

Algoritmi e Strutture Dati

QuickSort

QuickSort

Algoritmo di ordinamento “*sul posto*” che ha tempo di esecuzione che *teoricamente* è:

- $O(n^2)$ nel *caso peggiore*
- $O(n \log n)$ nel *caso medio*

Nonostante le cattive prestazioni nel *caso peggiore*, rimane il miglior algoritmo di ordinamento *in media*

QuickSort

È basato sulla metodologia Divide et Impera:

Dividi: L'array $A[p\dots r]$ viene “**partizionato**” (**tramite spostamenti di elementi**) in due sottoarray **non vuoti** $A[p\dots q]$ e $A[q+1\dots r]$ in cui:

↗ ogni elemento di $A[p\dots q]$ è **minore o uguale** ad ogni elemento di $A[q+1\dots r]$

Conquista: i due sottoarray $A[p\dots q]$ e $A[q+1\dots r]$ vengono ordinati ricorsivamente con QuickSort

Combina: i sottoarray vengono ordinati anche reciprocamente, quindi non è necessaria alcuna combinazione. $A[p\dots r]$ è già ordinato.

Algoritmo QuickSort

```
Quick-Sort ( $A, p, r$ )
```

```
  IF  $p < r$ 
```

```
    THEN  $q = \text{Partiziona}(A, p, r)$ 
```

```
      Quick-Sort ( $A, p, q$ )
```

```
      Quick-Sort ( $A, q+1, r$ )
```

Algoritmo QuickSort

Indice mediano

```
Quick-Sort (A, p, r)
  IF p < r
    THEN q = Partiziona (A, p, r)
         Quick-Sort (A, p, q)
         Quick-Sort (A, q + 1, r)
```

q è l'indice che **divide** l'array in due **sottoarray** dove

↑ tutti gli elementi a sinistra di **q** (compreso l'elemento in posizione **q**) sono minori o uguali tutti gli elementi a destra di **q**

Algoritmo QuickSort

**Ordinamento dei
due sottoarray**

```
Quick-Sort (A, p, r)
  IF  $p < r$ 
    THEN  $q = \text{Partiziona}(A, p, r)$ 
        Quick-Sort (A, p, q)
        Quick-Sort (A, q+1, r)
```

Poiché il **sottoarray** di **sinistra** contiene elementi tutti **minori o uguali** a tutti quelli del **sottoarray** di **destra**, **ordinare** i due sottoarray **separatamente** fornisce la **soluzione del problema**

Algoritmo QuickSort

passo Dividi

```
Quick-Sort (A, p, r)
  IF  $p < r$ 
    THEN  $q = \text{Partiziona}(A, p, r)$ 
         Quick-Sort (A, p, q)
         Quick-Sort (A, q+1, r)
```

Partition è la **chiave** di tutto l'algoritmo !

IMPORTANTE: **q** *deve* essere *strettamente minore* di **r**:

$$q < r$$

Algoritmo Partiziona

L'array $A[p\dots r]$ viene “**suddiviso**” in due sotto-array “**non vuoti**” $A[p\dots q]$ e $A[q+1\dots r]$ in cui ogni elemento di $A[p\dots q]$ è **minore o uguale** ad ogni elemento di $A[q+1\dots r]$:

⇒ l'algoritmo sceglie un valore dell'array che fungerà da elemento “**spartiacque**” tra i due sotto-array, detto valore **pivot**.

✂ sposta i **valori maggiori** del **pivot** verso l'estremità destra dell'array e i **valori minori** verso quella sinistra.

q dipenderà dal valore **pivot** scelto: sarà l'estremo della partizione a partire da sinistra nella quale, alla fine, si troveranno solo elemento **minori o uguali** al **pivot**.

Algoritmo Partiziona

```
int Partiziona (A, p, r)
    x = A[p]
    i = p - 1
    j = r + 1
    fine = false
    REPEAT
        REPEAT j = j - 1
            UNTIL A[j] ≤ x
        REPEAT i = i + 1
            UNTIL A[i] ≥ x
        IF i < j
            THEN "scambia A[i] con A[j]"
            ELSE fine = true
    UNTIL fine DO
return j
```

Algoritmo Partiziona

```
int Partiziona(A, p, r)
```

```
  x = A[p]
```

```
  i = p - 1
```

```
  j = r + 1
```

```
  fine = false
```

```
  REPEAT
```

```
    REPEAT j = j
```

```
      UNTIL A[j]
```

```
    REPEAT i = i + 1
```

```
      UNTIL A[i] ≥ x
```

```
    IF i < j
```

```
      THEN "scambia A[i] con A[j]"
```

```
      ELSE fine = true
```

```
    UNTIL fine DO
```

```
  return j
```

Elemento Pivot

**Gli elementi minori o uguali al
Pivot verranno spostati tutti
verso sinistra**

**Gli elementi maggiori o uguali
al Pivot verranno spostati tutti
verso destra**

Algoritmo Partiziona

```
int Partiziona (A, p, r)
  x = A[p]
  i = p - 1
  j = r + 1
  fine = false
  REPEAT
    REPEAT j = j - 1
      UNTIL A[j] ≤ x
    REPEAT i = i + 1
      UNTIL A[i] ≥ x
    IF i < j
      THEN "scambia A[i] con A[j]"
      ELSE fine = true
  UNTIL fine DO
return j
```

*Il ciclo continua finoché
i incrocia j*

Algoritmo Partiziona

```
int Partiziona (A, p, r)
  x = A[p]
  i = p - 1
  j = r + 1
  fine = false
  REPEAT
    REPEAT j = j - 1
      UNTIL A[j] ≤ x
    REPEAT i = i + 1
      UNTIL A[i] ≥ x
    IF i < j
      THEN "scambia A[i] con A[j]"
      ELSE fine = true
  UNTIL fine DO
return j
```

Cerca il primo elemento da destra che sia minore o uguale al Pivot x

Algoritmo Partiziona

```
int Partiziona (A, p, r)
  x = A[p]
  i = p - 1
  j = r + 1
  fine = false
  REPEAT
    REPEAT j = j - 1
      UNTIL A[j] ≤ x
    REPEAT i = i + 1
      UNTIL A[i] ≥ x
    IF i < j
      THEN "scambia A[i] con A[j]"
      ELSE fine = true
  UNTIL fine DO
  return j
```

Cerca il primo elemento da sinistra che sia maggiore o uguale al Pivot x

Algoritmo Partiziona

```
int Partiziona (A, p, r)
  x = A[p]
  i = p - 1
  j = r + 1
  fine = false
  REPEAT
    REPEAT j = j - 1
      UNTIL A[j] ≤ x
    REPEAT i = i + 1
      UNTIL A[i] ≥ x
    IF i < j
      THEN "scambia A[i] con A[j]"
      ELSE fine = true
  UNTIL fine DO
  return j
```

Se l'array non è stato scandito completamente $i < j$ (i due non indici si incrociano) allora :

- $A[i] \leq x$
- $A[j] \geq x$

gli elementi vengono scambiati



IF $i < j$

THEN "scambia A[i] con A[j]"

ELSE fine = true

Algoritmo Partiziona

```
int Partiziona (A, p, r)
  x = A[p]
  i = p - 1
  j = r + 1
  fine = false
  REPEAT
    REPEAT j = j - 1
      UNTIL A[j] ≤ x
    REPEAT i = i + 1
      UNTIL A[i] ≥ x
    IF i < j
      THEN "scambia A[i] con A[j]"
      ELSE fine = true
    UNTIL fine DO
  return j
```

Se l'array è stato scandito completamente $i \geq j$ (i due indici si incrociano) allora termina il ciclo

Algoritmo Partiziona

```
int Partiziona(A, p, r)
    x = A[p]
    i = p - 1
    j = r + 1
    fine = false
    REPEAT
        REPEAT j = j - 1
            UNTIL A[j] ≤ x
        REPEAT i = i + 1
            UNTIL A[i] ≥ x
        IF i < j
            THEN "scambia A[i] con A[j]"
            ELSE fine = true
    UNTIL fine DO
    return j
```

*Alla fine j è ritornato
come indice mediano
dell'array*

Algoritmo Partiziona: $\text{partiziona}(A, 1, 12)$

```
int Partiziona(A, p, r)
```

```
  x = A[p]
```

```
  i = p - 1
```

```
  j = r + 1
```

```
  fine = false
```

```
  REPEAT
```

```
    REPEAT j = j - 1
```

```
      UNTIL A[j] ≤ x
```

```
    REPEAT i = i + 1
```

```
      UNTIL A[i] ≥ x
```

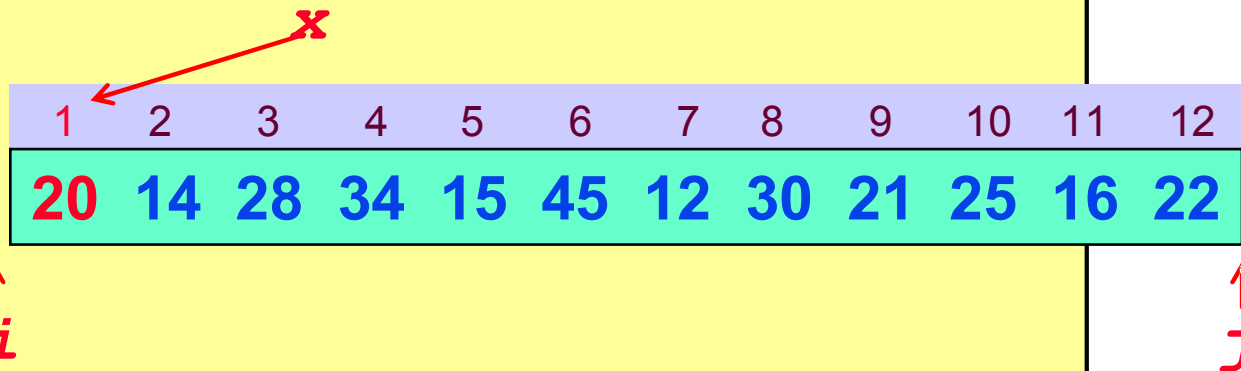
```
    IF i < j
```

```
      THEN "scambia A[i] con A[j]"
```

```
      ELSE fine = true
```

```
  UNTIL fine DO
```

```
  return j
```



Algoritmo Partiziona: $\text{partiziona}(A, 1, 12)$

```
int Partiziona(A, p, r)
```

```
  x = A[p]
```

```
  i = p - 1
```

```
  j = r + 1
```

```
  fine = false
```

```
  REPEAT
```

```
    REPEAT j = j - 1
```

```
      UNTIL A[j] ≤ x
```

```
    REPEAT i = i + 1
```

```
      UNTIL A[i] ≥ x
```

