Programmazione e Algoritmica

QuickSort

QuickSort

Divide

Impera

Combina

QuickSort

Divide

Impera

Combina

Scegli un elemento pivot, e dividi l'insieme da ordinare in elementi ≤ e >

QuickSort

Divide

Scegli un elemento pivot, e dividi l'insieme da ordinare in elementi ≤ e >

Impera

Ordina le due parti

Combina

QuickSort

Divide

Scegli un elemento pivot, e dividi l'insieme da ordinare in elementi ≤ e >

Impera

Ordina le due parti

Combina

Concatena le due parti

QuickSort

Alg.: QUICK-SORT(*A*, p, r)

```
if p < r
  then q = PARTIZIONA(A, p, r)        ▷ Divide
       QUICK-SORT(A, p, q - 1)        ▷ Impera
       QUICK-SORT(A, q + 1, r)        ▷ Impera
                                      ▷ Combina
```

Chiamata Iniziale: QUICK-SORT(A, 0, n-1)

Esempio

100	20	10	80	60	50	7	30	40

Scegliamo elemento pivot

Si può scegliere in diversi modi: ad esempio, prendiamo **l'ultimo**

Partiziona

Scelto un **pivot** (o **perno**), questa procedura partiziona l'array in modo che:

1. Una porzione contiene **elementi > pivot**
2. L'altra **elementi ≤ pivot**

Il tutto **in-place**!

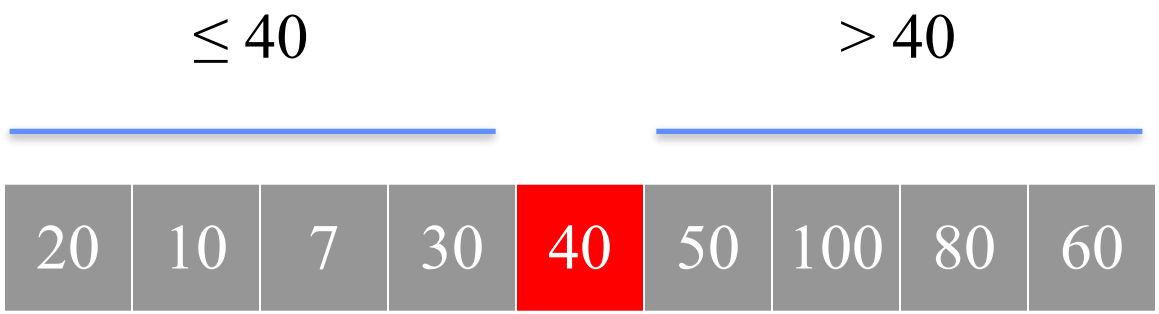
100	20	10	80	60	50	7	30	40

Partiziona

Scelto un **pivot** (o **perno**), questa procedura partiziona l'array in modo che:

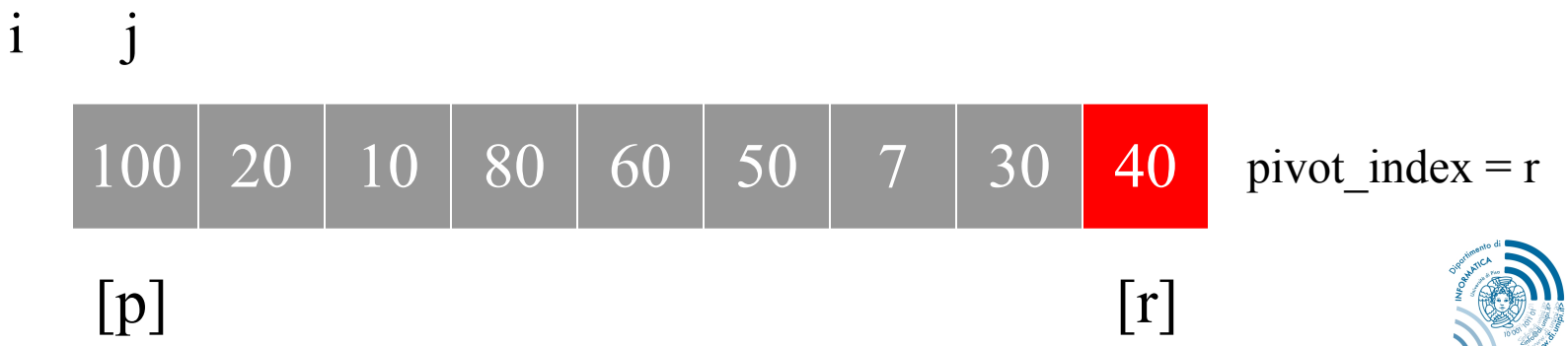
1. Una porzione contiene **elementi > pivot**
2. L'altra **elementi ≤ pivot**

Il tutto **in-place**!



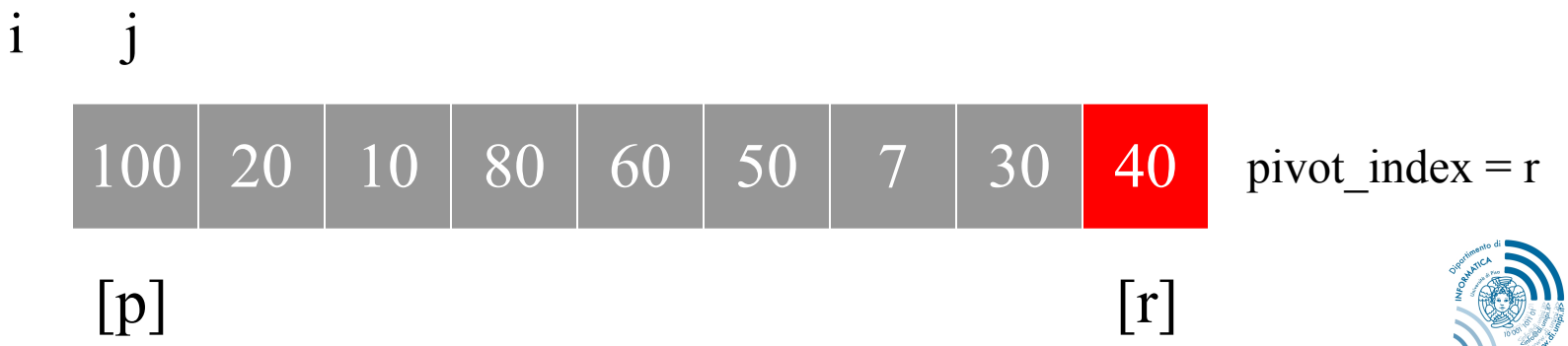
Partiziona(A, p, r)

Provate a scrivere Partiziona(A, p, r)



Partiziona(A, p, r)

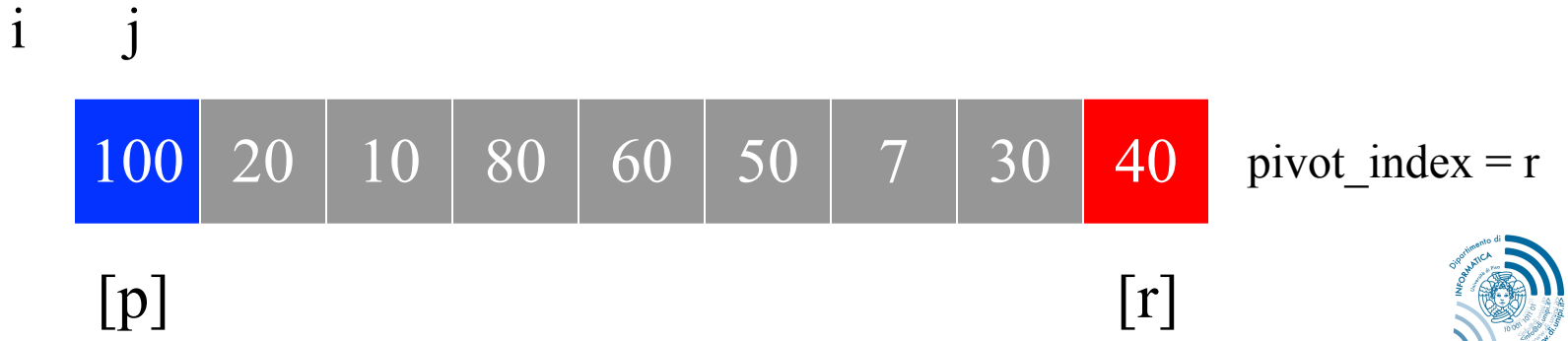
- 1.Inizializza *i = (p-1)* e *j = p*



Partiziona(A, p, r)

1. Inizializza $i = (p-1)$ e $j = p$
2. **If** A[j] ≤ pivot

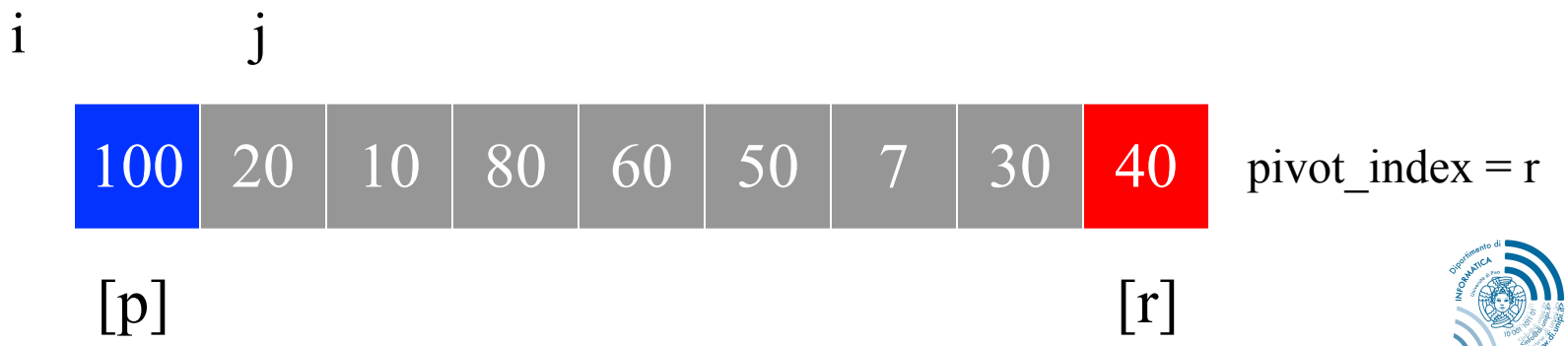
100 <? 40



Partiziona(A, p, r)

