# Esercitazione 2

Quicksort

# Caratteristiche

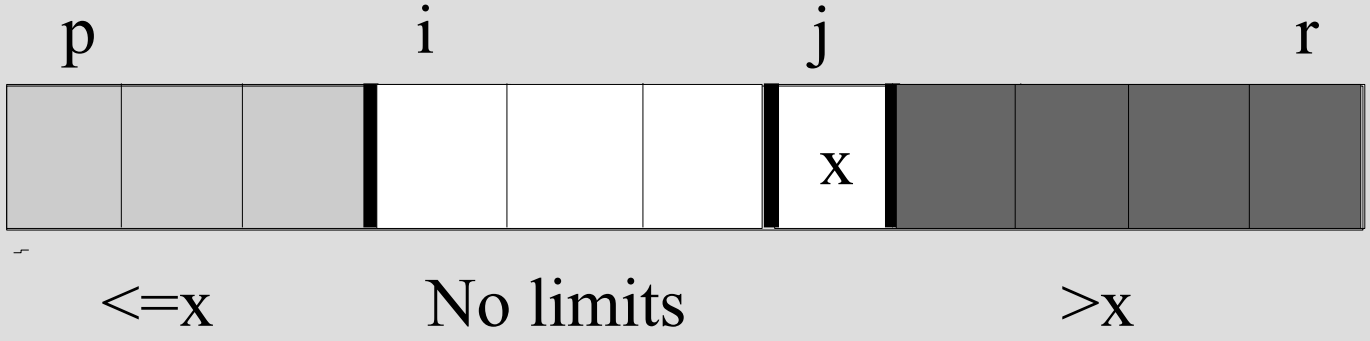
- Algoritmo di ordinamento ***mediamente molto efficiente***
- Vantaggio: ***ordinamento sul posto***
- Basato sul paradigma ***divide et impera*** (come merge sort)
- Elemento chiave: ***partizionamento***

# Divide et impera

- ***Divide***: ***(procedura Partition)*** **partiziona** l'array A[p...r] in due sotto-array A[p...q-1] e A[q+1...r] (eventualmente vuoti) tali che:
  1) Ogni elemento di A[p...q-1] sia <= A[q]
  2) Ogni elemento di A[q+1...r] sia >= A[q]
  - Il calcolo dell'**indice q** (***elemento pivot***) fa parte della procedura ***Partition*** stessa
- ***Impera***: ordina i sotto-array chiamando **ricorsivamente quicksort**
- ***Combina***: non occorre far nulla dato che i sotto-array sono ordinati sul posto

# Procedura Partition

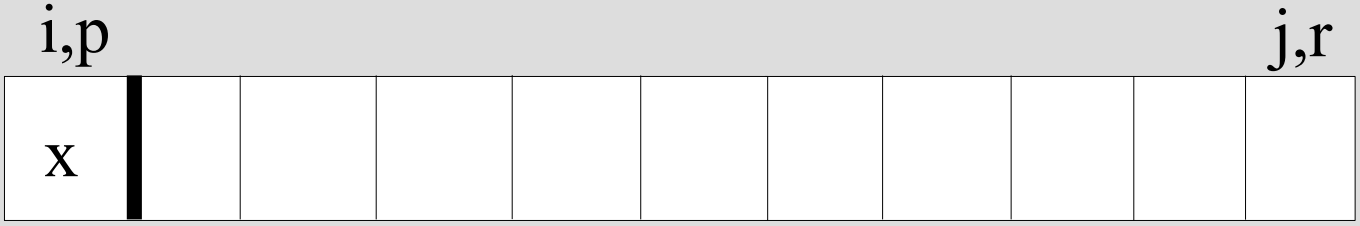
- La procedura iterativa ***Partition*** riarrangia l'array di input sul posto
  - Scelta ***elemento pivot*** *(es. x)*
  - Durante la procedura, l'array viene diviso in **4 regioni** *(eventualmente vuote)*
  - All'inizio di ogni iterazione della procedura, **ogni regione soddisfa alcune proprietà**