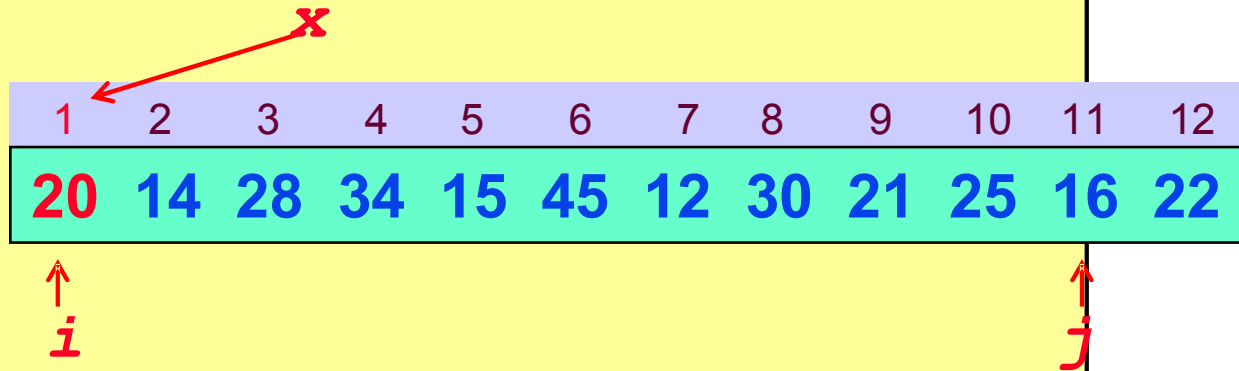
```
    IF i < j
```

```
      THEN "scambia A[i] con A[j]"
```

```
      ELSE fine = true
```

```
    UNTIL fine DO
```

```
  return j
```



Algoritmo Partiziona: `partiziona(A, 1, 12)`

```
int Partiziona(A, p, r)
```

```
  x = A[p]
```

```
  i = p - 1
```

```
  j = r + 1
```

```
  fine = false
```

```
  REPEAT
```

```
    REPEAT  $j = j - 1$ 
```

```
      UNTIL  $A[j] \leq x$ 
```

```
    REPEAT  $i = i + 1$ 
```

```
      UNTIL  $A[i] \geq x$ 
```

```
    IF  $i < j$ 
```

```
      THEN "scambia  $A[i]$  con  $A[j]$ "
```

```
    ELSE fine = true
```

```
  UNTIL fine DO
```

```
  return j
```

1	2	3	4	5	6	7	8	9	10	11	12
16	14	28	34	15	45	12	30	21	25	20	22

↑
i

↑
j

x
→

Algoritmo Partiziona: $\text{partiziona}(A, 1, 12)$

```
int Partiziona(A, p, r)
```

```
  x = A[p]
```

```
  i = p - 1
```

```
  j = r + 1
```

```
  fine = false
```

```
  REPEAT
```

```
    REPEAT j = j - 1
```

```
      UNTIL A[j] ≤ x
```

```
    REPEAT i = i + 1
```

```
      UNTIL A[i] ≥ x
```

```
    IF i < j
```

```
      THEN "scambia A[i] con A[j]"
```

```
      ELSE fine = true
```

```
  UNTIL fine DO
```

```
  return j
```

1	2	3	4	5	6	7	8	9	10	11	12
16	14	28	34	15	45	12	30	21	25	20	22

↑ *i* (at index 7) ↑ *j* (at index 7)

Algoritmo Partiziona: $\text{partiziona}(A, 1, 12)$

```
int Partiziona(A, p, r)
```

```
  x = A[p]
```

```
  i = p - 1
```

```
  j = r + 1
```

```
  fine = false
```

```
  REPEAT
```

```
    REPEAT j = j - 1
```

```
      UNTIL A[j] ≤ x
```

```
    REPEAT i = i + 1
```

```
      UNTIL A[i] ≥ x
```

```
    IF i < j
```

```
      THEN "scambia A[i] con A[j]"
```

```
      ELSE fine = true
```

```
  UNTIL fine DO
```

```
  return j
```

1	2	3	4	5	6	7	8	9	10	11	12
16	14	28	34	15	45	12	30	21	25	20	22

↑
i

↑
j

Algoritmo Partiziona: $\text{partiziona}(A, 1, 12)$

```
int Partiziona(A, p, r)
```

```
  x = A[p]
```

```
  i = p - 1
```

```
  j = r + 1
```

```
  fine = false
```

```
  REPEAT
```

```
    REPEAT j = j - 1
```

```
      UNTIL A[j] ≤ x
```

```
    REPEAT i = i + 1
```

```
      UNTIL A[i] ≥ x
```

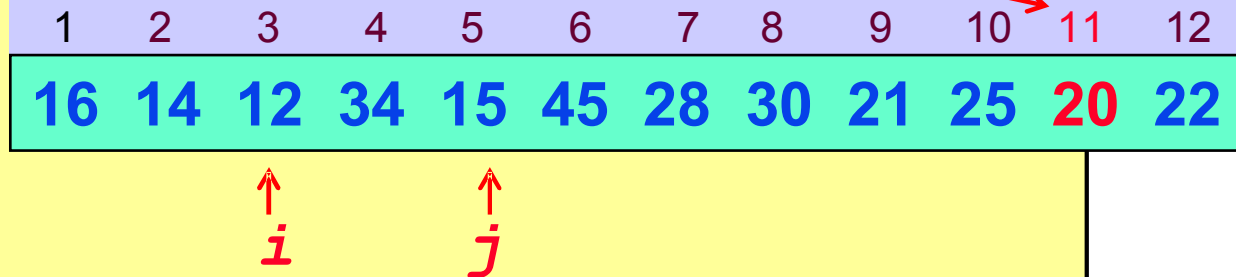
```
    IF i < j
```

```
      THEN "scambia A[i] con A[j]"
```

```
      ELSE fine = true
```

```
  UNTIL fine DO
```

```
  return j
```



Algoritmo Partiziona: $\text{partiziona}(A, 1, 12)$

```
int Partiziona(A, p, r)
```

```
  x = A[p]
```

```
  i = p - 1
```

```
  j = r + 1
```

```
  fine = false
```

```
  REPEAT
```

```
    REPEAT j = j - 1
```

```
      UNTIL A[j] ≤ x
```

```
    REPEAT i = i + 1
```

```
      UNTIL A[i] ≥ x
```

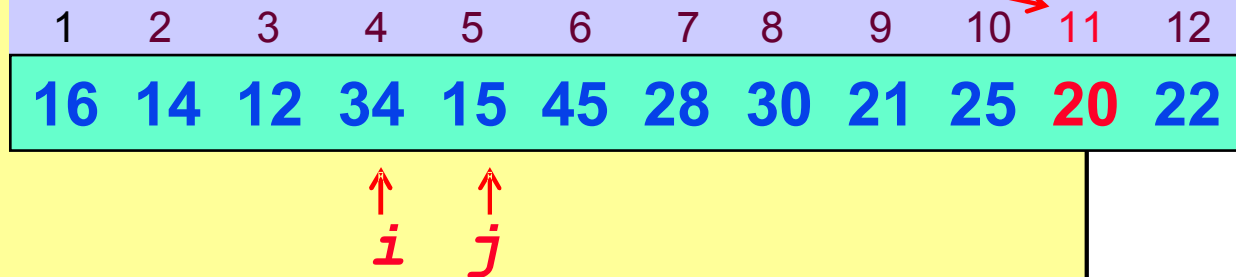
```
    IF i < j
```

```
      THEN "scambia A[i] con A[j]"
```

```
      ELSE fine = true
```

```
  UNTIL fine DO
```

```
  return j
```



Algoritmo Partiziona: $\text{partiziona}(A, 1, 12)$

```
int Partiziona(A, p, r)
```

```
  x = A[p]
```

```
  i = p - 1
```

```
  j = r + 1
```

```
  fine = false
```

```
  REPEAT
```

```
    REPEAT j = j - 1
```

```
      UNTIL A[j] ≤ x
```

```
    REPEAT i = i + 1
```

```
      UNTIL A[i] ≥ x
```

```
    IF i < j
```

```
      THEN "scambia A[i] con A[j]"
```

```
    ELSE fine = true
```

```
  UNTIL fine DO
```

```
  return j
```

1	2	3	4	5	6	7	8	9	10	11	12
16	14	12	15	34	45	28	30	21	25	20	22

↑ i ↑ j

Algoritmo Partiziona: $\text{partiziona}(A, 1, 12)$

```
int Partiziona(A, p, r)
```

```
  x = A[p]
```

```
  i = p - 1
```

```
  j = r + 1
```

```
  fine = false
```

```
  REPEAT
```

```
    REPEAT  $j = j - 1$ 
```

```
      UNTIL  $A[j] \leq x$ 
```

```
    REPEAT  $i = i + 1$ 
```

```
      UNTIL  $A[i] \geq x$ 
```

```
    IF  $i < j$ 
```

```
      THEN "scambia  $A[i]$  con  $A[j]$ "
```

```
      ELSE fine = true
```

```
    UNTIL fine DO
```

```
  return j
```

1	2	3	4	5	6	7	8	9	10	11	12
16	14	12	15	34	45	28	30	21	25	20	22

\uparrow
 j

\uparrow
 i

Algoritmo Partiziona: $\text{partiziona}(A, 1, 12)$

```
int Partiziona(A, p, r)
```

```
  x = A[p]
```

```
  i = p - 1
```

```
  j = r + 1
```

```
  fine = false
```

```
  REPEAT
```

```
    REPEAT j = j - 1
```

```
      UNTIL A[j] ≤ x
```

```
    REPEAT i = i + 1
```

```
      UNTIL A[i] ≥ x
```

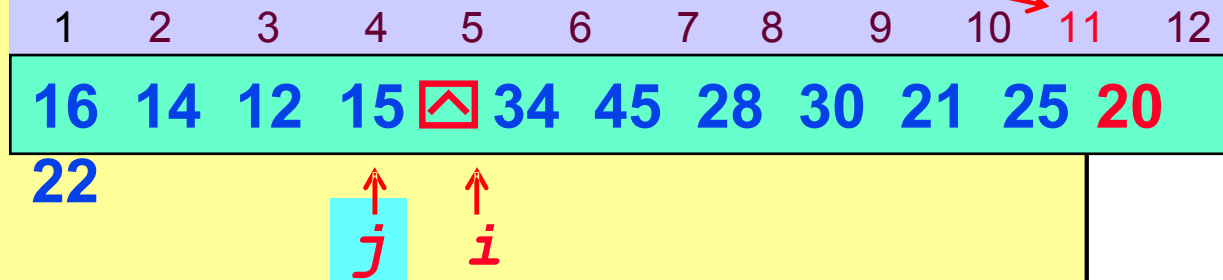
```
    IF i < j
```

```
      THEN "scambia A[i] con A[j]"
```

```
    ELSE fine = true
```

```
  UNTIL fine DO
```

```
  return j
```



Algoritmo Partiziona: casi estremi

```
int Partiziona(A, p, r)
```

```
  x = A[p]
```

```
  i = p - 1
```

```
  j = r + 1
```

```
  fine = false
```

```
  REPEAT
```

```
    REPEAT j = j - 1
```

```
      UNTIL A[j] ≤ x
```

```
    REPEAT i = i + 1
```

```
      UNTIL A[i] ≥ x
```

```
    IF i < j
```

```
      THEN "scambia A[i] con A[j]"
```

```
      ELSE fine = true
```

```
    UNTIL fine DO
```

```
  return j
```

1	2	3	4	5	6	7	8	9	10	11	12
1	14	28	34	15	45	12	30	21	25	16	22

↑
i

↑
j

Se esiste un solo elemento minore o uguale al pivot, ...

