A[p..r]$.

Vittorio Maniezzo - Università di Bologna

5

5

Mergesort: merge

| | | | | | |
|----|----|----|----|----|----|
| 12 | 27 | 36 | 38 | 47 | 54 |
|----|----|----|----|----|----|

| | | | | | |
|---|----|----|----|----|----|
| 1 | 23 | 25 | 25 | 25 | 25 |
|---|----|----|----|----|----|

| | | | | | | | | | |
|---|----|----|--|--|--|--|--|--|--|
| 1 | 12 | 23 | | | | | | | |
|---|----|----|--|--|--|--|--|--|--|

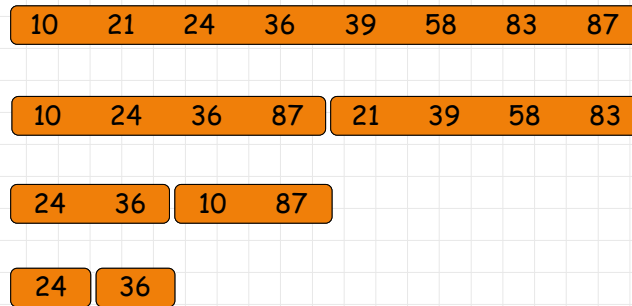
... e così via

Vittorio Maniezzo - Università di Bologna

6

6

Mergesort: dividi

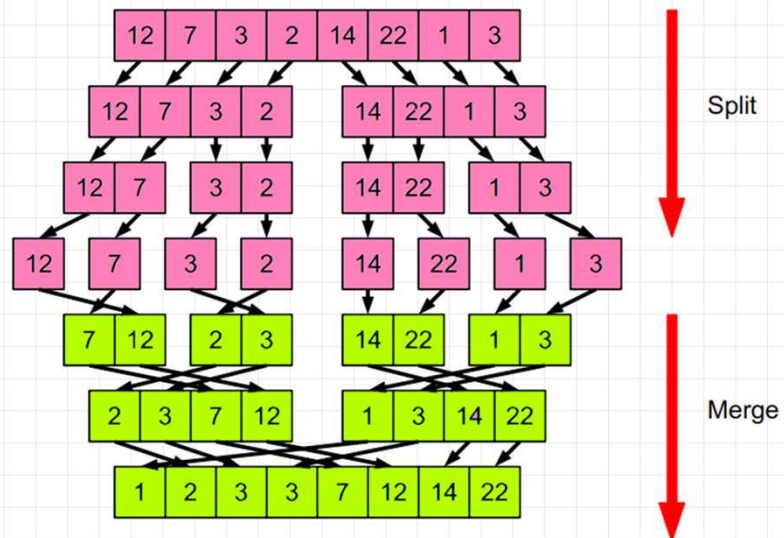


Vittorio Maniezzo - Università di Bologna

7

7

MergeSort, esempio



Vittorio Maniezzo - Università di Bologna

8

8

Merge sort, complessità

$$T(n) = 2T(n/2) + \Theta(n) = \Theta(n \log n)$$

La complessità di Merge Sort non dipende dalla configurazione iniziale dell'array da ordinare:

→ la complessità è la stessa nei casi ottimo/pessimo/medio

| | Insertion sort | Merge sort |
|--------------|----------------|---------------------|
| Caso pessimo | $\Theta(n^2)$ | $\Theta(n \log(n))$ |
| Caso medio | $\Theta(n^2)$ | $\Theta(n \log(n))$ |
| Caso ottimo | $\Theta(n)$ | $\Theta(n \log(n))$ |

Vittorio Maniezzo - Università di Bologna

9

9

Merge Sort

Costo computazionale:

- Caso **pessimo**: $\Theta(n \log n)$
- Caso **medio**: $\Theta(n \log n)$
- Caso **ottimo**: $\Theta(n \log n)$

Di solito proposto in versione **ricorsiva** e **non in place**.

Se algoritmo non in place: richiede $O(n)$ di **memoria ausiliaria**.