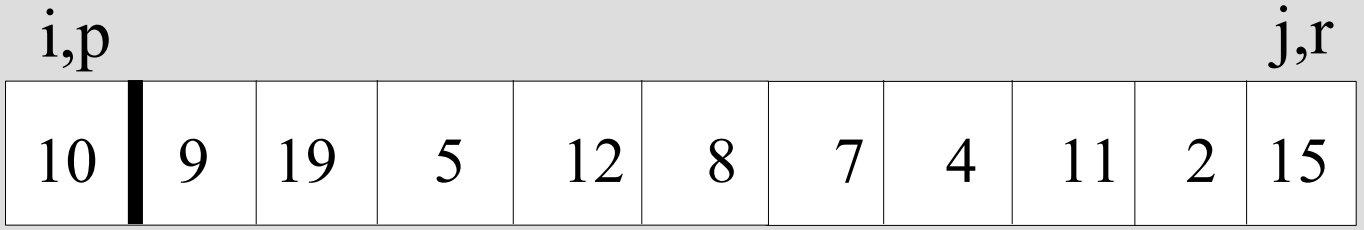
# Inizializzazione e iterazioni

***Partition (A, p, r)***

- Array A di input, indice iniziale p e indice finale r
- ***Scelta del pivot***: x=A[p]
- Utilizzo di due ***indici ausiliari*** i,j che scorrono l'array da sinistra a destra (i) e viceversa (j), confrontando gli elementi con l'elemento pivot
- L'elemento pivot "***rimbalza***" avanti e indietro finchè non si trova nella posizione corretta



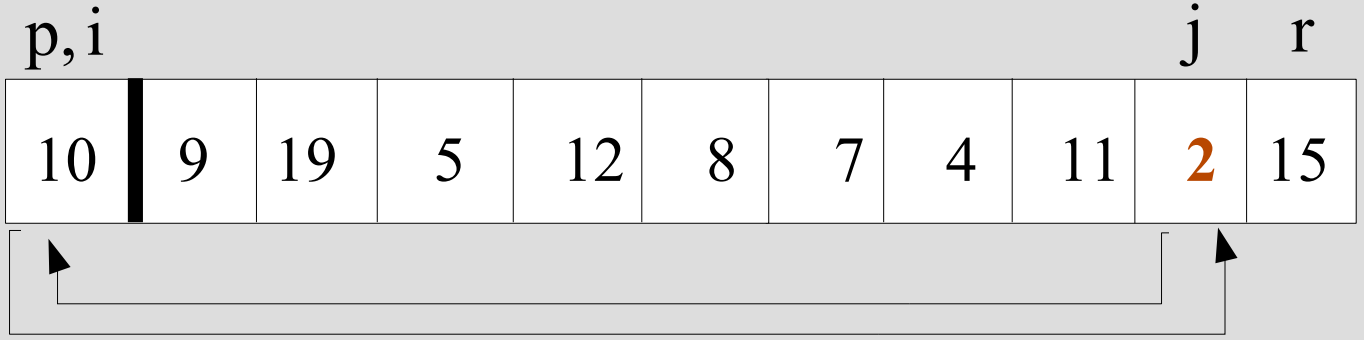
# Funzionamento




Pivot x=10

Scansione dell'array
tramite decremento di j

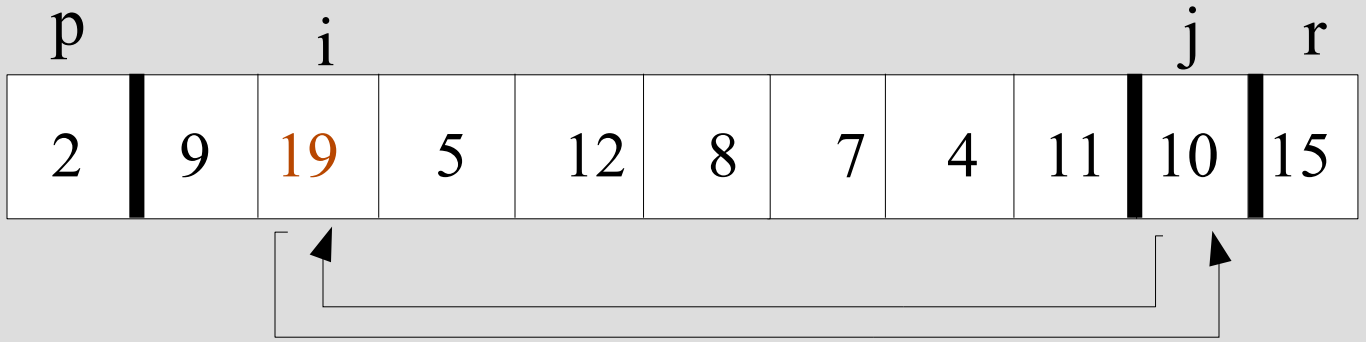
Proseguo nella scansione fino a quando A[j] >= x
Al primo elemento A[j] < x:
1) effettuo lo scambio col pivot
2) incremento i