Algoritmo Partiziona: casi estremi

```
int Partiziona(A, p, r)
```

```
  x = A[p]
```

```
  i = p - 1
```

```
  j = r + 1
```

```
  fine = false
```

```
  REPEAT
```

```
    REPEAT j = j - 1
```

```
      UNTIL A[j] ≤ x
```

```
    REPEAT i = i + 1
```

```
      UNTIL A[i] ≥ x
```

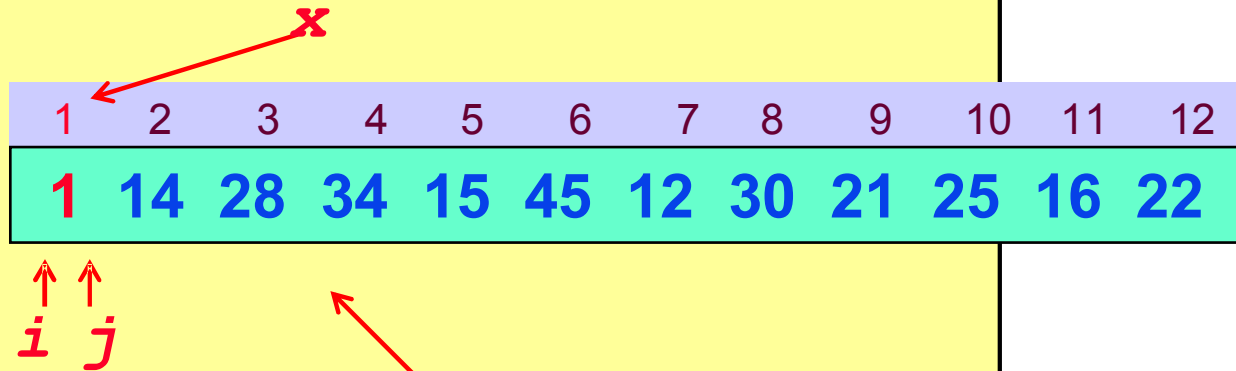
```
    IF i < j
```

```
      THEN "scambia A[i] con A[j]"
```

```
      ELSE fine = true
```

```
  UNTIL fine DO
```

```
  return j
```



Se esiste un solo elemento minore o uguale al pivot, ...

Algoritmo Partiziona: casi estremi

```
int Partiziona(A, p, r)
```

```
  x = A[p]
```

```
  i = p - 1
```

```
  j = r + 1
```

```
  fine = false
```

```
  REPEAT
```

```
    REPEAT j = j - 1
```

```
      UNTIL A[j] ≤ x
```

```
    REPEAT i = i + 1
```

```
      UNTIL A[i] ≥ x
```

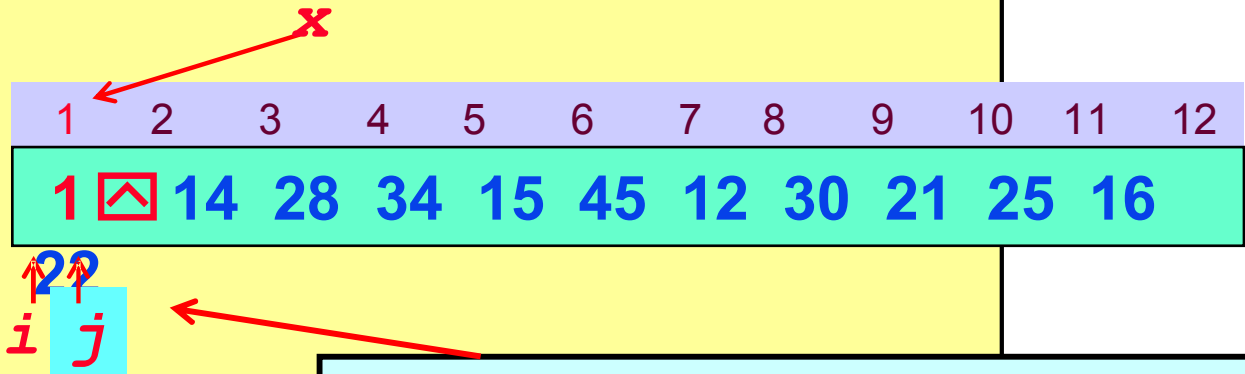
```
    IF i < j
```

```
      THEN "scambia A[i] con A[j]"
```

```
    ELSE fine = true
```

```
  UNTIL fine DO
```

```
  return j
```



Se esiste un solo elemento minore o uguale al pivot, l'array è partizionato in due porzioni: quella sinistra ha dimensione 1 e quella destra ha dimensione n-1

Algoritmo Partiziona: casi estremi

```
int Partiziona(A, p, r)
```

```
  x = A[p]
```

```
  i = p - 1
```

```
  j = r + 1
```

```
  fine = false
```

```
  REPEAT
```

```
    REPEAT j = j - 1
```

```
      UNTIL A[j] ≤ x
```

```
    REPEAT i = i + 1
```

```
      UNTIL A[i] ≥ x
```

```
    IF i < j
```

```
      THEN "scambia A[i] con A[j]"
```

```
      ELSE fine = true
```

```
  UNTIL fine DO
```

```
  return j
```

1	2	3	4	5	6	7	8	9	10	11	12
2	1	28	34	15	45	12	30	21	25	16	22

**Se esistono solo due
elementi minori o uguali al
pivot, ...**

Algoritmo Partiziona: casi estremi

```
int Partiziona(A, p, r)
```

```
  x = A[p]
```

```
  i = p - 1
```

```
  j = r + 1
```

```
  fine = false
```

```
  REPEAT
```

```
    REPEAT j = j - 1
```

```
      UNTIL A[j] ≤ x
```

```
    REPEAT i = i + 1
```

```
      UNTIL A[i] ≥ x
```

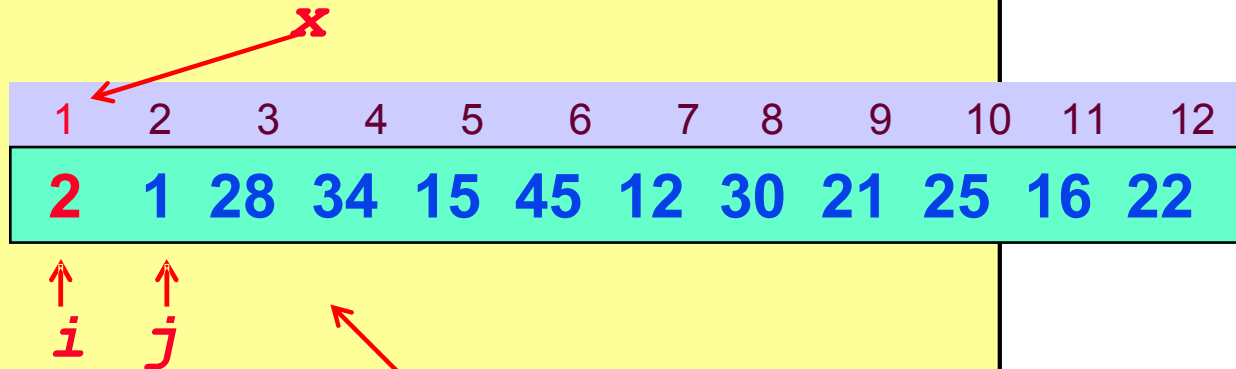
```
    IF i < j
```

```
      THEN "scambia A[i] con A[j]"
```

```
      ELSE fine = true
```

```
  UNTIL fine DO
```

```
  return j
```



**Se esistono solo due
elementi minori o uguali al
pivot, ...**

Algoritmo Partiziona: casi estremi

```
int Partiziona(A, p, r)
```

```
  x = A[p]
```

```
  i = p - 1
```

```
  j = r + 1
```

```
  fine = false
```

```
  REPEAT
```

```
    REPEAT j = j - 1
```

```
      UNTIL A[j] ≤ x
```

```
    REPEAT i = i + 1
```

```
      UNTIL A[i] ≥ x
```

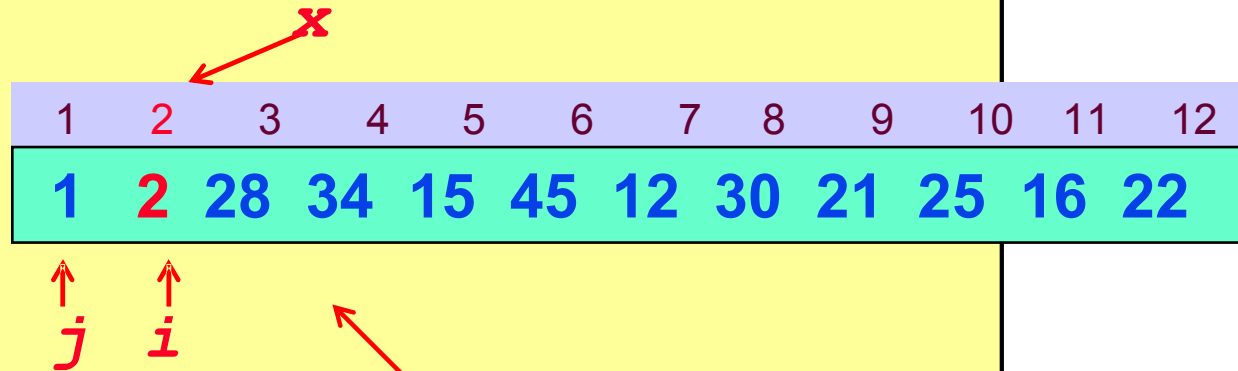
```
    IF i < j
```

```
      THEN "scambia A[i] con A[j]"
```

```
      ELSE fine = true
```

```
  UNTIL fine DO
```

```
  return j
```



**Se esistono solo due
elementi minori o uguali al
pivot, ...**

Algoritmo Partiziona: casi estremi

```
int Partiziona(A, p, r)
```

```
  x = A[p]
```

```
  i = p - 1
```

```
  j = r + 1
```

```
  fine = false
```

```
  REPEAT
```

```
    REPEAT j = j - 1
```

```
      UNTIL A[j] ≤ x
```

```
    REPEAT i = i + 1
```

```
      UNTIL A[i] ≥ x
```

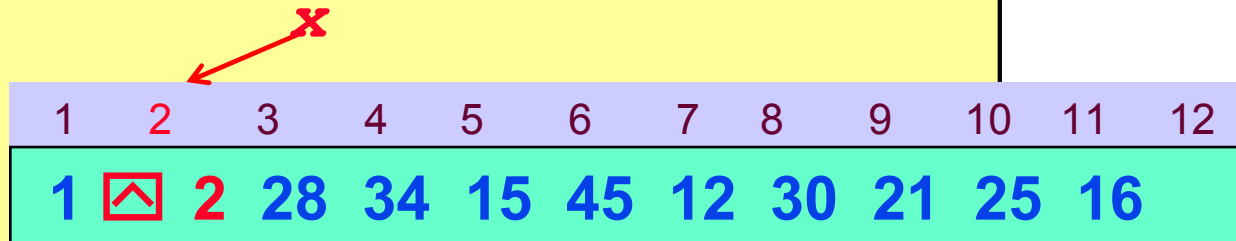
```
    IF i < j
```

```
      THEN "scambia A[i] con A[j]"
```

```
    ELSE fine = true
```

```
  UNTIL fine DO
```

```
  return j
```



22
j

i

**Se esistono solo due
elemento minori o uguali al
pivot, l'array è partizionato in
due porzioni: quella sinistra
ha dimensione 1 e quella
destra ha dimensione n-1**

Algoritmo Partiziona: casi estremi

Partiziona è quindi tale che:

SE il numero di elementi dell'array minori o uguali all'elemento $A[p]$, scelto come *pivot*, è pari a **1** (cioè $A[p]$ è l'elemento minimo) o a **2**,

ALLORA le *dimensioni* delle partizioni restituite sono **1** per la *partizione di sinistra* e **$n-1$** per *quella di destra*.