1. Inizializza *i = (p-1)* e *j = p*
2. **If** A[j] ≤ pivot ….
3. **Else** j++ e vai a 2.

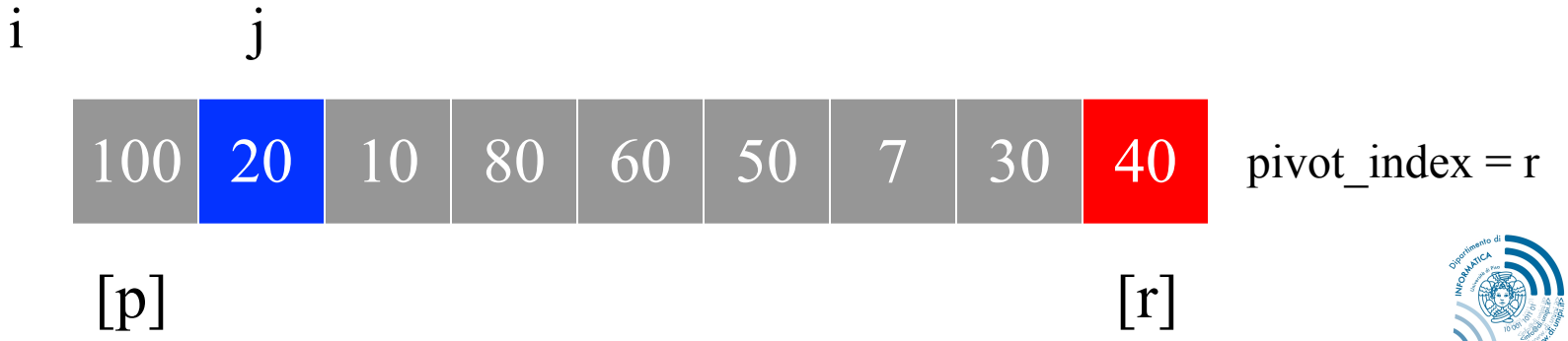
100 <? 40



Partiziona(A, p, r)

1. Inizializza $i = (p-1)$ e $j = p$
2. **If** A[j] ≤ pivot

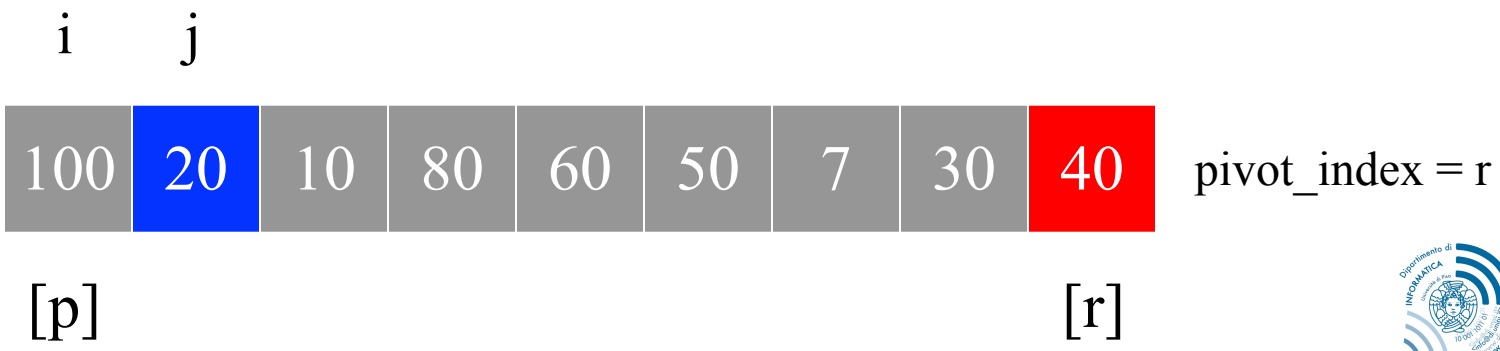
20 <? 40



Partiziona(A, p, r)

1. Inizializza $i = (p-1)$ e $j = p$
2. **If** A[j] ≤ pivot
   1. i++

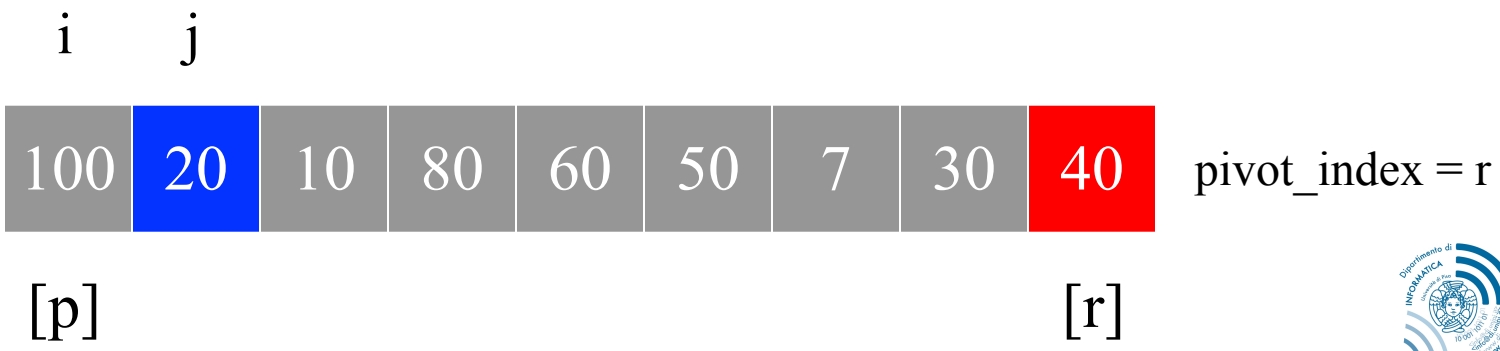
20 <? 40



Partiziona(A, p, r)

1. Inizializza *i = (p-1)* e *j = p*
2. **If** A[j] ≤ pivot
    1. i++
    2. Scambia A[i] e A[j]

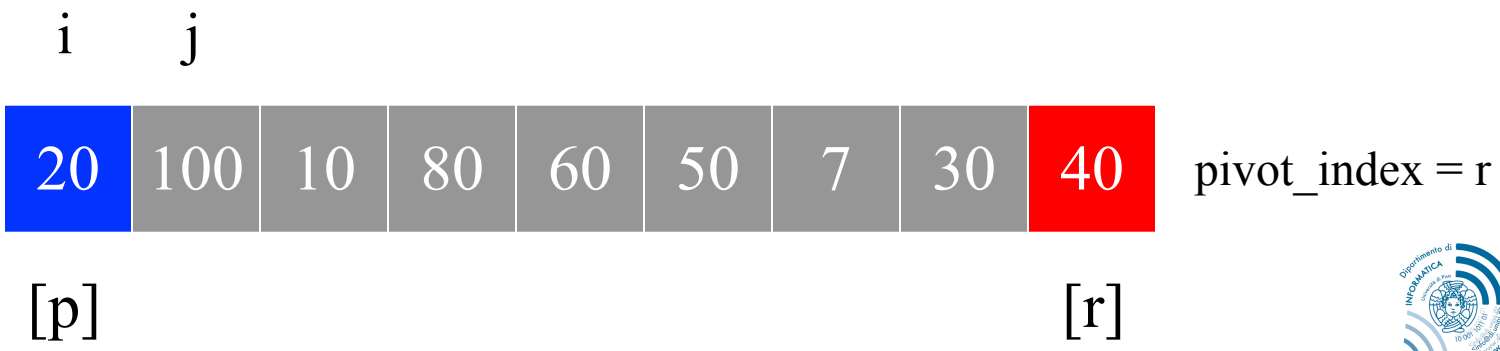
20 <? 40



Partiziona(A, p, r)

1. Inizializza *i = (p-1)* e *j = p*
2. **If** A[j] ≤ pivot
    1. i++
    2. Scambia A[i] e A[j]

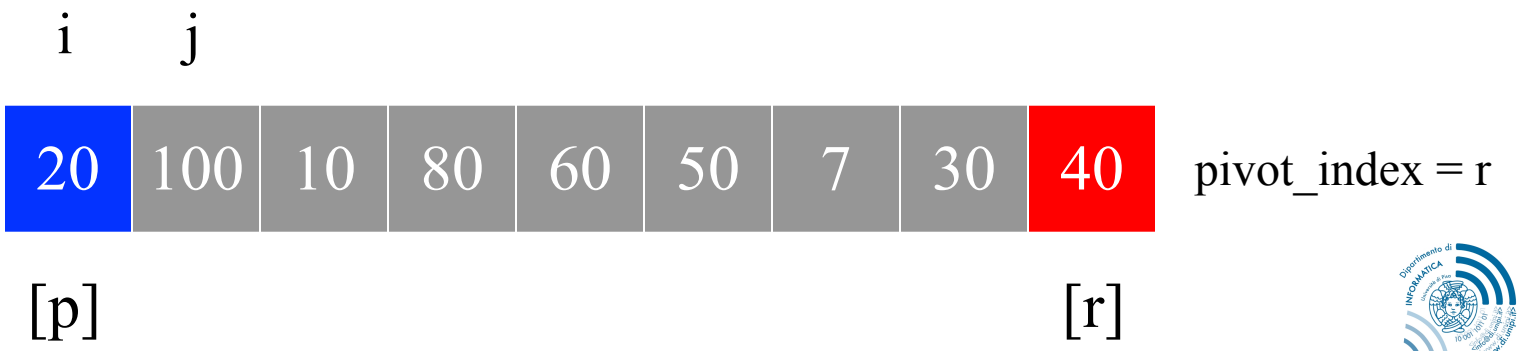
20 <? 40



Partiziona(A, p, r)