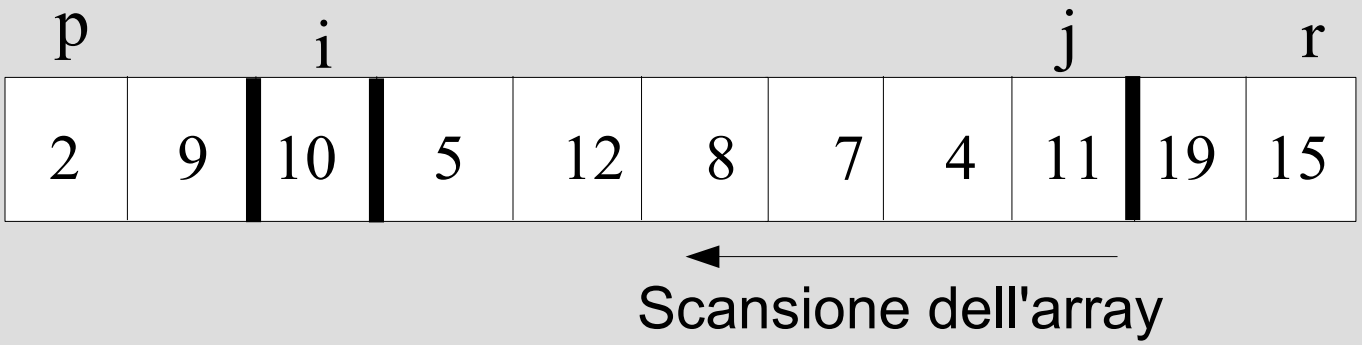
# Funzionamento


Scansione dell'array
tramite incremento di i

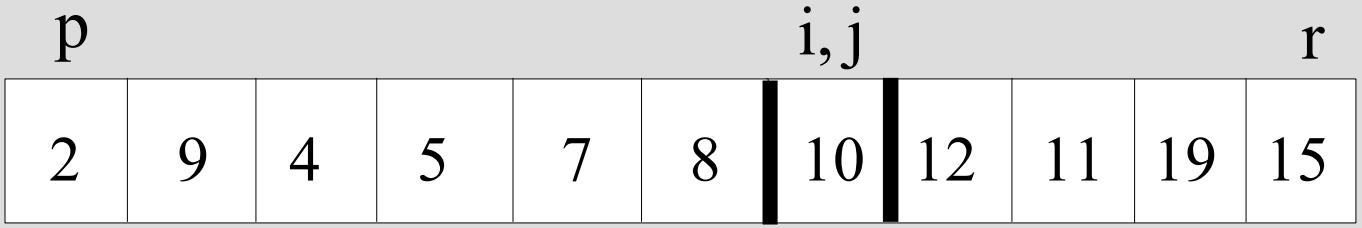
Proseguo nella scansione fino a quando A[i] <= x
Al primo elemento A[i] > x:
1) effettuo lo scambio col pivot
2) decremento j