Algoritmo QuickSort

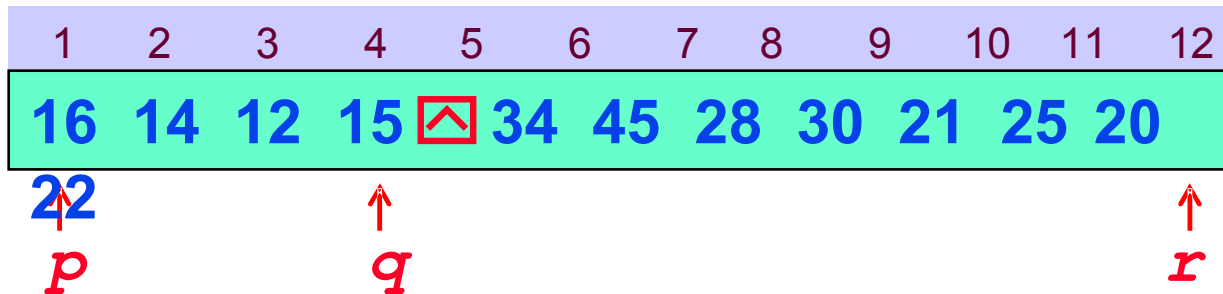
```
Quick-Sort ( $A, p, r$ )
```

```
  IF  $p < r$ 
```

```
    THEN  $q = \text{Partiziona}(A, p, r)$ 
```

```
        Quick-Sort ( $A, p, q$ )
```

```
        Quick-Sort ( $A, q+1, r$ )
```



Algoritmo QuickSort

Quick-Sort (A, p, r)

IF $p < r$

THEN $q = \text{Partiziona}(A, p, r)$

Quick-Sort (A, p, q)

Quick-Sort ($A, q+1, r$)

5	6	7	8	9	10	11	12
34	45	28	30	21	25	20	22

1	2	3	4	5	6	7	8	9	10	11	12
16	14	12	15	34	45	28	30	21	25	20	

↑
 p

↑
 r

Algoritmo QuickSort

```
Quick-Sort ( $A, p, r$ )
```

```
  IF  $p < r$ 
```

```
    THEN  $q = \text{Partiziona}(A, p, r)$ 
```

```
        Quick-Sort ( $A, p, q$ )
```

```
        Quick-Sort ( $A, q+1, r$ )
```

5	6	7	8	9	10	11	12
34	45	28	30	21	25	20	22

1	2	3	4	5	6	7	8	9	10	11	12
16	14	12	15	34	45	28	30	21	25	20	

↑
 p

↑
 r

Algoritmo QuickSort

```
Quick-Sort (A, p, r)
```

```
  IF  $p < r$ 
```

```
    THEN  $q = \text{Partiziona}(A, p, r)$ 
```

```
      Quick-Sort (A, p, q)
```

```
      Quick-Sort (A, q+1, r)
```

5	6	7	8	9	10	11	12
34	45	28	30	21	25	20	22

4
16

1	2	3	4	5	6	7	8	9	10	11	12
15	14	12	16	34	45	28	30	21	25	20	

↑
 p

↑ ↑
 q r

Algoritmo QuickSort

Quick-Sort (A, p, r)

IF $p < r$

THEN $q = \text{Partiziona}(A, p, r)$

Quick-Sort (A, p, q)

Quick-Sort ($A, q+1, r$)

5	6	7	8	9	10	11	12
34	45	28	30	21	25	20	22

4
16

1	2	3	4	5	6	7	8	9	10	11	12
15	14	12	16	34	45	28	30	21	25	20	

↑
22
 p

↑
 r

Algoritmo QuickSort

```
Quick-Sort (A, p, r)
```

```
  IF  $p < r$ 
```

```
    THEN  $q = \text{Partiziona}(A, p, r)$ 
```

```
        Quick-Sort (A, p, q)
```

```
        Quick-Sort (A, q+1, r)
```

5	6	7	8	9	10	11	12
34	45	28	30	21	25	20	22

4
16

1	2	3	4	5	6	7	8	9	10	11	12
12	14	15	16	34	45	28	30	21	25		

20 22
↑ ↑ ↑
 p q r

Algoritmo QuickSort

Quick-Sort (A, p, r)

IF $p < r$

THEN $q = \text{Partiziona}(A, p, r)$

Quick-Sort (A, p, q)

Quick-Sort ($A, q+1, r$)

5	6	7	8	9	10	11	12
34	45	28	30	21	25	20	22

4
16

1	2	3	4	5	6	7	8	9	10	11	12
12	14	15	16	34	45	28	30	21	25		

20 22
↑ ↑
 p q r

Algoritmo QuickSort

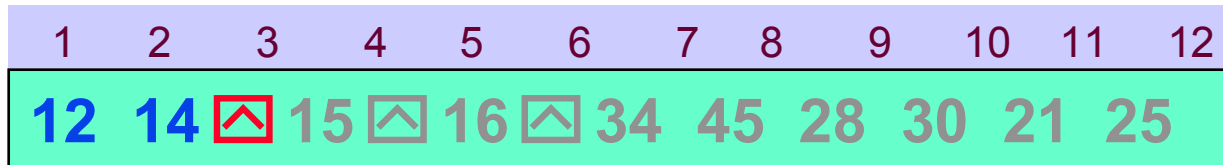
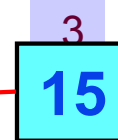
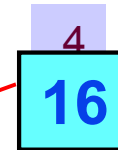
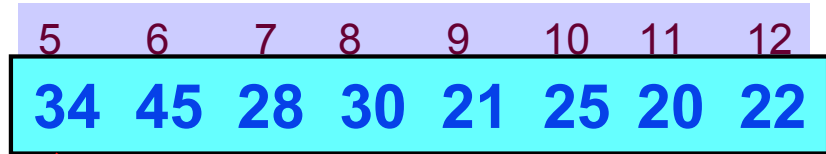
Quick-Sort (A, p, r)

IF $p < r$

THEN $q = \text{Partiziona}(A, p, r)$

Quick-Sort (A, p, q)

Quick-Sort ($A, q+1, r$)



\uparrow \uparrow
 p r

Algoritmo QuickSort

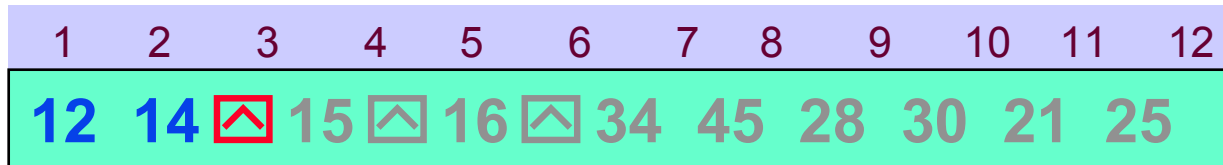
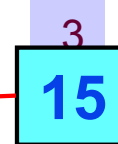
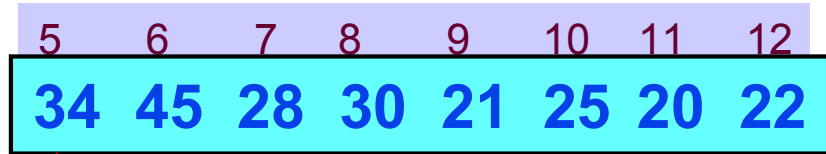
```
Quick-Sort ( $A, p, r$ )
```

```
  IF  $p < r$ 
```

```
    THEN  $q = \text{Partiziona}(A, p, r)$ 
```

```
    Quick-Sort ( $A, p, q$ )
```

```
    Quick-Sort ( $A, q+1, r$ )
```



↑ 20
 p

↑ 22
 r

Algoritmo QuickSort

Quick-Sort (A, p, r)

IF $p < r$

THEN $q = \text{Partiziona}(A, p, r)$

Quick-Sort (A, p, q)

Quick-Sort ($A, q+1, r$)

5	6	7	8	9	10	11	12
34	45	28	30	21	25	20	22

4
16

1	2	3	4	5	6	7	8	9	10	11	12
12	14	15	16	34	45	28	30	21	25	20	

22
↑
 p

↑
 q

↑
 r

Algoritmo QuickSort

Quick-Sort (A, p, r)

IF $p < r$

THEN $q = \text{Partiziona}(A, p, r)$

Quick-Sort (A, p, q)

Quick-Sort ($A, q+1, r$)

5	6	7	8	9	10	11	12
34	45	28	30	21	25	20	22

4
16

1	2	3	4	5	6	7	8	9	10	11	12
12	14	15	16	34	45	28	30	21	25	20	

22
↑ ↑
pr

Algoritmo QuickSort

Quick-Sort (A, p, r)

IF $p < r$

THEN $q = \text{Partiziona}(A, p, r)$

Quick-Sort (A, p, q)

Quick-Sort ($A, q+1, r$)

5	6	7	8	9	10	11	12
34	45	28	30	21	25	20	22

4
16

1	2	3	4	5	6	7	8	9	10	11	12
12	14	15	16	34	45	28	30	21	25	20	

22
↑↑
 pr

Algoritmo QuickSort

```
Quick-Sort ( $A, p, r$ )
```

```
  IF  $p < r$ 
```

```
    THEN  $q = \text{Partiziona}(A, p, r)$ 
```

```
        Quick-Sort ( $A, p, q$ )
```

```
        Quick-Sort ( $A, q+1, r$ )
```

5	6	7	8	9	10	11	12
34	45	28	30	21	25	20	22

1	2	3	4	5	6	7	8	9	10	11	12
12	14	15	16	34	45	28	30	21	25	20	

22

↑ ↑
 pr

Algoritmo QuickSort

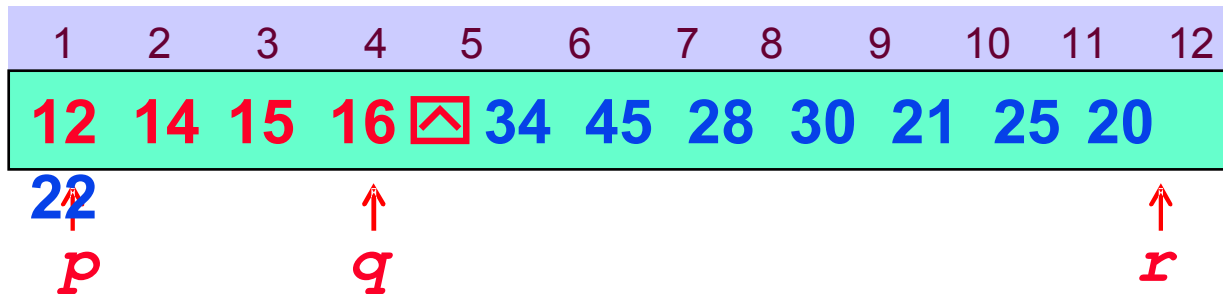
```
Quick-Sort ( $A, p, r$ )
```

```
  IF  $p < r$ 
```

```
    THEN  $q = \text{Partiziona}(A, p, r)$ 
```

```
        Quick-Sort ( $A, p, q$ )
```

```
        Quick-Sort ( $A, q+1, r$ )
```