1. Inizializza *i = (p-1)* e *j = p*
2. **If** A[j] ≤ pivot
    1. i++
    2. Scambia A[i] e A[j]
    3. j++ e vai a 2.

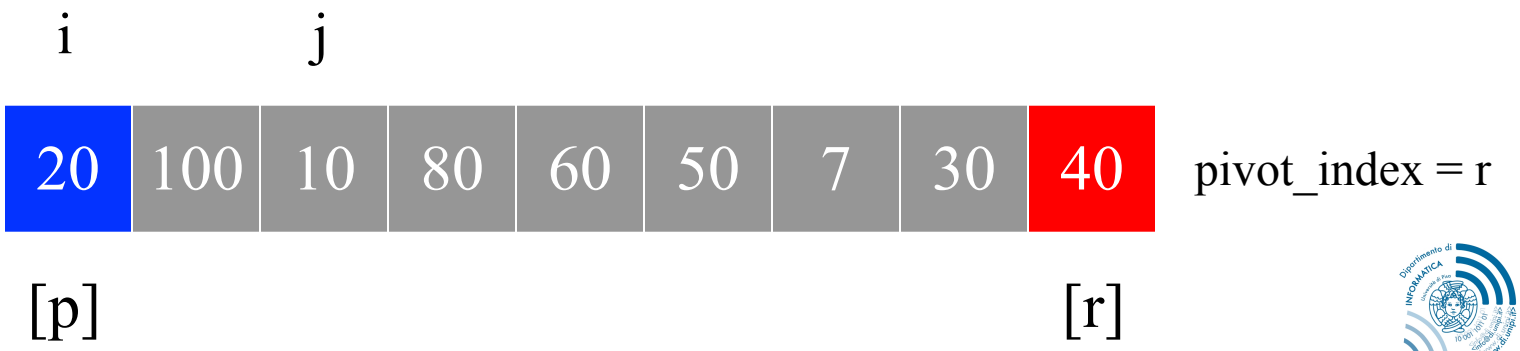
20 <? 40



Partiziona(A, p, r)

1. Inizializza *i = (p-1)* e *j = p*
2. **If** A[j] ≤ pivot
    1. i++
    2. Scambia A[i] e A[j]
    3. j++ e vai a 2.

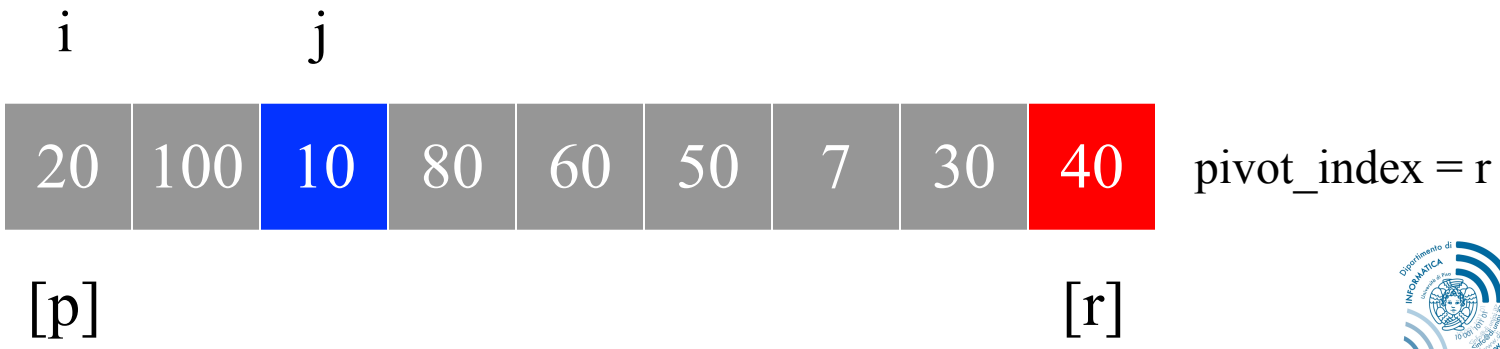
20 <? 40



Partiziona(A, p, r)

1. Inizializza *i = (p-1)* e *j = p*
2. **If** A[j] ≤ pivot
   1. i++
   2. Scambia A[i] e A[j]
   3. j++ e vai a 2.

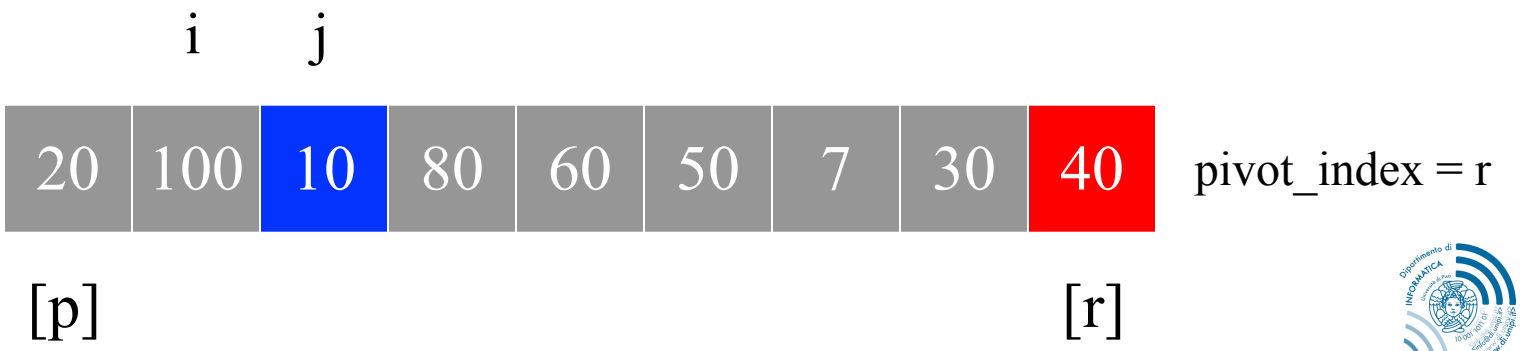
10 <? 40



Partiziona(A, p, r)

1. Inizializza *i = (p-1)* e *j = p*
2. **If** A[j] ≤ pivot
   1. i++
   2. Scambia A[i] e A[j]
   3. j++ e vai a 2.

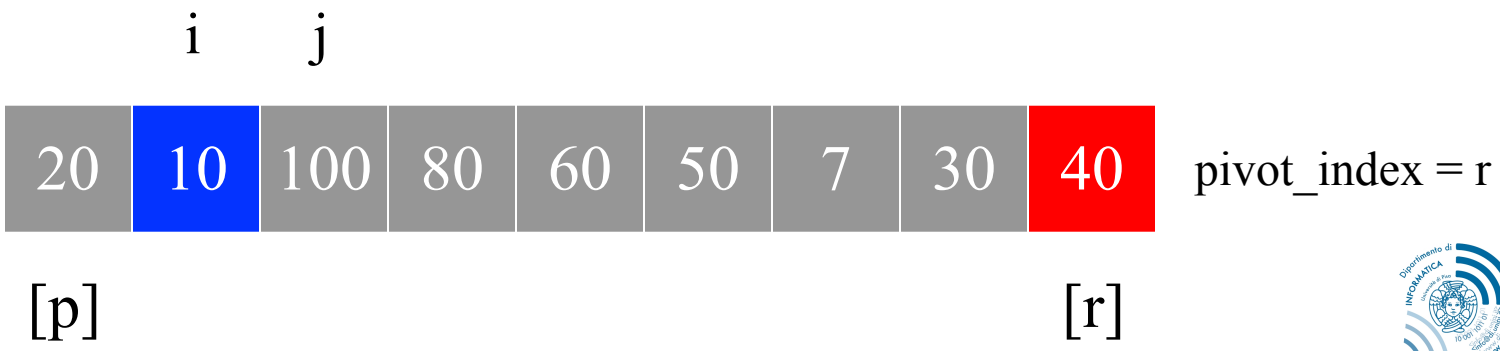
10 <? 40



Partiziona(A, p, r)

1. Inizializza *i = (p-1)* e *j = p*
2. **If** A[j] ≤ pivot
    1. i++
    2. Scambia A[i] e A[j]
    3. j++ e vai a 2.

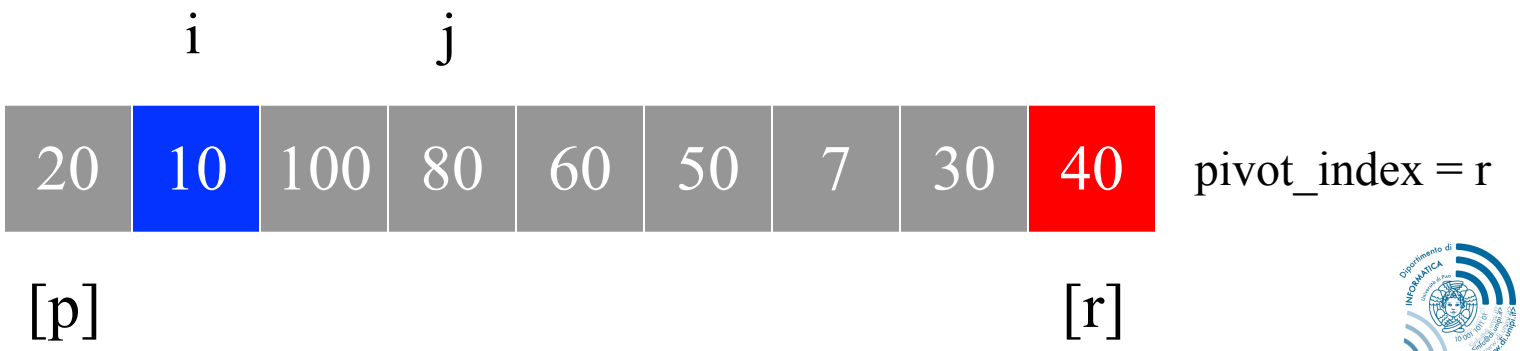
10 <? 40



Partiziona(A, p, r)