# Funzionamento



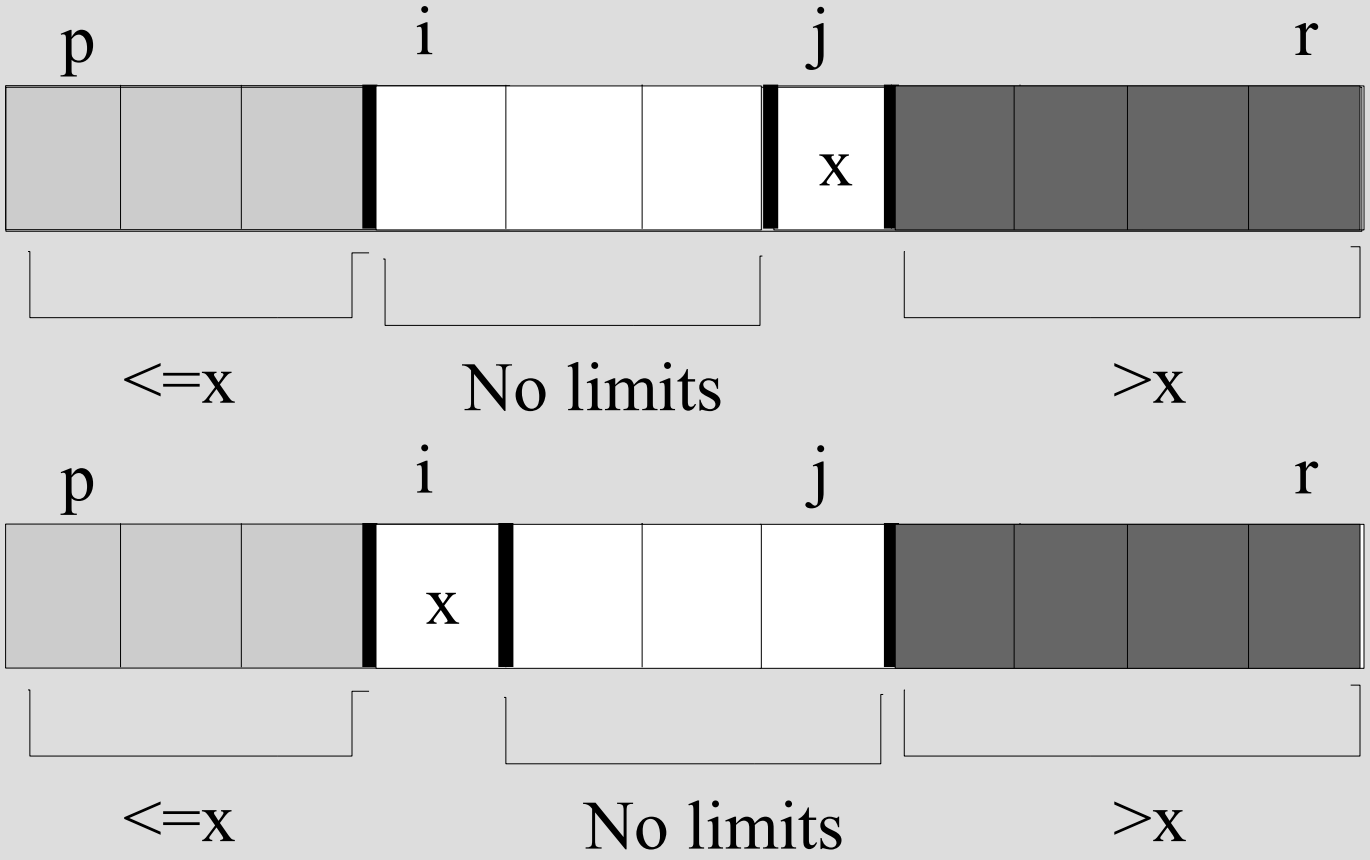

. . .




Situazione dell'array alla fine
della procedura **Partition**

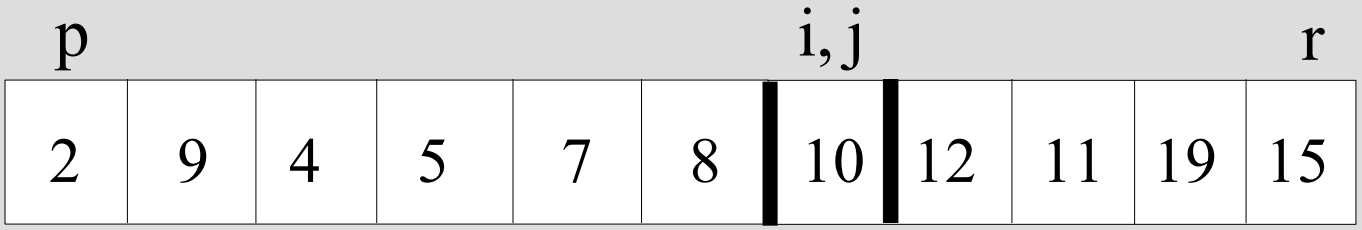
# Regioni e proprietà

2 casi possibili ad ogni iterazione



# Quicksort(A,p,r)

L'algoritmo quicksort *si richiama ricorsivamente* sulle due partizioni create dalla procedura **Partition** ai lati dell'elemento pivot




Quicksort(A, p, i-1)

Quicksort(A, i+1, r)

***Condizione di Terminaziore di Quicksort(A, p, r)?***

***p==r***

# Programma

- *quick_sort.cc*
- Programma che ordini un array di elementi in ordine non decrescente usando l'algoritmo quicksort
- Utile implementare una funzione *scambia*
- ***Suggerimento per iniziare**:* implementare la sola funzione **Partition** chiamata su un array di input riempito con numeri casuali e controllarne la funzionalità