Algoritmo QuickSort

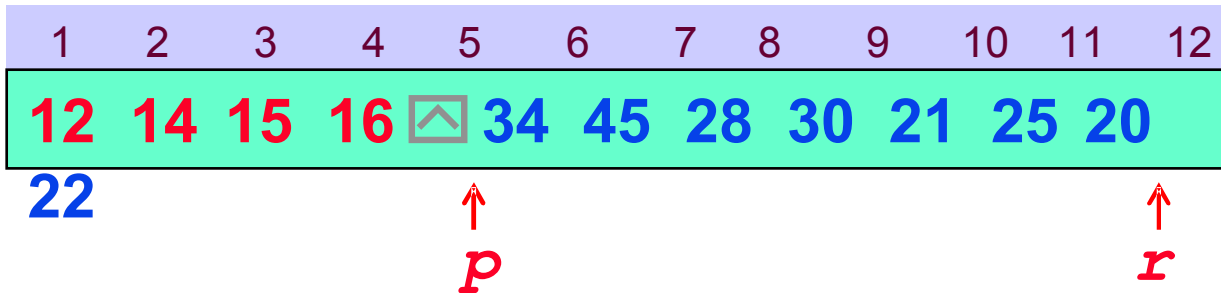
```
Quick-Sort ( $A, p, r$ )
```

```
IF  $p < r$ 
```

```
THEN  $q = \text{Partiziona}(A, p, r)$ 
```

```
Quick-Sort ( $A, p, q$ )
```

```
Quick-Sort ( $A, q+1, r$ )
```



Algoritmo QuickSort

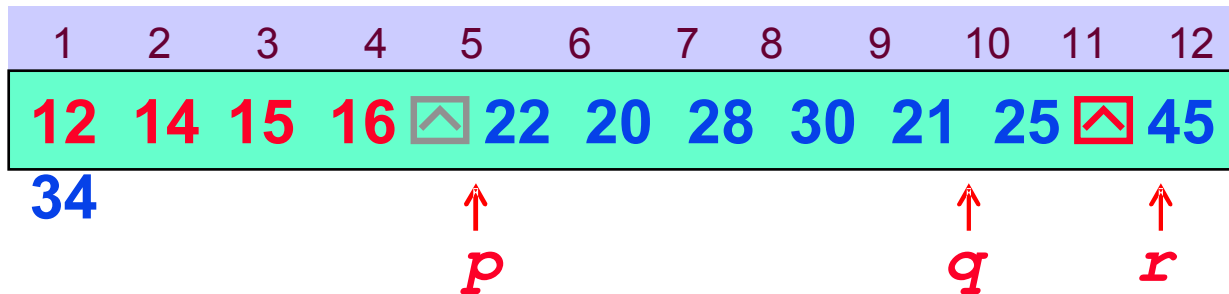
```
Quick-Sort ( $A, p, r$ )
```

```
  IF  $p < r$ 
```

```
    THEN  $q = \text{Partiziona}(A, p, r)$ 
```

```
        Quick-Sort ( $A, p, q$ )
```

```
        Quick-Sort ( $A, q+1, r$ )
```



Algoritmo QuickSort

```
Quick-Sort ( $A, p, r$ )
```

```
  IF  $p < r$ 
```

```
    THEN  $q = \text{Partiziona}(A, p, r)$ 
```

```
      Quick-Sort ( $A, p, q$ )
```

```
      Quick-Sort ( $A, q+1, r$ )
```

11 12

45 34

1 2 3 4 5 6 7 8 9 10 11 12

12 14 15 16 22 20 28 30 21 25 45

34

↑
 p

↑
 q

↑
 r

Algoritmo QuickSort

```
Quick-Sort ( $A, p, r$ )
```

```
IF  $p < r$ 
```

```
THEN  $q = \text{Partiziona}(A, p, r)$ 
```

```
Quick-Sort ( $A, p, q$ )
```

```
Quick-Sort ( $A, q+1, r$ )
```

11 12

45 34

1 2 3 4 5 6 7 8 9 10 11 12

12 14 15 16 \triangleleft 21 20 \triangleright 28 30 22 25 \triangleleft

45 34

\uparrow
 p q

\uparrow
 r

Algoritmo QuickSort

```
Quick-Sort ( $A, p, r$ )
```

```
IF  $p < r$ 
```

```
THEN  $q = \text{Partiziona}(A, p, r)$ 
```

```
Quick-Sort ( $A, p, q$ )
```

```
Quick-Sort ( $A, q+1, r$ )
```

11 12

45 34

1 2 3 4 5 6 7 8 9 10 11 12

12 14 15 16 \triangleleft 21 20 \triangleright 28 30 22 25 \triangleleft

45 34

\uparrow
 p q

\uparrow
 r

Algoritmo QuickSort

```
Quick-Sort (A, p, r)
```

```
  IF  $p < r$ 
```

```
    THEN  $q = \text{Partiziona}(A, p, r)$ 
```

```
      Quick-Sort (A, p, q)
```

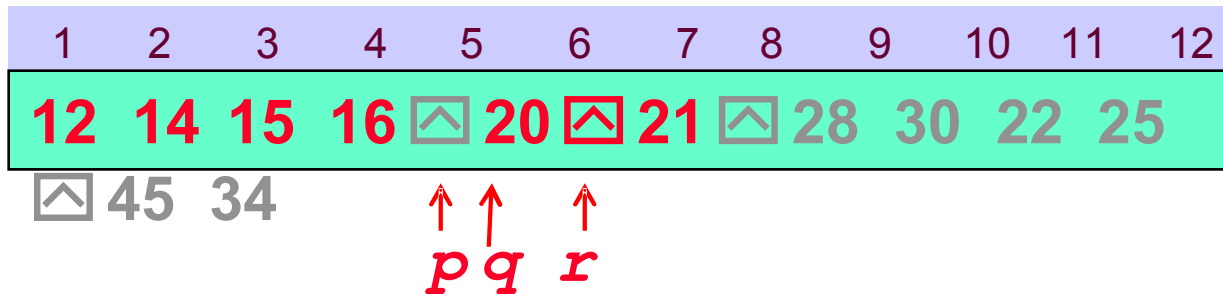
```
      Quick-Sort (A, q+1, r)
```

11 12

45 34

7 8 9 10

28 30 22 25



Algoritmo QuickSort

```
Quick-Sort ( $A, p, r$ )
```

```
  IF  $p < r$ 
```

```
    THEN  $q = \text{Partiziona}(A, p, r)$ 
```

```
        Quick-Sort ( $A, p, q$ )
```

```
        Quick-Sort ( $A, q+1, r$ )
```

11 12

45 34

1 2 3 4 5 6 7 8 9 10 11 12

12 14 15 16 \triangleleft 20 21 \triangleright 28 30 22 25 \triangleleft

45 34

\uparrow
 p q

\uparrow
 r

Algoritmo QuickSort

Quick-Sort (A, p, r)

IF $p < r$

THEN $q = \text{Partiziona}(A, p, r)$

Quick-Sort (A, p, q)

Quick-Sort ($A, q+1, r$)

11 12

45 34

1 2 3 4 5 6 7 8 9 10 11 12

12 14 15 16 \sphericalangle 20 21 \sphericalangle 28 30 22 25 \sphericalangle

45 34

\uparrow
 p

\uparrow
 r

Algoritmo QuickSort

```
Quick-Sort ( $A, p, r$ )
```

```
IF  $p < r$ 
```

```
THEN  $q = \text{Partiziona}(A, p, r)$ 
```

```
Quick-Sort ( $A, p, q$ )
```

```
Quick-Sort ( $A, q+1, r$ )
```

11 12

45 34

1 2 3 4 5 6 7 8 9 10 11 12

12 14 15 16 \triangleleft 20 21 \triangleleft 25 22 \triangleleft 30 28

\triangleleft 45 34

\uparrow \uparrow \uparrow
 p q r

Algoritmo QuickSort

```
Quick-Sort ( $A, p, r$ )
```

```
  IF  $p < r$ 
```

```
    THEN  $q = \text{Partiziona}(A, p, r)$ 
```

```
      Quick-Sort ( $A, p, q$ )
```

```
      Quick-Sort ( $A, q+1, r$ )
```

11 12

45 34

1 2 3 4 5 6 7 8 9 10 11 12

12 14 15 16 20 21 22 25 28 30

45 34

\uparrow \uparrow \uparrow
 p q r

Algoritmo QuickSort

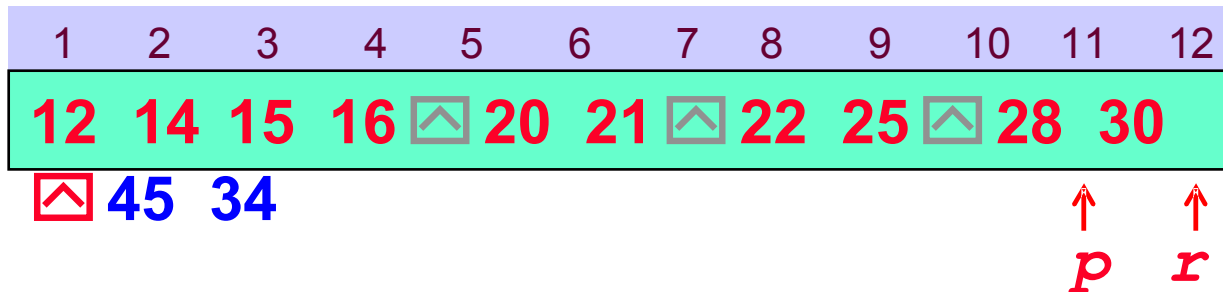
```
Quick-Sort ( $A, p, r$ )
```

```
IF  $p < r$ 
```

```
THEN  $q = \text{Partiziona}(A, p, r)$ 
```

```
Quick-Sort ( $A, p, q$ )
```

```
Quick-Sort ( $A, q+1, r$ )
```



Algoritmo QuickSort

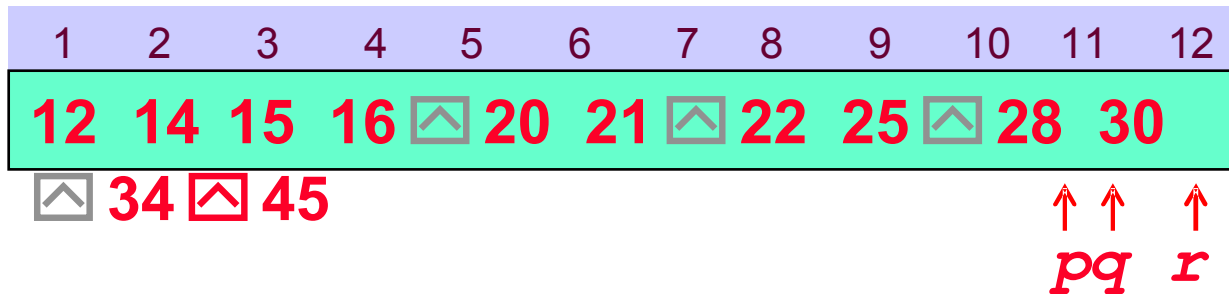
```
Quick-Sort ( $A, p, r$ )
```

```
IF  $p < r$ 
```

```
THEN  $q = \text{Partiziona}(A, p, r)$ 
```

```
Quick-Sort ( $A, p, q$ )
```

```
Quick-Sort ( $A, q+1, r$ )
```



Algoritmo QuickSort

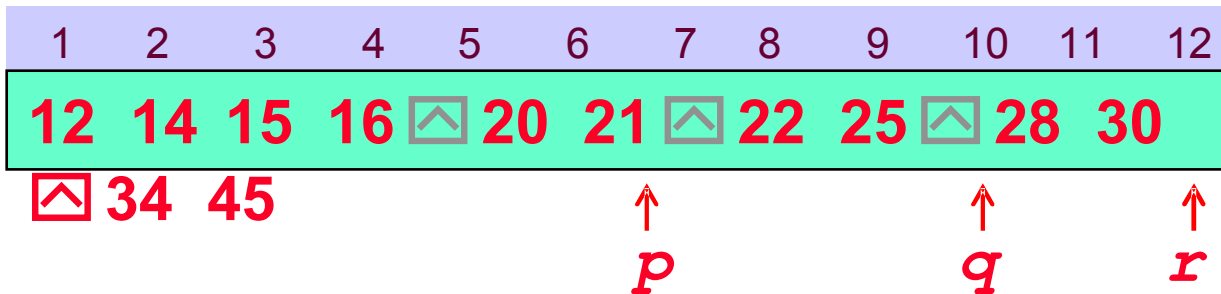
```
Quick-Sort ( $A, p, r$ )
```

```
  IF  $p < r$ 
```

```
    THEN  $q = \text{Partiziona}(A, p, r)$ 
```

```
        Quick-Sort ( $A, p, q$ )
```


```
        Quick-Sort ( $A, q+1, r$ )
```



Algoritmo QuickSort

```
Quick-Sort ( $A, p, r$ )  
  IF  $p < r$   
    THEN  $q = \text{Partiziona}(A, p, r)$   
         Quick-Sort ( $A, p, q$ )  
         Quick-Sort ( $A, q+1, r$ )
```

L'array A ora è ordinato!