1. Inizializza *i = (p-1)* e *j = p*
2. **If** A[j] ≤ pivot
    1. i++
    2. Scambia A[i] e A[j]
    3. j++ e vai a 2.

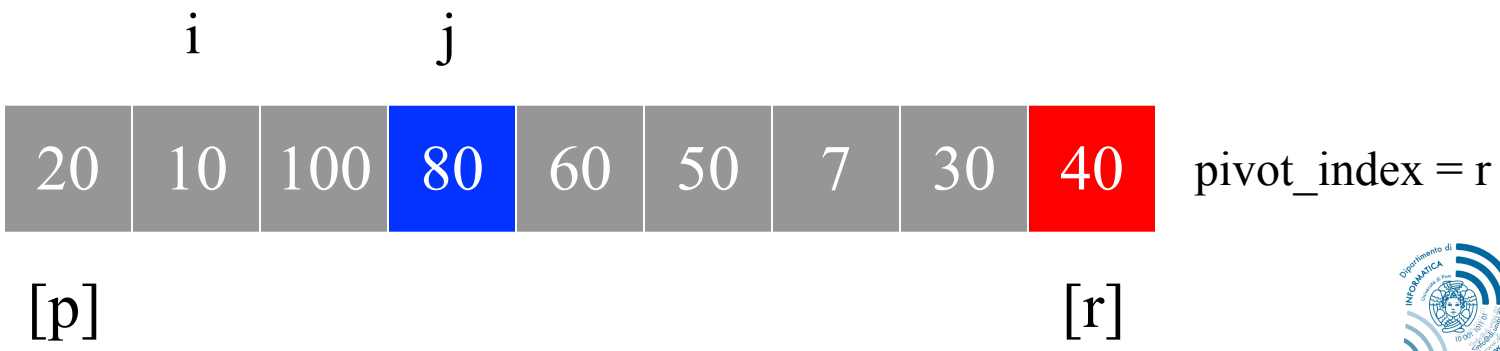
10 <? 40



Partiziona(A, p, r)

1. Inizializza $i = (p-1)$ e $j = p$
2. **If** A[j] ≤ pivot
   1. i++
   2. Scambia A[i] e A[j]
   3. j++ e vai a 2.

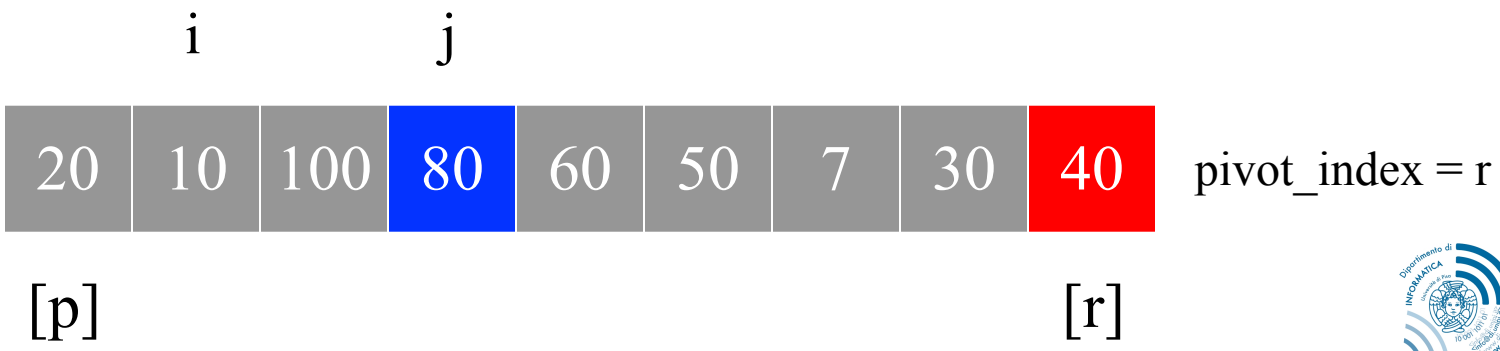
80 <? 40



Partiziona(A, p, r)

1. Inizializza *i = (p-1)* e *j = p*
2. **If** A[j] ≤ pivot
    1. i++
    2. Scambia A[i] e A[j]
    3. j++ e vai a 2.
3. **Else** j++ e vai a 2.

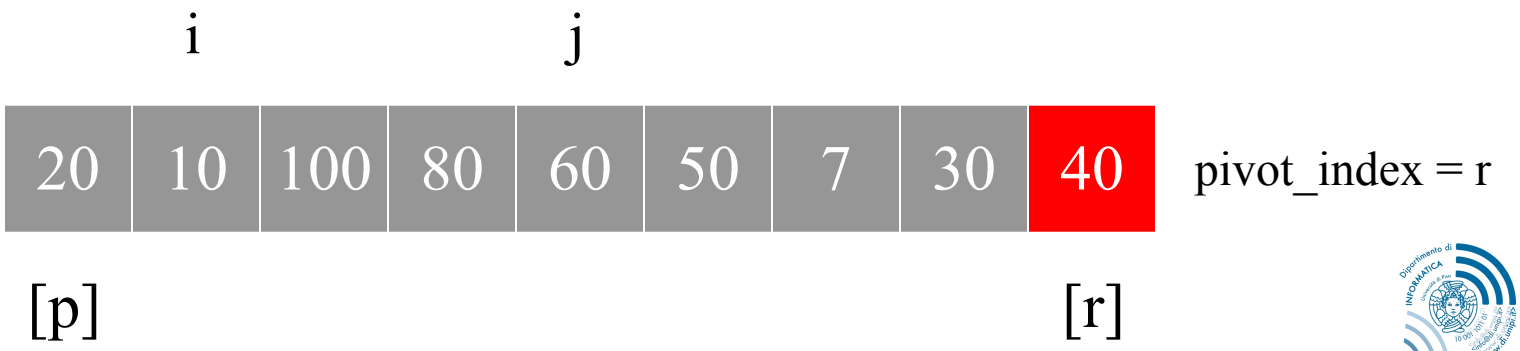
80 <? 40



Partiziona(A, p, r)

1. Inizializza *i = (p-1)* e *j = p*
2. **If** A[j] ≤ pivot
    1. i++
    2. Scambia A[i] e A[j]
    3. j++ e vai a 2.
3. **Else** j++ e vai a 2.

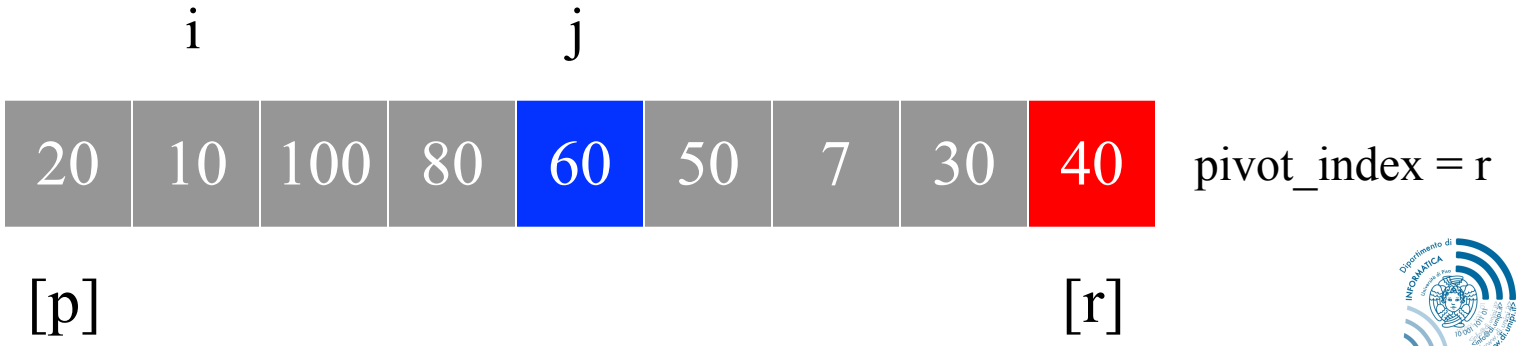
80 <? 40



Partiziona(A, p, r)

1. Inizializza *i = (p-1)* e *j = p*
2. **If** A[j] ≤ pivot
   1. i++
   2. Scambia A[i] e A[j]
   3. j++ e vai a 2.
3. **Else** j++ e vai a 2.

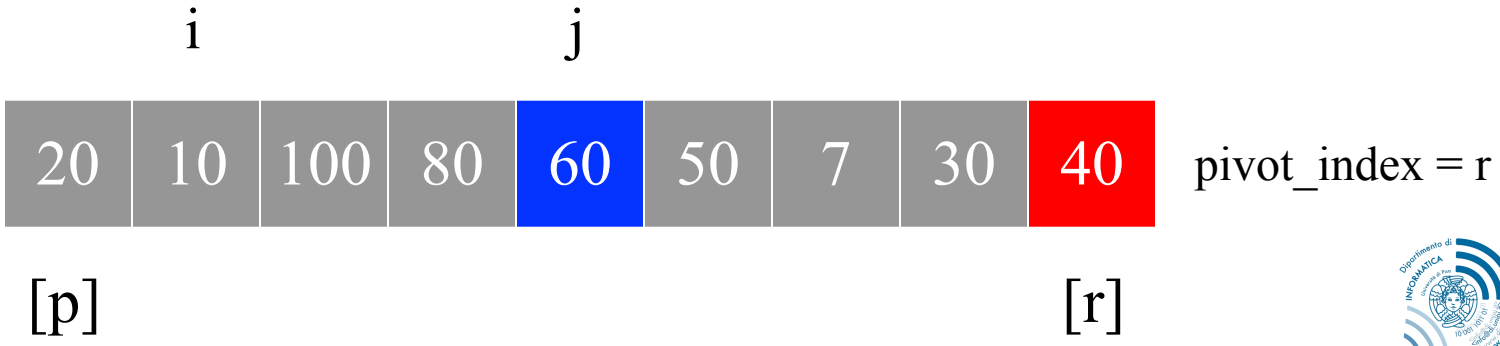
60 <? 40



Partiziona(A, p, r)

1. Inizializza *i = (p-1)* e *j = p*
2. **If** A[j] ≤ pivot
    1. i++
    2. Scambia A[i] e A[j]
    3. j++ e vai a 2.
3. **Else** j++ e vai a 2.