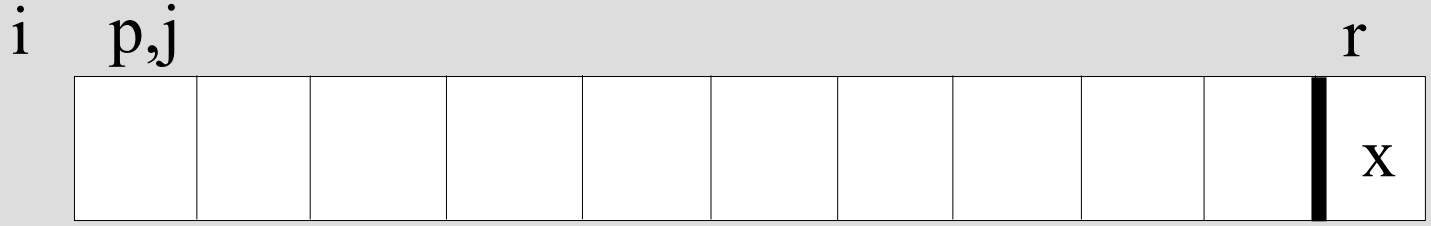
# Procedura Partition

- **Prototipo**

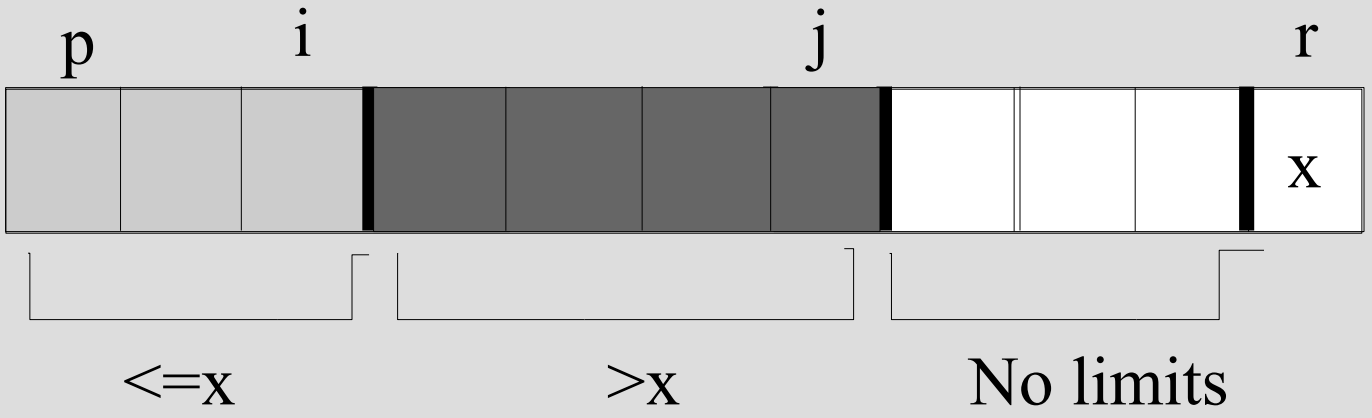
  int partition(int a[], int p, int r)
- p ed r indicano rispettivamente gli indici del primo e dell'ultimo elemento dell'array a[] da considerare
- il **valore di ritorno** della funzione indica la nuova posizione del pivot dopo la partizione degli elementi

# Procedura alternativa di Partition

- Situazione di partenza




- Ad ogni iterazione



# Procedura alternativa di Partition

- L'indice j scorre l'array in avanti
- Prosegue nella scansione fino a quando A[j] >= x
- Al primo elemento A[j] < x
  - incremento i
  - effettuo lo scambio tra A[i] e A[j]
- Al termine della scansione effettuata attraverso j (j==r), scambio A[i+1] col pivot
- i indica la posizione sentinella, di margine tra le prime due aree del vettore

# Prestazioni di quicksort

- Le prestazioni dell'algoritmo dipendono dal fatto che il ***partizionamento*** dell'array sia ***bilanciato o sbilanciato***
- **Dipendenza delle prestazioni dalla scelta dell'elemento pivot**
- ***Caso peggiore***: ogni partizione produce un sottoproblema completamente sbilanciato (n-1 e 0 elementi)

  - **Caso di array inizialmente ordinato**

# Soluzione

- Versione ***randomizzata*** di quicksort
- Scelta random dell'elemento pivot all'interno dell'array

  *i ← random(p,r)*

  *A[p] ↔ A[i]  //**Scambio** dell'elemento!*

  *Partition(A, p, r)*
- **Misurare le prestazioni** della versione randomizzata e di quella deterministica nel caso di **array iniziale ordinato**