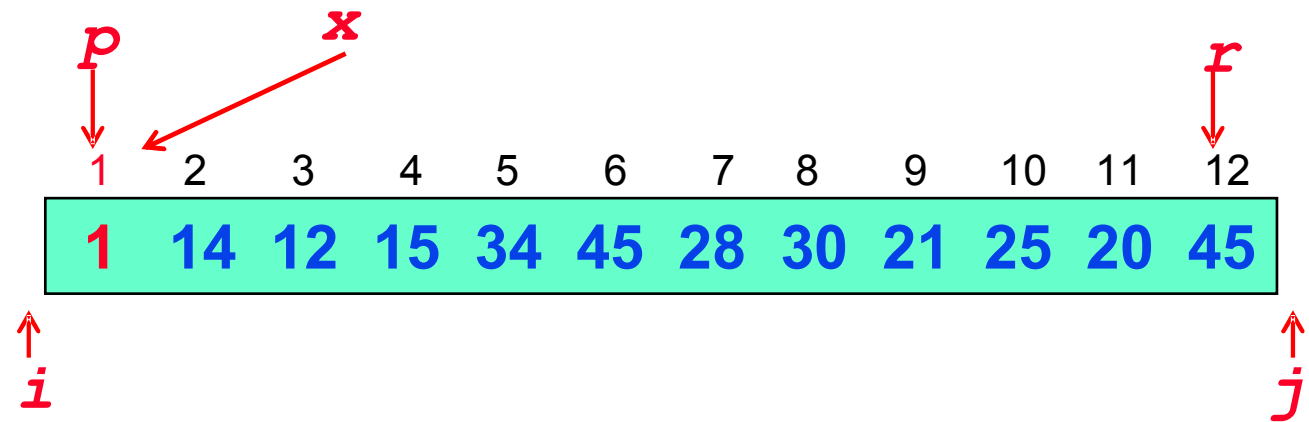
1	2	3	4	5	6	7	8	9	10	11	12
12	14	15	16	20	21	22	25	28	30	34	45

Algoritmo Partiziona: analisi

Gli indici i e j che scandiscono la sequenza non ne eccedono mai i limiti. Cioè vale sempre che $i \leq r$ e $j \geq p$

Algoritmo Partiziona: analisi

Gli indici i e j che scandiscono la sequenza non ne eccedono mai i limiti. Cioè vale sempre che $i \leq r$ e $j \geq p$

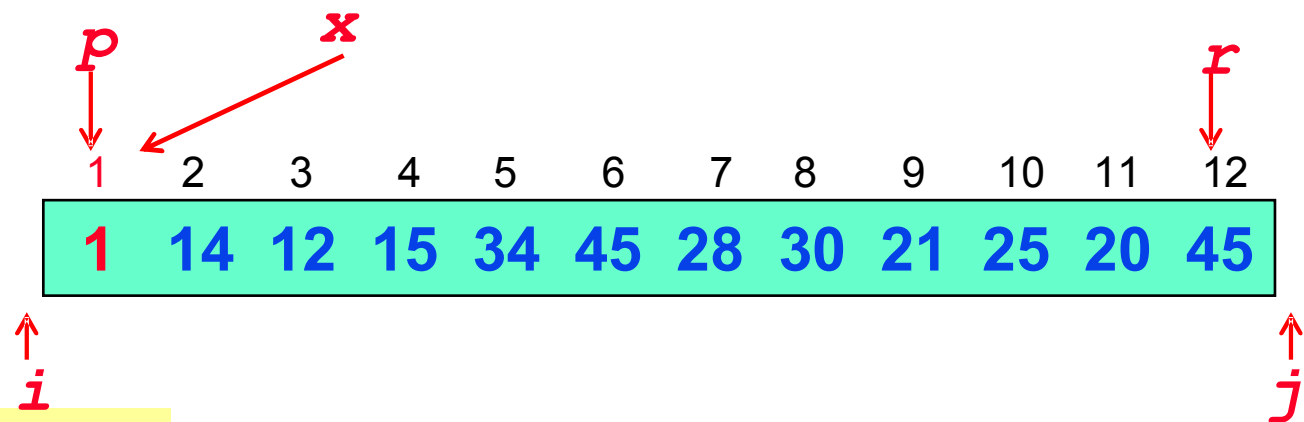


2 Casi. Partiziona effettua:

- *nessuno spostamento*
- *almeno uno spostamento*

Algoritmo Partiziona: analisi

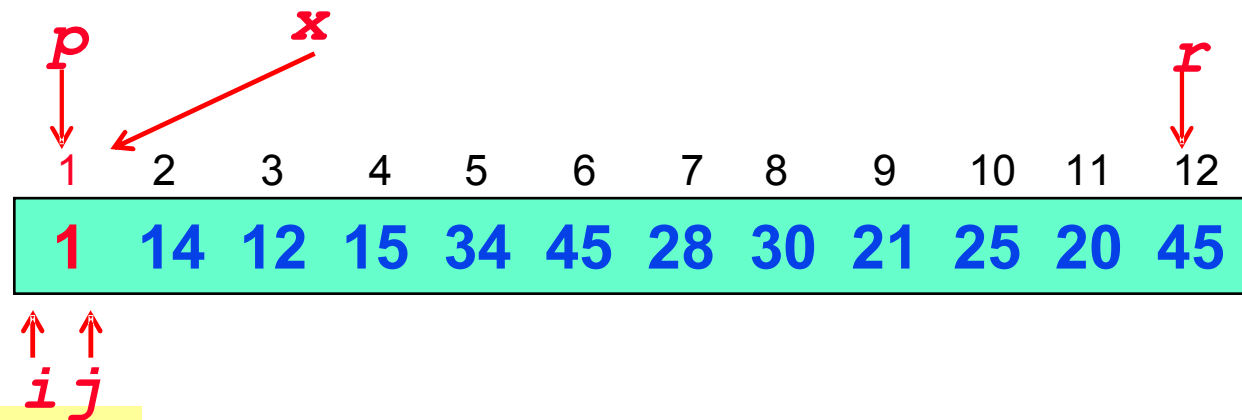
Gli indici i e j che scandiscono la sequenza non ne eccedono mai i limiti. Cioè vale sempre che $i \leq r$ e $j \geq p$



```
...  
REPEAT  $j = j - 1$   
  UNTIL  $A[j] \leq x$   
REPEAT  $i = i + 1$   
  UNTIL  $A[i] \geq x$   
...
```

Algoritmo Partiziona: analisi

Gli indici i e j che scandiscono la sequenza non ne eccedono mai i limiti. Cioè vale sempre che $i \leq r$ e $j \geq p$



```
...  
REPEAT  $j = j - 1$   
  UNTIL  $A[j] \leq x$   
REPEAT  $i = i + 1$   
  UNTIL  $A[i] \geq x$   
...
```

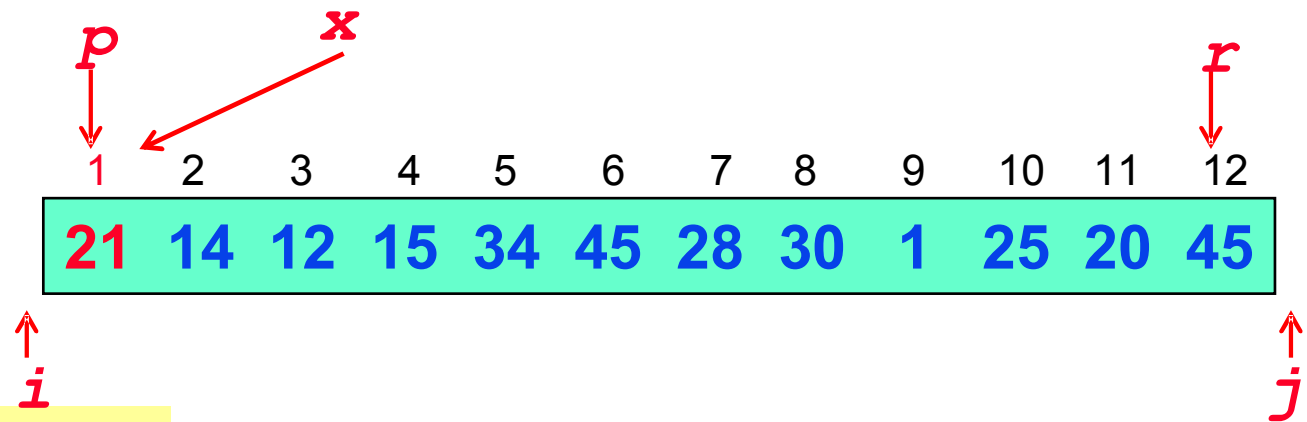
nessuno spostamento

$A[j] \leq x$ per $j \geq p$

$A[i] \geq x$ per $i \leq p$

Algoritmo Partiziona: analisi

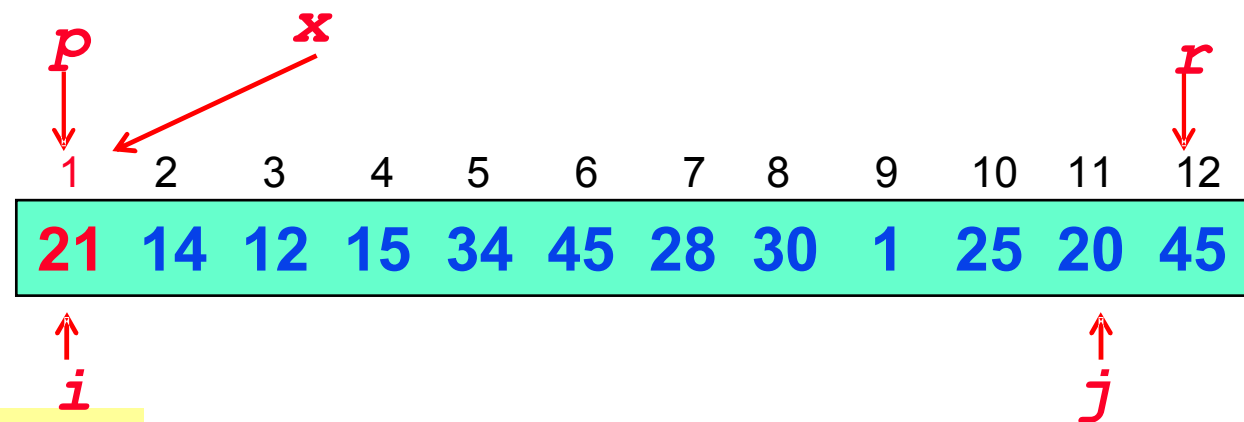
Gli indici i e j che scandiscono la sequenza non ne eccedono mai i limiti. Cioè vale sempre che $i \leq r$ e $j \geq p$



```
...  
REPEAT  $j = j - 1$   
  UNTIL  $A[j] \leq x$   
REPEAT  $i = i + 1$   
  UNTIL  $A[i] \geq x$   
...
```