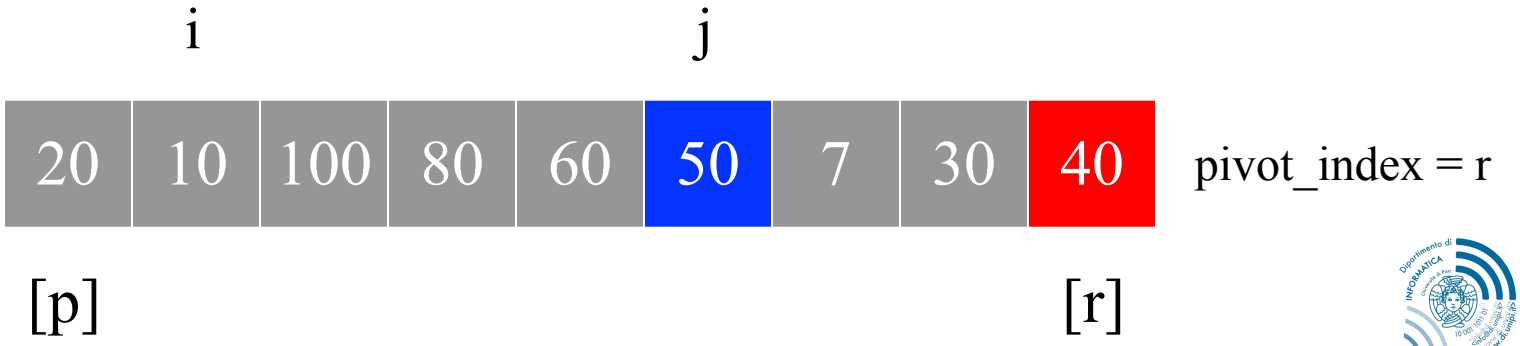
60 <? 40



Partiziona(A, p, r)

1. Inizializza *i = (p-1)* e *j = p*
2. **If** A[j] ≤ pivot
   1. i++
   2. Scambia A[i] e A[j]
   3. j++ e vai a 2.
3. **Else** j++ e vai a 2.

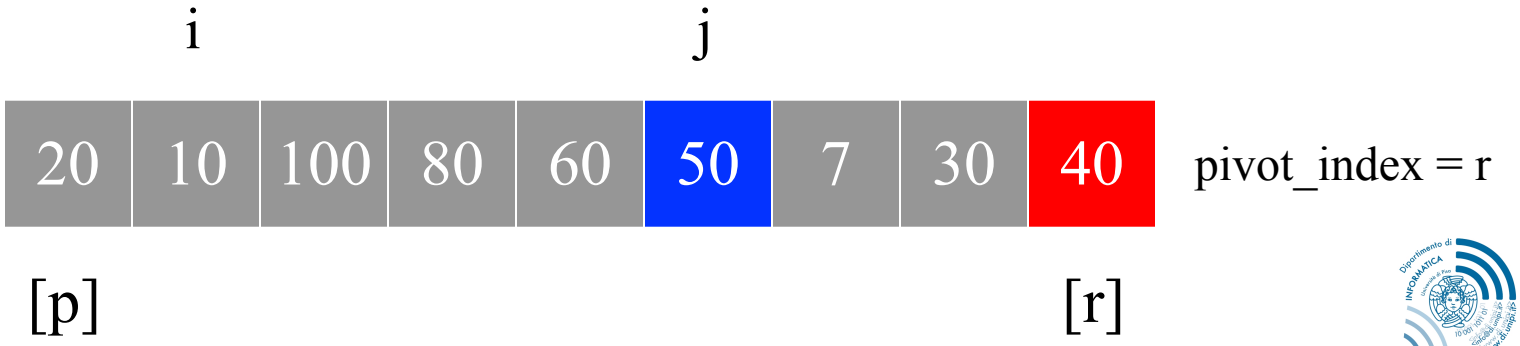
50 <? 40



Partiziona(A, p, r)

1. Inizializza *i = (p-1)* e *j = p*
2. **If** A[j] ≤ pivot
    1. i++
    2. Scambia A[i] e A[j]
    3. j++ e vai a 2.
3. **Else** j++ e vai a 2.

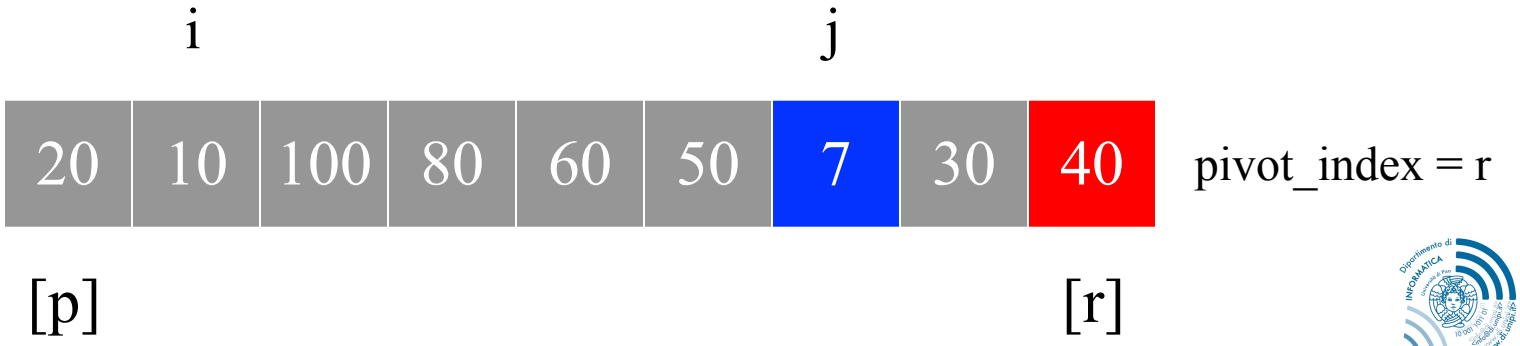
50 <? 40



Partiziona(A, p, r)

1. Inizializza *i = (p-1)* e *j = p*
2. **If** A[j] ≤ pivot
   1. i++
   2. Scambia A[i] e A[j]
   3. j++ e vai a 2.
3. **Else** j++ e vai a 2.

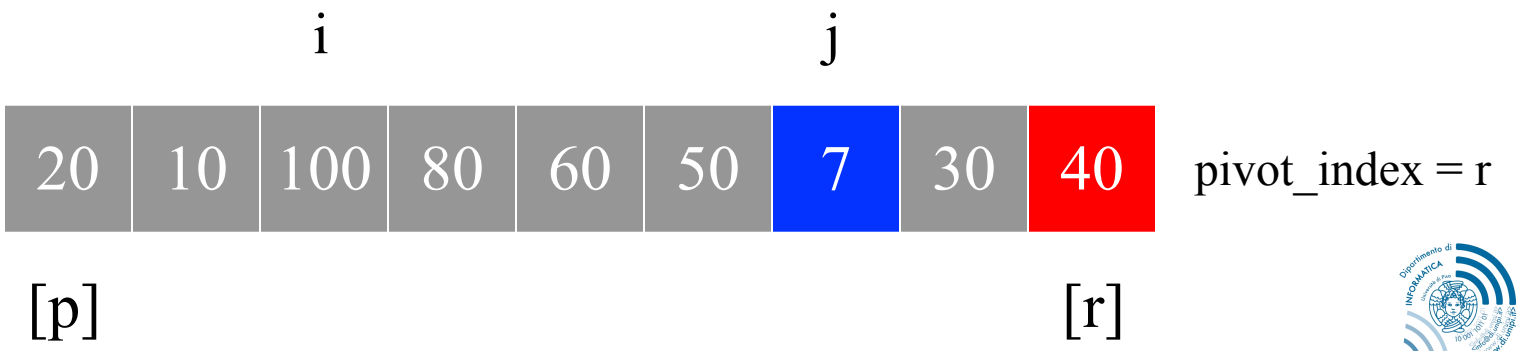
7 <? 40



Partiziona(A, p, r)

1. Inizializza *i = (p-1)* e *j = p*
2. **If** A[j] ≤ pivot
   1. i++
   2. Scambia A[i] e A[j]
   3. j++ e vai a 2.
3. **Else** j++ e vai a 2.

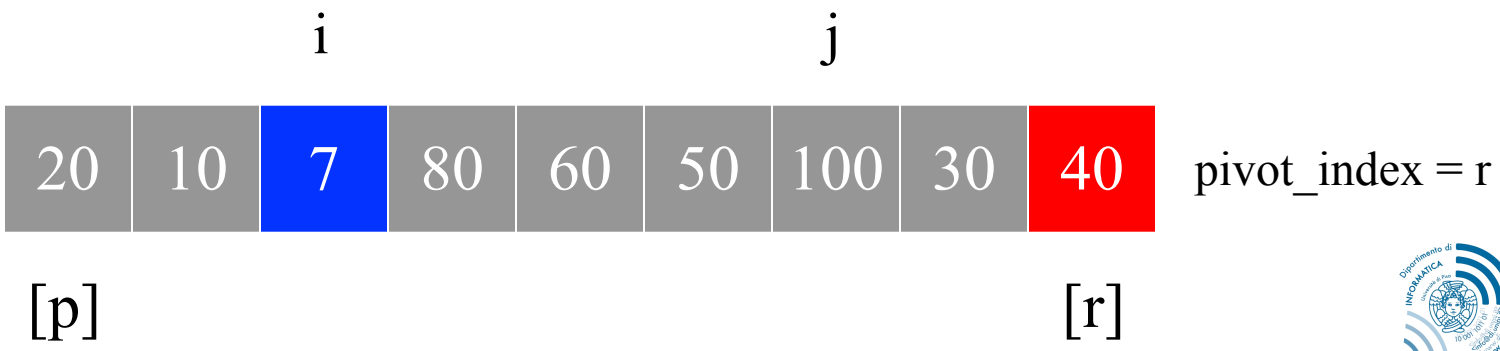
7 <? 40



Partiziona(A, p, r)

1. Inizializza *i = (p-1)* e *j = p*
2. **If** A[j] ≤ pivot
    1. i++
    2. Scambia A[i] e A[j]
    3. j++ e vai a 2.
3. **Else** j++ e vai a 2.