Vittorio Maniezzo - Università di Bologna

10

10

Merge Sort: pseudocodice

Pseudo codice dell'algoritmo MergeSort.
Ordina l'array A dalla posizione p alla r

```
1: procedure MergeSort(A, p, r)
2:   if p < r then
3:     q ← (p + r)/2
4:     MergeSort(A, p, q)
5:     MergeSort(A, q+1, r)
6:     Merge(A, p, q, r)
7:   end if
8: end procedure
```

Pseudo codice dell'algoritmo Merge.

```
1: procedure Merge(A, p, q, r)
2:   i ← p, j ← q+1, k ← 0
3:   while i ≤ q and j ≤ r do
4:     if A[i] < A[j] then
5:       B[k] ← A[i]
6:       i ← i+1
7:     else
8:       B[k] ← A[j]
9:       j ← j+1
10:    end if
11:    k ← k+1
12:  end while
13:  while i ≤ q do
14:    B[k] ← A[i]
15:    i ← i+1
16:    k ← k+1
17:  end while
18:  while j ≤ r do
19:    B[k] ← A[j]
20:    j ← j+1
21:    k ← k+1
22:  end while
23:  for k ← p to r do
24:    A[k] ← B[k - p]
25:  end for
26: end procedure
```

Leftover sx

Leftover dx

Ricopia in A

Vittorio Maniezzo - Università di Bologna

11

11

Merge sort, implementazioni

```
void mergeSortIter(int a[], int n)
{
  int* b;
  int linit, rinit, rend;
  int i, j, length, m;
  b = (int*) calloc(n, sizeof(int));
  for (length = 1; length < n; length *= 2)
  {
    for (linit = 0; linit + length < n;
         linit += length*2)
    {
      rinit = linit + length;
      rend = rinit + length;
      if (rend > n) rend = n;
      m = linit; i = linit;
      while (i < rinit)
      {
        if (a[i] < a[j])
        {
          b[m] = a[i];
          m++;
        }
        else
        {
          b[m] = a[j];
          m++;
        }
        i++; j++;
      }
      while (i < rinit) { b[m] = a[i]; i++; m++; }
      while (j < rend) { b[m] = a[j]; j++; m++; }
      for (m = linit; m < rend; m++)
        a[m] = b[m];
    }
  }
  free(b);
}
```

```
void mergeInPlace(int* a, int l, int m, int u)
{
  int i, y;
  for (; l < m; ++l)
  {
    if (a[l] < a[m])
    {
      y = a[l];
      a[l] = a[m];
      a[m] = y;
    }
  }
}
```

```
void msortInPlace(int* a, int l, int u)
{
  int m = (l+u)/2;
  if (l < u)
  {
    msortInPlace(a, l, m);
    msortInPlace(a, m+1, u);
    mergeInPlace(a, l, m, u);
  }
}
```

Non lo chiedo

In place

Iterativo

E iterativo in place?

(v. anche <https://www.codeproject.com/Articles/5275988/Fastest-sort-algorithm>)

Vittorio Maniezzo - Università di Bologna

12

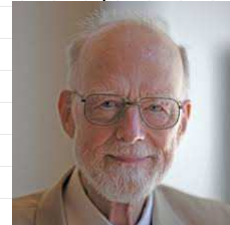
12

Quick sort

Algoritmo di ordinamento basato sulla tecnica **Divide et Impera**.

Ideato da *Tony Hoare* (o meglio, da *Sir Charles Anthony Richard Hoare*) nel 1960, in visita come studente presso la Moscow State University.

Oggetto della tesi di dottorato di Robert Sedgewick nel 1975.



Implementato come *algoritmo di ordinamento standard* nella libreria C di Unix (**qsort()**).

Vittorio Maniezzo - Università di Bologna

13

13

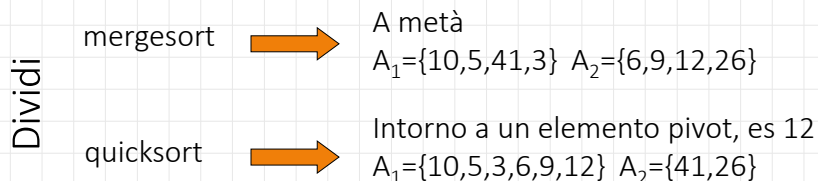
Quicksort: l'idea

Dividi: Dividi il vettore in due parti non vuote.