Algoritmo Partiziona: analisi

Gli indici i e j che scandiscono la sequenza non ne eccedono mai i limiti. Cioè vale sempre che $i \leq r$ e $j \geq p$



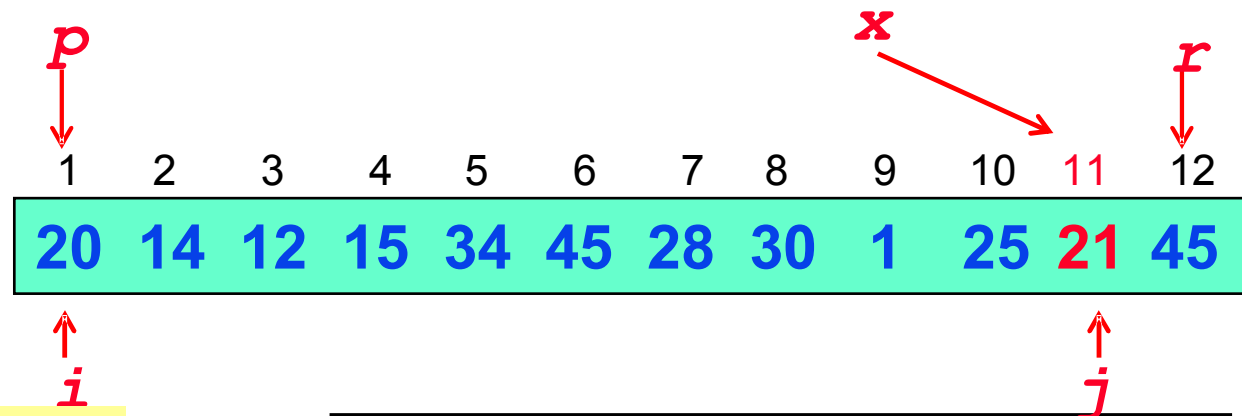
...

```
REPEAT  $j = j - 1$   
  UNTIL  $A[j] \leq x$   
REPEAT  $i = i + 1$   
  UNTIL  $A[i] \geq x$ 
```

...

Algoritmo Partiziona: analisi

Gli indici i e j che scandiscono la sequenza non ne eccedono mai i limiti. Cioè vale sempre che $i \leq r$ e $j \geq p$

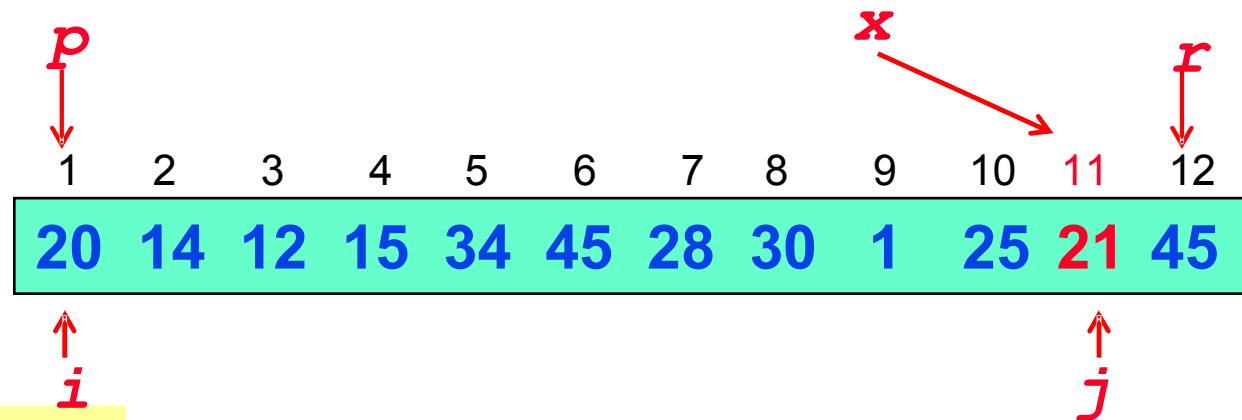


```
...  
IF  $i < j$   
  THEN "scambia  
         $A[i]$  con  $A[j]$ "  
  ELSE fine = true  
...
```

dopo il primo spostamento,
esiste un k tale che
 $A[k] \leq x$ con $p \leq k \leq j$
esiste un z tale che
 $A[z] \geq x$ con $i \leq z \leq r$

Algoritmo Partiziona: analisi

Gli indici i e j che scandiscono la sequenza non ne eccedono mai i limiti. Cioè vale sempre che $i \leq r$ e $j \geq p$



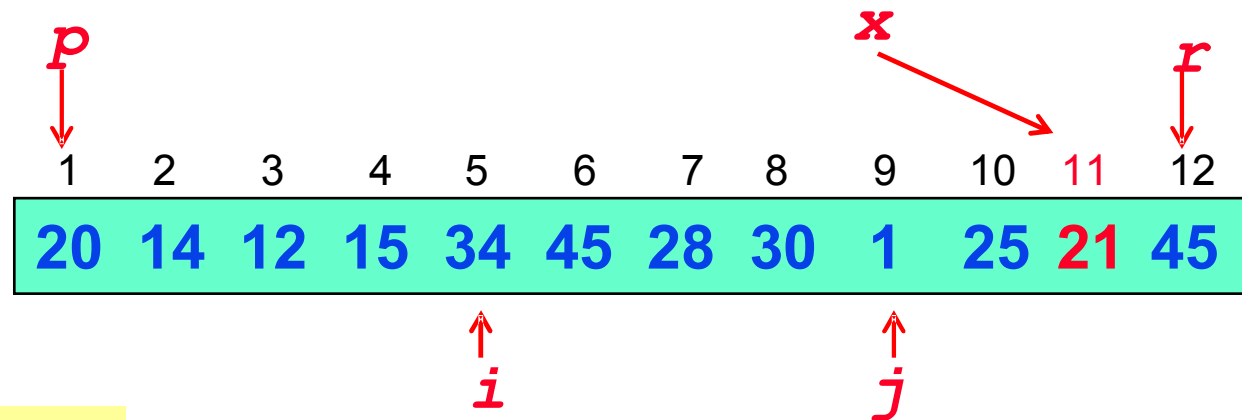
```
...  
REPEAT  $j = j - 1$   
  UNTIL  $A[j] \leq x$   
REPEAT  $i = i + 1$   
  UNTIL  $A[i] \geq x$   
...
```

In generale, dopo ogni scambio:

- un elemento minore o uguale ad x viene spostato tra p e $j-1$
- un elemento maggiore o uguale ad x viene spostato tra $i+1$ e r

Algoritmo Partiziona: analisi

Gli indici i e j che scandiscono la sequenza non ne eccedono mai i limiti. Cioè vale sempre che $i \leq r$ e $j \geq p$



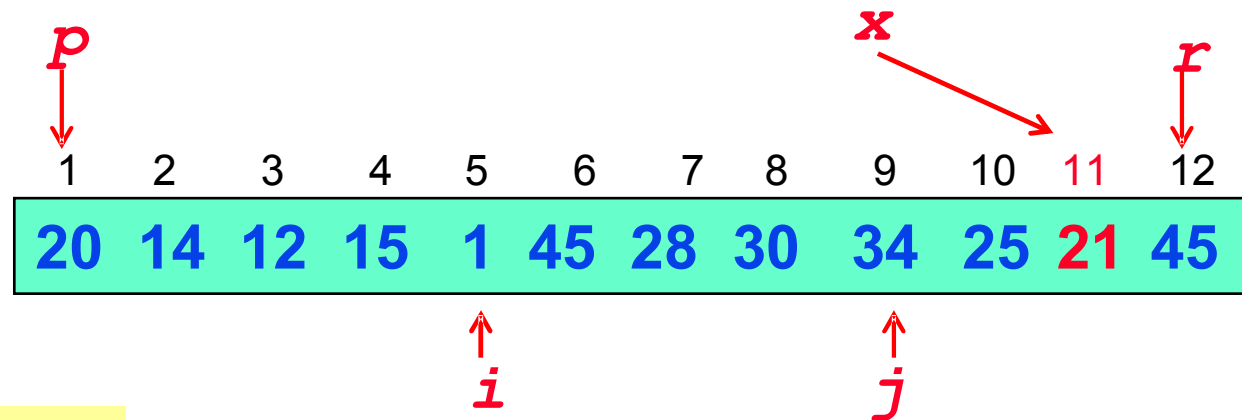
```
...  
REPEAT  $j = j - 1$   
  UNTIL  $A[j] \leq x$   
REPEAT  $i = i + 1$   
  UNTIL  $A[i] \geq x$   
...
```

In generale, dopo ogni scambio:

- tra p e $j-1$ ci sarà sicuramente un elemento minore o uguale ad x
- tra $i+1$ e r ci sarà sicuramente un elemento maggiore o uguale ad x

Algoritmo Partiziona: analisi

Gli indici i e j che scandiscono la sequenza non ne eccedono mai i limiti. Cioè vale sempre che $i \leq r$ e $j \geq p$



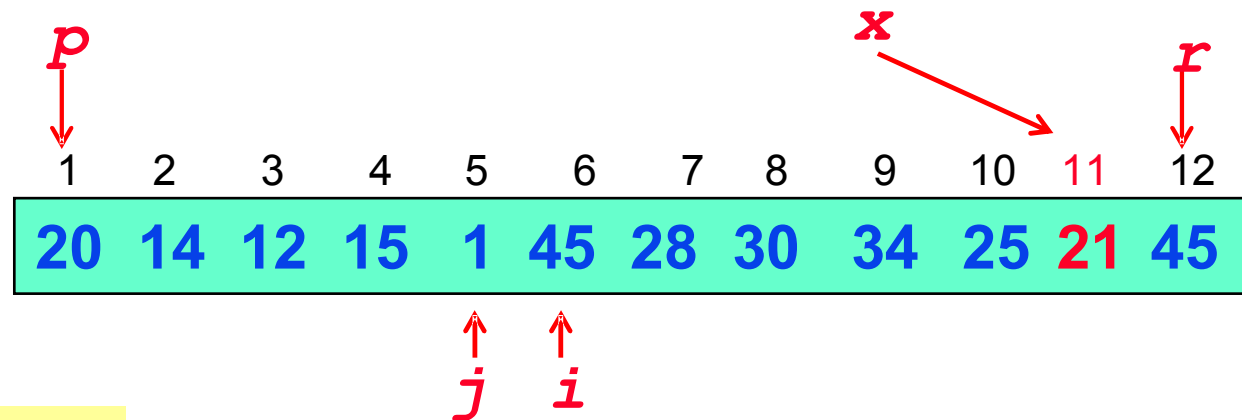
```
...  
IF  $i < j$   
  THEN "scambia  
         $A[i]$  con  $A[j]$ "  
  ELSE fine = true  
...
```

In generale, dopo ogni scambio:

- tra p e $j-1$ ci sarà sicuramente un elemento minore o uguale ad x
- tra $i+1$ e r ci sarà sicuramente un elemento maggiore o uguale ad x