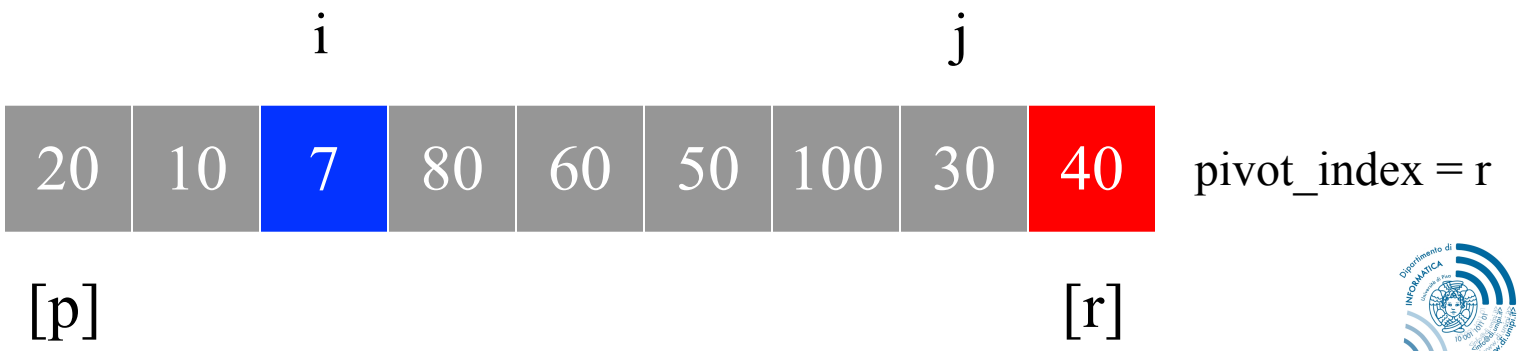
7 <? 40



Partiziona(A, p, r)

1. Inizializza *i = (p-1)* e *j = p*
2. **If** A[j] ≤ pivot
    1. i++
    2. Scambia A[i] e A[j]
    3. j++ e vai a 2.
3. **Else** j++ e vai a 2.

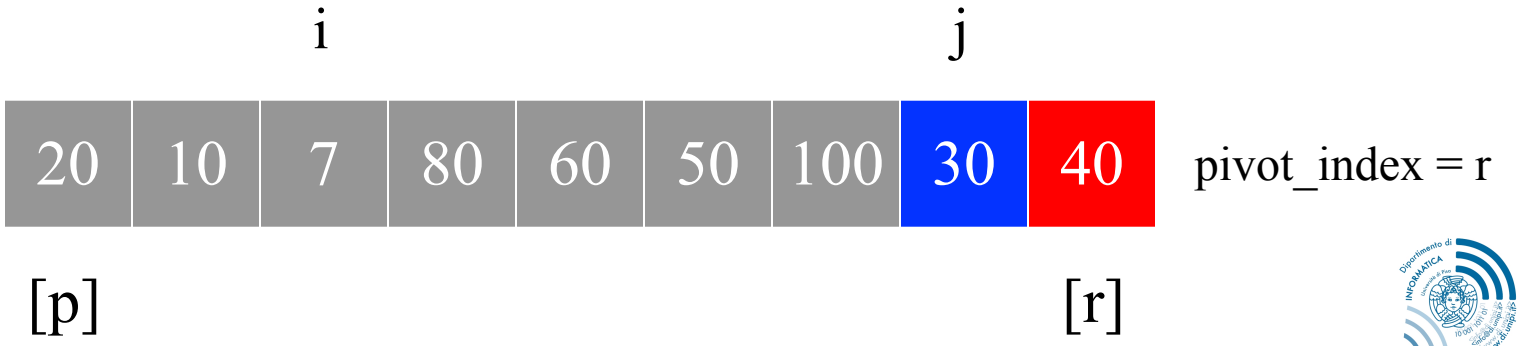
7 <? 40



Partiziona(A, p, r)

1. Inizializza *i = (p-1)* e *j = p*
2. **If** A[j] ≤ pivot
   1. i++
   2. Scambia A[i] e A[j]
   3. j++ e vai a 2.
3. **Else** j++ e vai a 2.

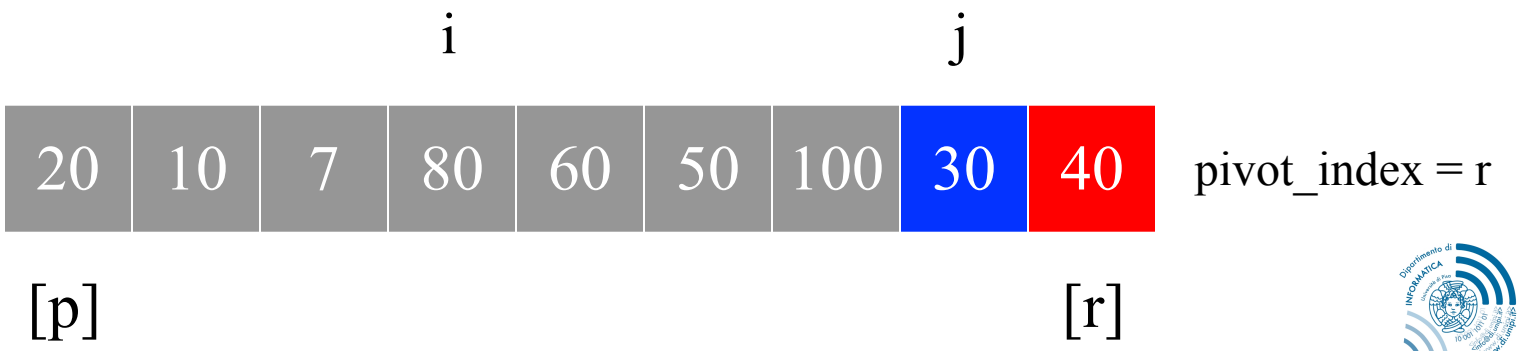
30 <? 40



Partiziona(A, p, r)

1. Inizializza *i = (p-1)* e *j = p*
2. **If** A[j] ≤ pivot
   1. i++
   2. Scambia A[i] e A[j]
   3. j++ e vai a 2.
3. **Else** j++ e vai a 2.

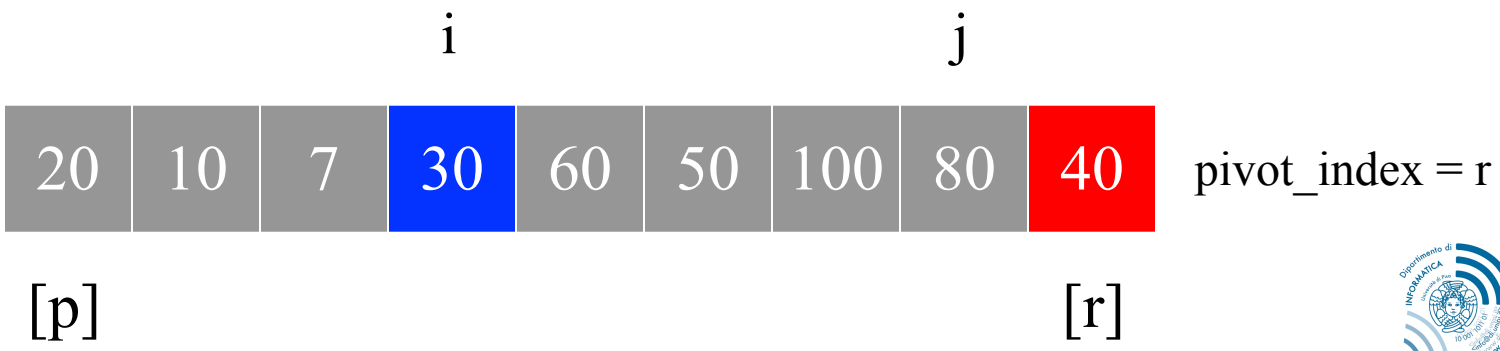
30 <? 40



Partiziona(A, p, r)

1. Inizializza *i = (p-1)* e *j = p*
2. **If** A[j] ≤ pivot
    1. i++
    2. Scambia A[i] e A[j]
    3. j++ e vai a 2.
3. **Else** j++ e vai a 2.

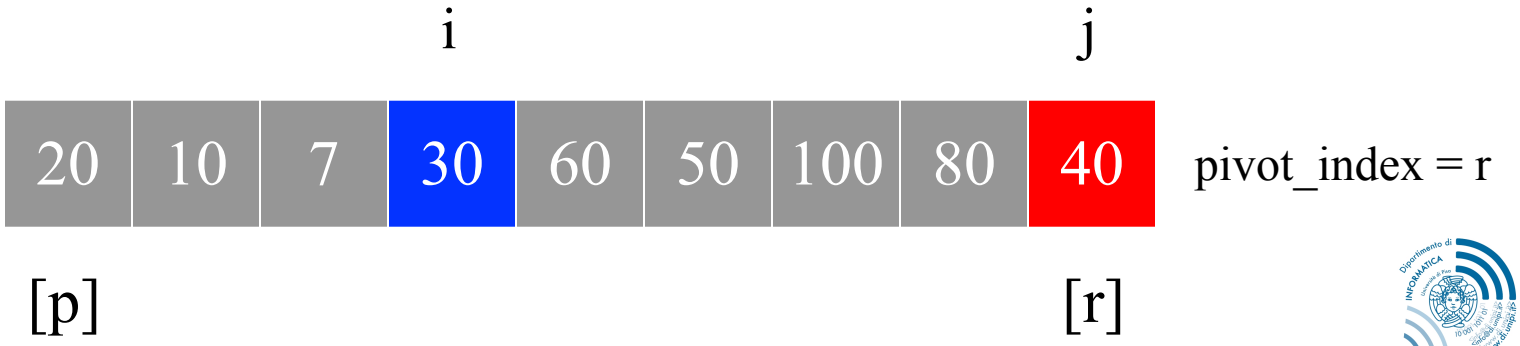
30 <? 40



Partiziona(A, p, r)

1. Inizializza *i = (p-1)* e *j = p*
2. **If** A[j] ≤ pivot
    1. i++
    2. Scambia A[i] e A[j]
    3. j++ e vai a 2.
3. **Else** j++ e vai a 2.