Risolvi: ordina le due parti ricorsivamente

Combina: fondi le due parti ottenendo un vettore ordinato.

$A = \{10, 5, 41, 3, 6, 9, 12, 26\}$



Vittorio Maniezzo - Università di Bologna

14

14

Quicksort

Quicksort(A,p,r)

if p < r then

q = partition(A,p,r)

Quicksort(A,p,q)

Quicksort(A,q+1,r)

Nota:

Mergesort lavora **dopo** la ricorsione

Quicksort lavora **prima** della ricorsione

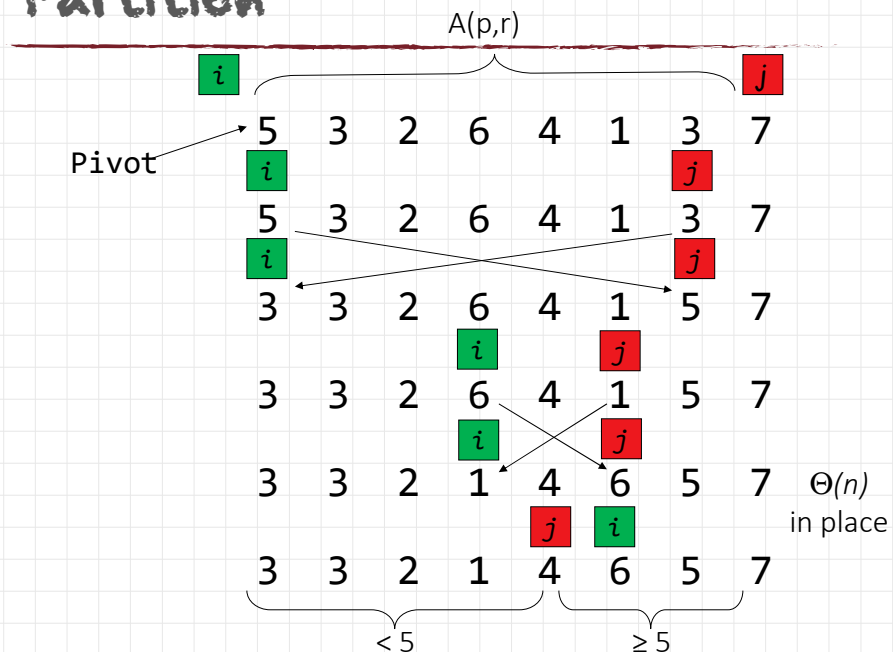
Partition è cruciale !!!

Vittorio Maniezzo - Università di Bologna

15

15

Partition

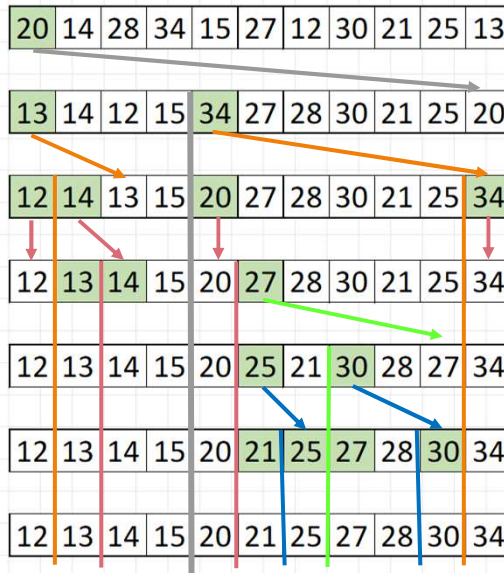


Vittorio Maniezzo - Università di Bologna

16

16

QuickSort, esempio



Vittorio Maniezzo - Università di Bologna

17

17

Quick Sort: pseudocodice

```

1: procedure QUICKSORT(A, p, r)
2:   if p < r then
3:     q ← PIVOT(A, p, r)
4:     j ← PARTITION(A, p, q, r)
5:     QUICKSORT(A, p, j)
6:     QUICKSORT(A, j+1, r)
7:   end if
8: end procedure