Algoritmo Partiziona: analisi

Gli indici i e j che scandiscono la sequenza non ne eccedono mai i limiti. Cioè vale sempre che $i \leq r$ e $j \geq p$



```
...  
REPEAT  $j = j - 1$   
  UNTIL  $A[j] \leq x$   
REPEAT  $i = i + 1$   
  UNTIL  $A[i] \geq x$   
...
```

In generale, dopo ogni scambio:

- tra p e $j-1$ ci sarà sicuramente un elemento minore o uguale ad x
- tra $i+1$ e r ci sarà sicuramente un elemento maggiore o uguale ad x

Algoritmo Partiziona: analisi

```
int Partiziona(A, p, r)
```

```
  x = A[p]
```

```
  i = p - 1
```

```
  j = r + 1
```

```
  fine = false
```

} = $\Theta(1)$

```
  REPEAT
```

```
    REPEAT j = j - 1
```

```
      UNTIL A[j] ≤ x
```

```
    REPEAT i = i + 1
```

```
      UNTIL A[i] ≥ x
```

```
    IF i < j
```

```
      THEN "scambia A[i] con A[j]"
```

```
      ELSE fine = true
```

} = $\Theta(1)$

```
  UNTIL fine DO
```

```
  return j
```

Algoritmo Partiziona: analisi

```
int Partiziona (A, p, r)
```

```
  x = A[p]
```

```
  i = p - 1
```

```
  j = r + 1
```

```
  fine = false
```

```
  REPEAT
```

```
    REPEAT j = j -
```

```
      UNTIL A[j]
```

```
    REPEAT i = i +
```

```
      UNTIL A[i]
```

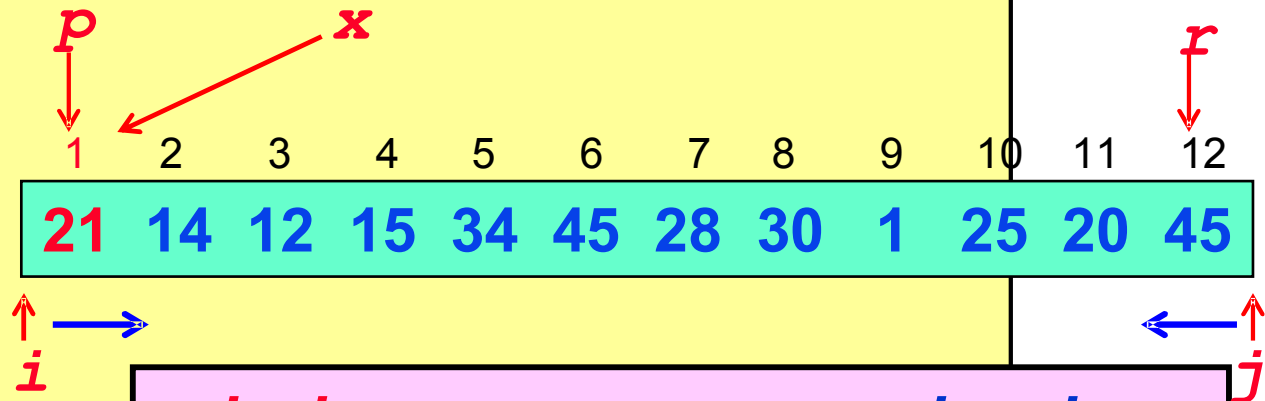
```
    IF i < j
```

```
      THEN "scam"
```

```
      ELSE fine = true
```

```
  UNTIL fine DO
```

```
  return j
```



- ☆ i e j non possono eccedere i limiti dell'array,
- 🕒 i e j sono sempre rispettivamente crescente e decrescente
- 🕒 l'algoritmo termina quando $i \geq j$ quindi il costo del REPEAT sarà proporzionale ad n , cioè $\Theta(n)$

Algoritmo Partiziona: analisi

```
int Partiziona(A, p, r)
```

```
  x = A[p]
```

```
  i = p - 1
```

```
  j = r + 1
```

```
  fine = false
```

} = $\Theta(1)$

```
  REPEAT
```

```
    REPEAT j = j - 1
```

```
      UNTIL A[j] ≤ x
```

```
    REPEAT i = i + 1
```

```
      UNTIL A[i] ≥ x
```

```
    IF i < j
```

```
      THEN "scambia A[i] con A[j]"
```

```
      ELSE fine = true
```

```
  UNTIL fine DO
```

```
  return j
```

} = $\Theta(n)$

Algoritmo Partiziona: analisi

```
int Partiziona(A, p, r)
  x = A[p]
  i = p - 1
  j = r + 1
  fine = false
  REPEAT
    REPEAT j = j - 1
      UNTIL A[j] ≤ x
    REPEAT i = i + 1
      UNTIL A[i] ≥ x
    IF i < j
      THEN "scambia A[i] con A[j]"
      ELSE fine = true
  UNTIL fine DO
return j
```

$$T(n) = \Theta(n)$$

Analisi di QuickSort: intuizioni

Il *tempo di esecuzione* di QuickSort dipende dalla *bilanciamento* delle partizioni effettuate dall'algoritmo *partiziona*:

$$T(1) = \Theta(1)$$

$$T(n) = T(q) + T(n - q) + \Theta(n) \quad \text{se } n > 1$$

- Il *caso migliore* si verifica quando le partizioni sono *perfettamente bilanciate*, entrambe di dimensione $n/2$
- Il *caso peggiore* si verifica quando una partizione è sempre di dimensione **1** (la seconda è quindi di dimensione $n - 1$)

Analisi di QuickSort: caso migliore

```
Quick-Sort (A, p, r)
  IF p < r
    THEN q = Partiziona (A, p, r)
         Quick-Sort (A, p, q)
         Quick-Sort (A, q+1, r)
```

Le partizioni sono di uguale dimensione:

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

e per il *caso 2* del *metodo principale*:

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

Analisi di QuickSort: caso migliore

```
Quick-Sort (A, p, r)
  IF p < r
    THEN q = Partiziona (A, p, r)
         Quick-Sort (A, p, q)
         Quick-Sort (A, q+1, r)
```

Quando si verifica il caso migliore, ad esempio?

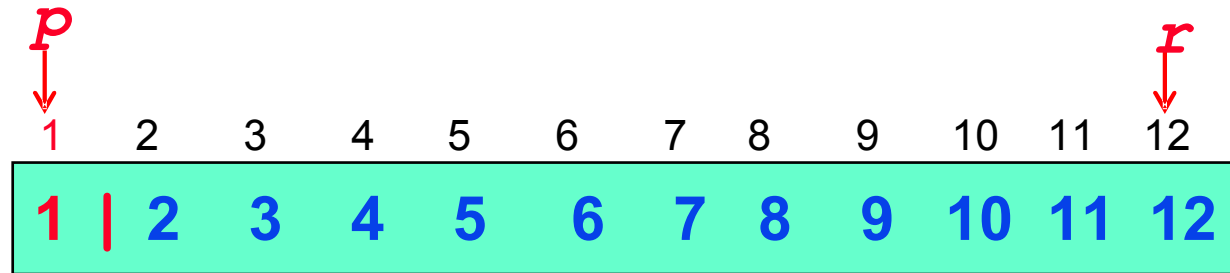
Le partizioni sono di uguale dimensione:

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

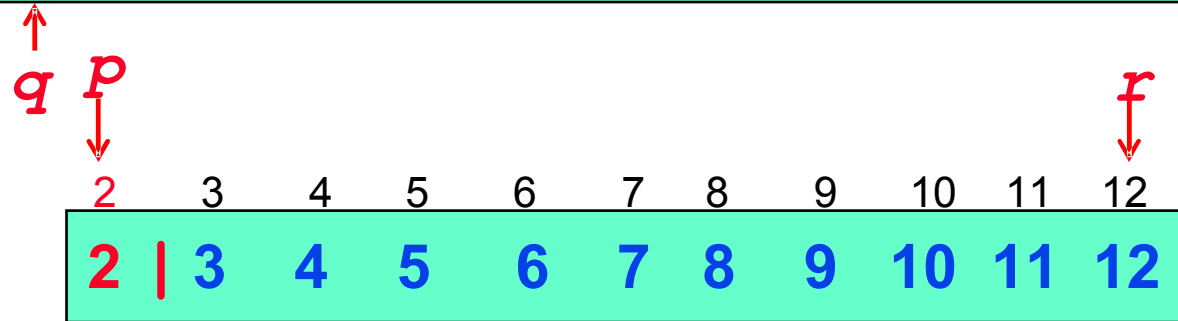
e per il **caso 2** del **metodo principale**:

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

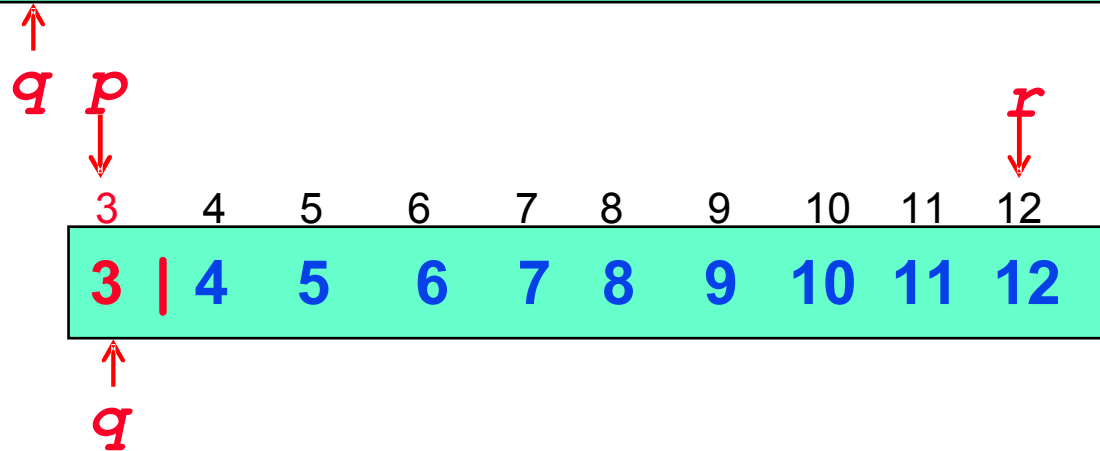
Analisi di QuickSort: caso peggiore



Pivot = 1



Pivot = 2



Pivot = 3

Analisi di QuickSort: caso peggiore

```
Quick-Sort (A, p, r)
  IF p < r
    THEN q = Partiziona (A, p, r)
         Quick-Sort (A, p, q)
         Quick-Sort (A, q+1, r)
```

La partizione sinistra ha dimensione **1** mentre quella destra ha dimensione **$n-1$** :

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

poiché **$T(1) = 1$** otteniamo

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

Analisi di QuickSort: caso peggiore

L'equazione di ricorrenza può essere risolta facilmente col *metodo iterativo*

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

$$= \sum_{k=1}^n \Theta(k) =$$

$$= \Theta\left(\sum_{k=1}^n k\right) =$$

$$= \Theta(n^2)$$

Analisi di QuickSort: caso peggiore

```
Quick-Sort (A, p, r)
  IF  $p < r$ 
    THEN  $q = \text{Partiziona}(A, p, r)$ 
         Quick-Sort (A, p, q)
         Quick-Sort (A, q+1, r)
```

*Quando si verifica
il caso peggiore, ad
esempio?*

La partizione sinistra ha dimensione **1** mentre
quella destra ha dimensione $n-1$:

$$\begin{aligned} T(n) &= T(n-1) + \Theta(n) = \\ &= \Theta(n^2) \end{aligned}$$