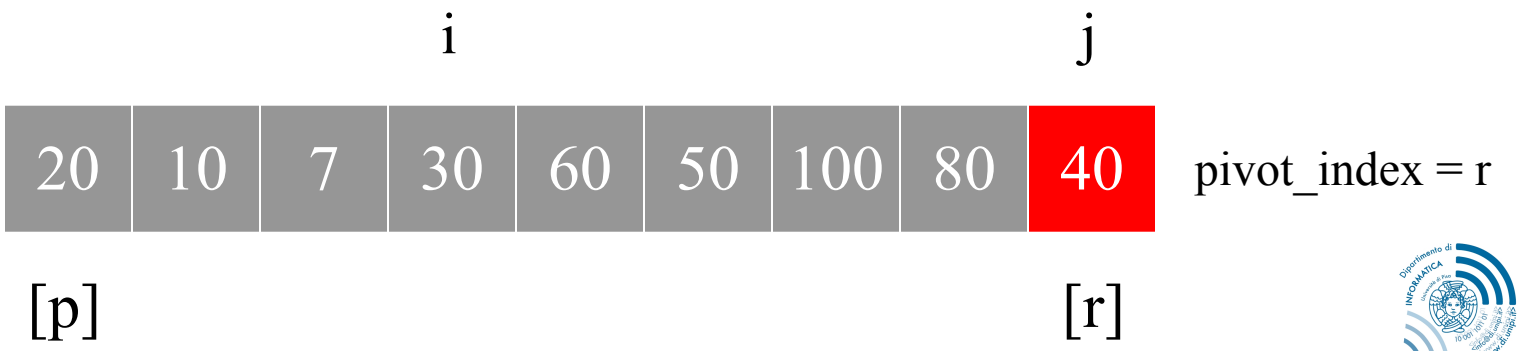
30 <? 40



Partiziona(A, p, r)

1. Inizializza *i = (p-1)* e *j = p*
2. **If** A[j] ≤ pivot
   1. i++
   2. Scambia A[i] e A[j]
   3. j++ e vai a 2.
3. **Else** j++ e vai a 2.

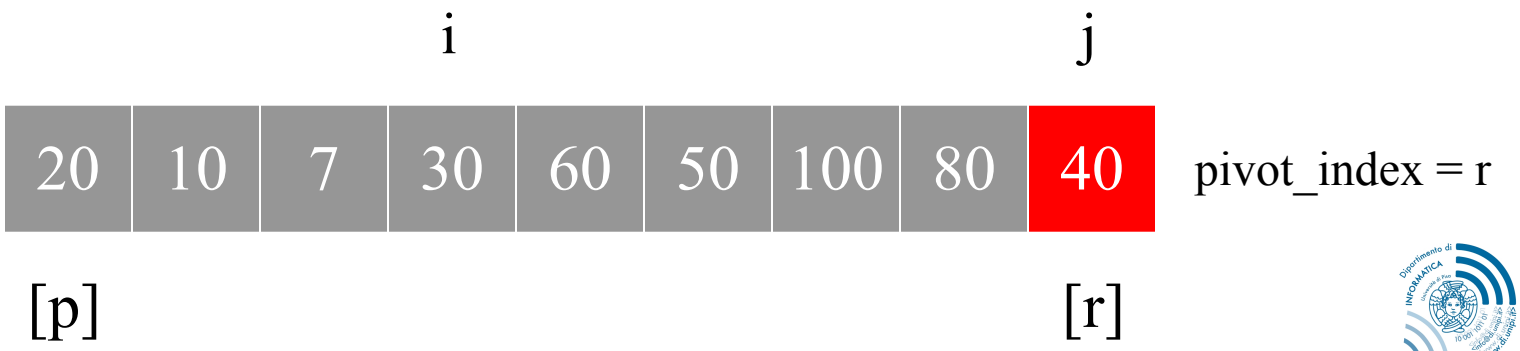
Termina quando j raggiunge il pivot!!!!



Partiziona(A, p, r)

1. Inizializza *i = (p-1)* e *j = p*
2. **If** A[j] ≤ pivot
    1. i++
    2. Scambia A[i] e A[j]
    3. j++ e vai a 2.
3. **Else** j++ e vai a 2.
4. Al termine, scambia A[i+1] e il pivot

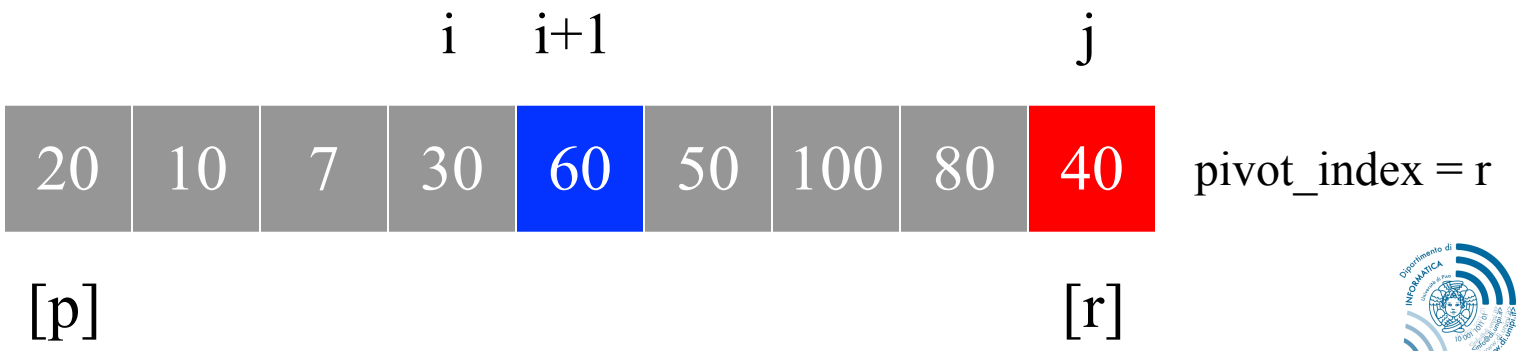
Termina quando j raggiunge il pivot!!!!



Partiziona(A, p, r)

1. Inizializza *i = (p-1)* e *j = p*
2. **If** A[j] ≤ pivot
    1. i++
    2. Scambia A[i] e A[j]
    3. j++ e vai a 2.
3. **Else** j++ e vai a 2.
4. Al termine, scambia A[i+1] e il pivot

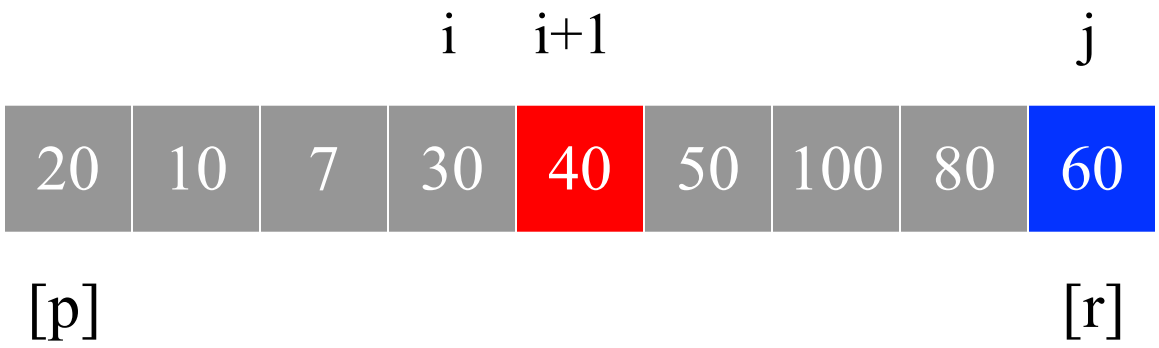
Termina quando j raggiunge il pivot!!!!



Partiziona(A, p, r)

1. Inizializza *i = (p-1)* e *j = p*
2. **If** A[j] ≤ pivot
    1. i++
    2. Scambia A[i] e A[j]
    3. j++ e vai a 2.
3. **Else** j++ e vai a 2.
4. Al termine, scambia A[i+1] e il pivot

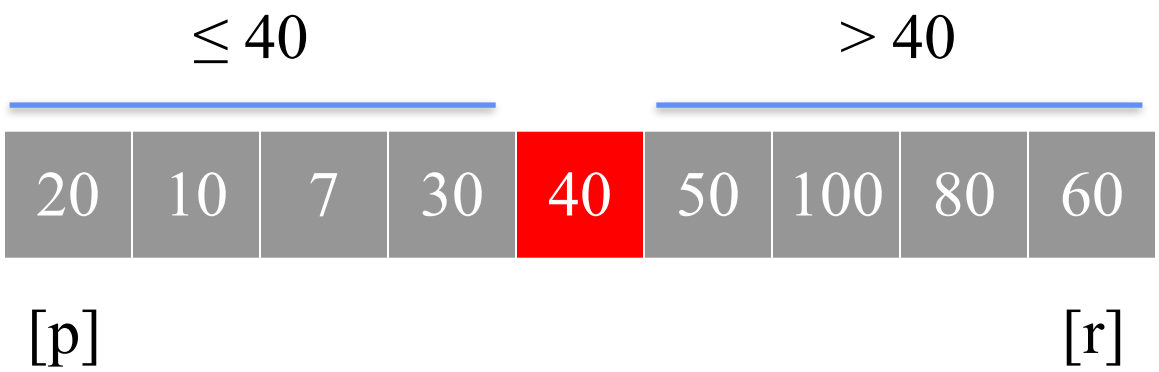
Termina quando j raggiunge il pivot!!!!



Partiziona(A, p, r)

1. Inizializza *i = (p-1)* e *j = p*
2. **If** A[j] ≤ pivot
    1. i++
    2. Scambia A[i] e A[j]
    3. j++ e vai a 2.
3. **Else** j++ e vai a 2.
4. Al termine, scambia A[i+1] e il pivot

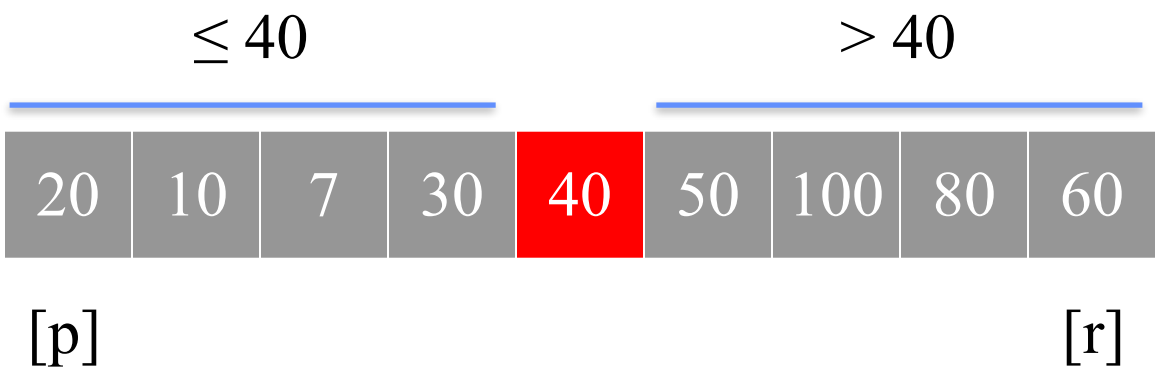
Termina quando j raggiunge il pivot!!!!