```

```

1: procedure SWAP(A, i, j)
2:   tmp ← A[i]
3:   A[i] ← A[j]
4:   A[j] ← tmp
5: end procedure

```

```

1: function PIVOT(A, p, r)
2:   return (p + rand()%(r-p+1)) //random in [p,...,r]
3: end function

```

```

1: function PARTITION(A, p, q, r)
2:   i ← p
3:   j ← r
4:   pivot ← A[q]
5:   SWAP(A, p, q)
6:   while i < j do
7:     while j > p and pivot ≤ A[j] do
8:       j ← j - 1
9:     end while
10:    while i < r and pivot > A[i] do
11:      i ← i + 1
12:    end while
13:    if i < j then
14:      SWAP(A, i, j)
15:    end if
16:  end while
17:  SWAP(A, p, j)
18:  return j
19: end function

```

Vittorio Maniezzo - Università di Bologna

18

18

Analisi di QS nel caso ottimo

Caso ottimo: partizioni bilanciate

$$T(n) = 2T(n/2) + \Theta(n)$$

quindi: $T(n) = \Theta(n \log(n))$

QS nel caso pessimo

Caso pessimo: partizioni sbilanciate

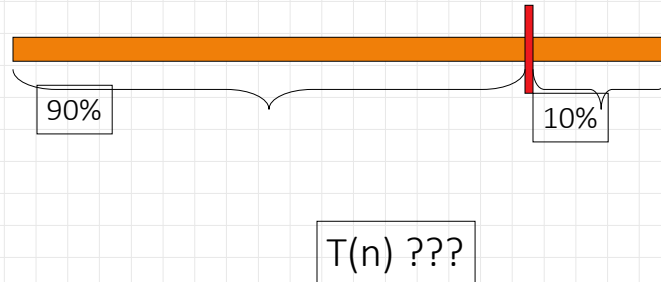
$$T(n) = T(n-1) + \Theta(n)$$

ricorsione

partition

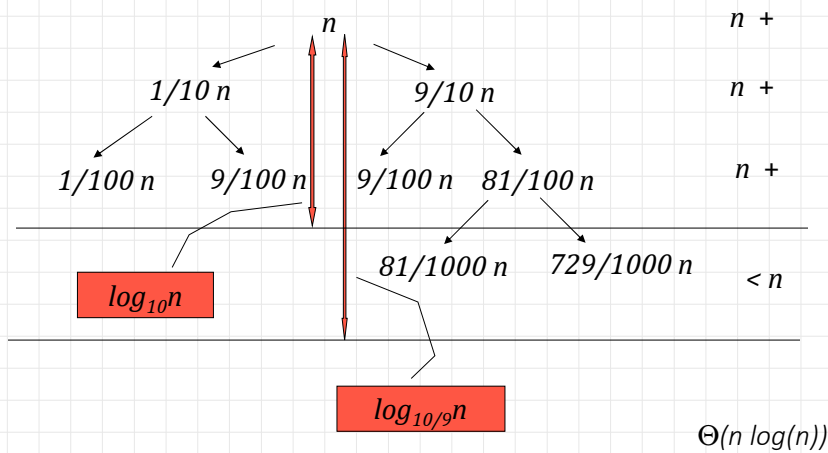
quindi: $T(n) = \Theta(n^2)$

QS nel caso ... non buono



21

Albero di ricorsione

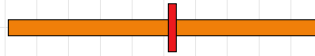


22

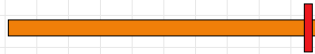
caso medio, :una intuizione.

Caso medio: a volte facciamo una buona partition a volte no...

buona partition:



cattiva partition:



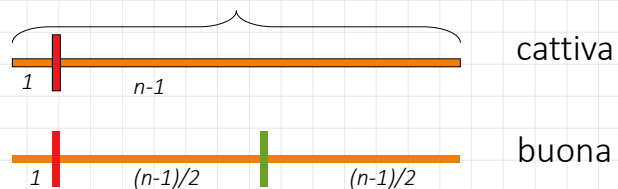
Vittorio Maniezzo - Università di Bologna

23

23

Caso medio

le buone e le cattive partition si alternano...



dopo una cattiva e una buona partizione in successione siamo più o meno nella situazione in cui la cattiva partizione non è stata fatta

Vittorio Maniezzo - Università di Bologna

24

24

QS: distribuzione degli input

Abbiamo assunto implicitamente che tutte le sequenze di numeri da ordinare fossero equiprobabili.

Se ciò non fosse vero potremmo avere costi computazionali più alti.

Possiamo “rendere gli input equiprobabili” ?

come procediamo → casualmente prima di ordinare

Scegliamo il pivot a caso.

QS randomizzato

QSR usa una versione randomizzata della procedura Partition.

```
Randomized-partition(A,p,r)
    i=random(p,r)
    exchange(A[p],A[i])
    return Partition(A,p,r)
```

Un algoritmo randomizzato non ha un input pessimo, bensì ha una sequenza di scelte pessime di pivot.

Quick Sort

Costo computazionale:

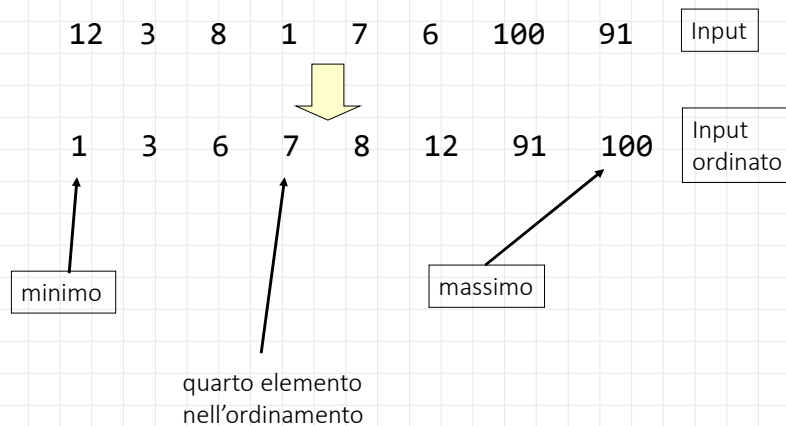
- Costo **pessimo**: $\theta(n^2)$
- Caso **medio**: $\theta(n \log n)$
- Caso **ottimo**: $\theta(n \log n)$

Anche quick sort *di solito* è *presentato in forma ricorsiva*,
ma senza utilizzare memoria ausiliaria (versione in place).

Ovviamente, se ne può fare una **versione iterativa**.

27

Selezione: esempio



28

Selezione

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

Selezione: *Calcolare l'iesimo elemento nell'ordinamento.*

Selezione del minimo/massimo $\Theta(n)$

Selezione: caso generale?

Selezione

Input: un insieme A di n numeri distinti e un numero i tra 1 e n

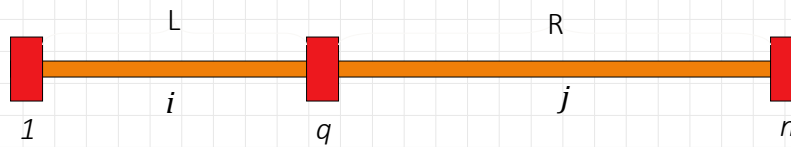
Output: l'elemento x in A maggiore di esattamente $i-1$ altri numeri in A

Soluzione banale: ordino gli elementi di A e prendo l'iesimo elemento nell'ordinamento.

$\Theta(n \log(n))$ (analisi del caso pessimo)

Selezione

$A[1..n]$



Supponiamo che
 $A[i] \leq A[j]$ per ogni $1 \leq i \leq q$ e per ogni $q < j \leq n$

Domanda:
il k-esimo elemento nell'ordinamento sta in L o in R ?

Risposta:
Facile. Se $k \leq q \rightarrow L$, altrimenti R.

Con partition posso ricondurmi sempre in $O(n)$ a questo caso.

Vittorio Maniezzo - Università di Bologna

31

31

Selezione in tempo medio lineare

```
RAND-SELECT(A,p,r,i)
  if (p=r) then return A[p]
  q = RAND-PARTITION(A,p,r)
  k = q-p+1
  if i ≤ k
    then return RAND-SELECT(A,p,q,i)
    else return RAND-SELECT(A,q+1,r,i-k)
```

caso pessimo $\Theta(n^2)$
caso medio $\Theta(n)$ (senza dimostrazione)

Vittorio Maniezzo - Università di Bologna

32

32

Selezione in tempo lineare nel caso pessimo

Bisogna evitare il caso pessimo, in cui si separa sempre un solo elemento (o un numero indipendente da n).

Necessario un buon algoritmo di partition, che garantisca di lasciare un numero sufficiente di elementi in ogni partizione

$$\varepsilon n$$

$$(1 - \varepsilon)n$$

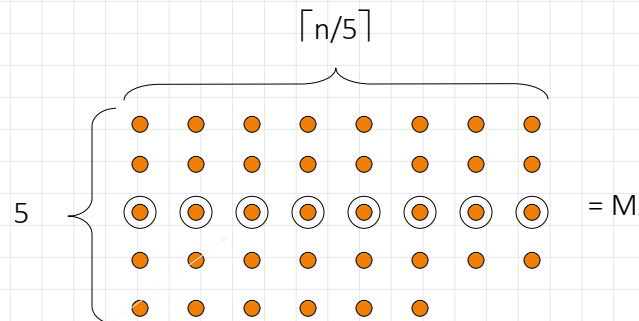
Nota: basta che ε sia maggiore di zero e indipendente da n

Select lineare

Select(A, i)

1. Dividi i numeri in input in **gruppi da 5** elementi ciascuno.
2. **Ordina ogni gruppo** (qualsiasi metodo va bene). Trova il **mediano in ciascun gruppo**.
3. Usa *Select* ricorsivamente per trovare il **mediano dei mediani**. Lo chiamiamo x .
4. **Partiziona** il vettore in ingresso usando x ottenendo due vettori A' e A'' di lunghezza k e $n-k$.
5. Se $i \leq k$ allora *Select*(i) sul vettore A' altrimenti *Select*($i-k$) sul vettore A'' .

Select lineare



Mediano della terza colonna

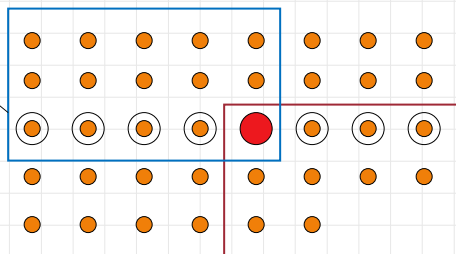
Calcoliamo il mediano di M usando Select ricorsivamente

35

Select lineare

Supponiamo di aver riordinato le colonne a seconda del valore del loro mediano.

Minori o uguali
di



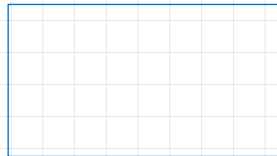
= mediano dei mediani.

Maggiori o uguali
di

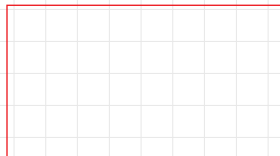


36

Select lineare



più o meno $3n/10$



più o meno $3n/10$

Se partizioniamo intorno a ● lasciamo almeno (circa) $3n/10$ elementi da una parte e almeno (circa) $3n/10$ elementi dall'altra !!! OK

37

Select: costo computazionale

$$T(n) = \begin{cases} \Theta(1) & \text{se } n < c \\ \Theta(n) + T(n/5) + T(7n/10) & \text{se } n \geq c \end{cases}$$

Costo per ordinare le colonne

Costo per calcolare il mediano dei mediani

Costo per la chiamata ricorsiva di select

$T(n) \leq k n$,
 k costante opportuna
 Dim: sarebbe col master theorem, noi niente

38