Partiziona(A, p, r)

Impera

Ordina ricorsivamente le due parti



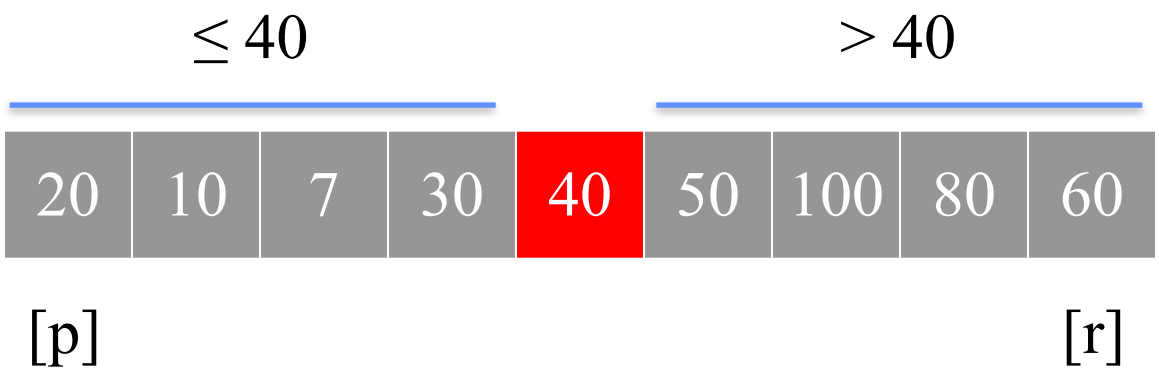
Partiziona(A, p, r)

Impera

Ordina ricorsivamente le due parti

Combina

Cosa fa????



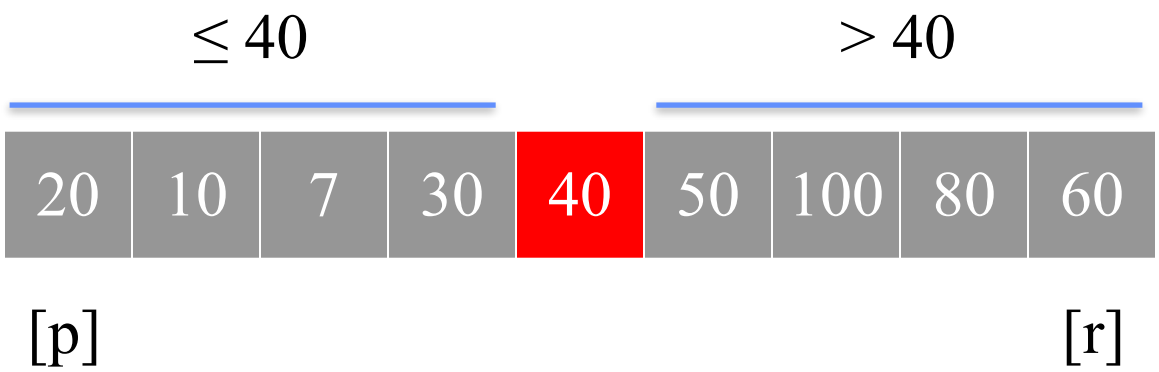
Partiziona(A, p, r)

Impera

Ordina ricorsivamente le due parti

Combina

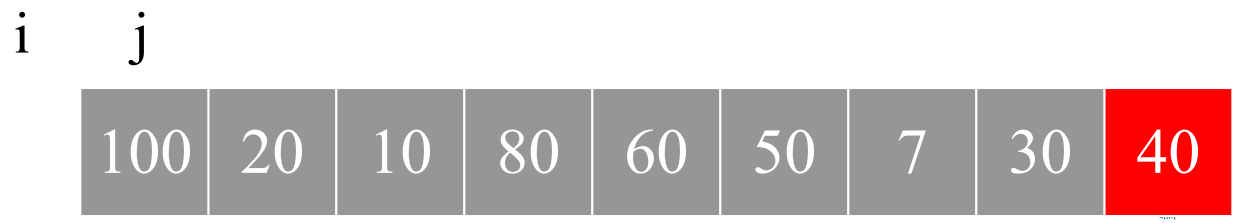
NULLA!!!!



Partiziona - Pseudocodice

Alg.: PARTIZIONA(A, p, r)

1. x = A[r]
2. i = p-1
3. **for** j = p **to** r-1
4. **if** A[j] ≤ x
5. i =i + 1
6. Scambia A[i] con A[j]
7. Scambia A[i+1] con A[r]
8. **return** i + 1



Partiziona - Pseudocodice

Alg.: PARTIZIONA(A, p, r)

1. x = A[r]
2. i = p-1
3. **for** j = p **to** r-1
4. **if** A[j] ≤ x
5. i =i + 1
6. Scambia A[i] con A[j]
7. Scambia A[i+1] con A[r]
8. **return** i + 1

Cosa c'è in posizione i + 1 alla fine?

100	20	10	80	60	50	7	30	40

Partiziona - Pseudocodice

Alg.: PARTIZIONA(A, p, r)

1. x = A[r]
2. i = p-1
3. **for** j = p **to** r-1
4. **if** A[j] ≤ x
5. i =i + 1
6. Scambia A[i] con A[j]
7. Scambia A[i+1] con A[r]
8. **return** i + 1

Alg.: QUICK-SORT(*A*, p, r)

if p < r

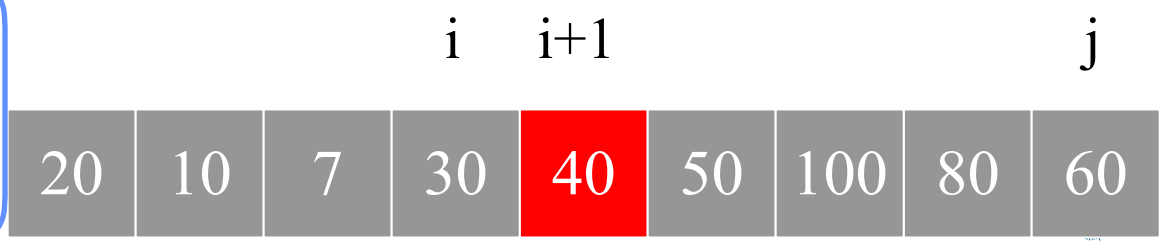
then *q* = PARTIZIONA(A, p, r)

QUICK-SORT(*A*, p, *q* - 1)

QUICK-SORT(*A*, *q* + 1, r)

return *A*

Il PIVOT!!!!



Partiziona - Correttezza

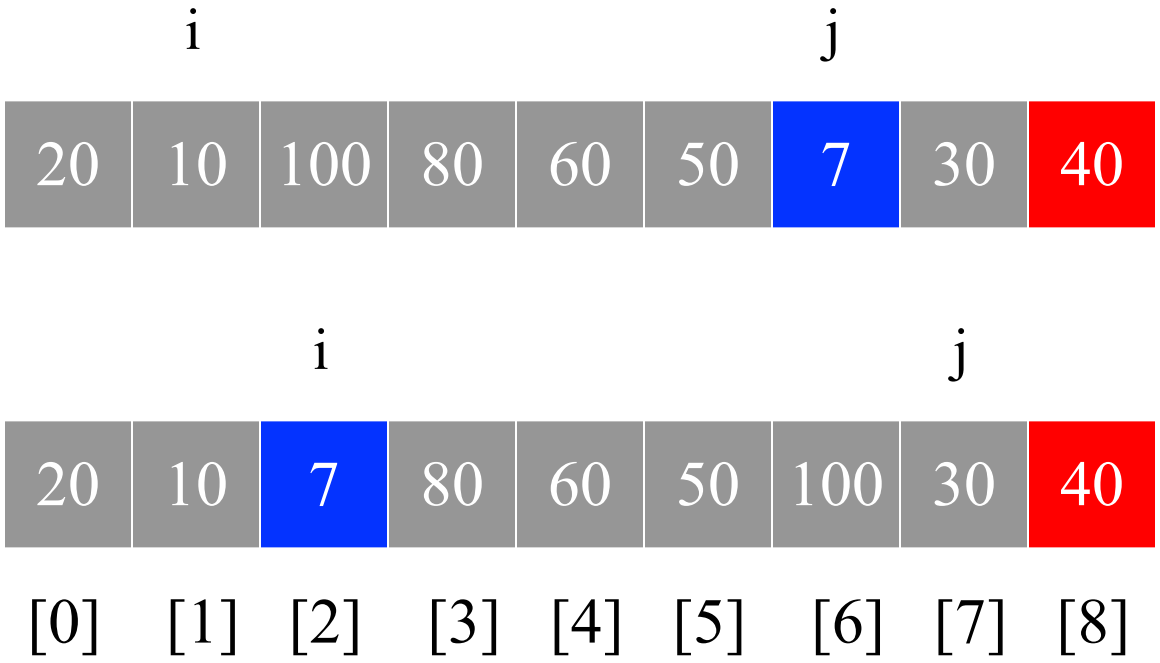
Invariante

Alg.: PARTIZIONA(A, p, r)

1. x = A[r]
2. i = p-1
3. **for** j = p **to** r-1
4. **if** A[j] ≤ x
5. i =i + 1
6. Scambia A[i] con A[j]
7. Scambia A[i+1] con A[r]
8. **return** i + 1

1. **Se** $p \leq k \leq i$, **allora** $A[k] \leq x$
2. **Se** $i+1 \leq k \leq j - 1$, **allora** $A[k] > x$
3. **Se** $k = r$, **allora** $A[k] = x$

1. $\mathbf{0 \leq k \leq 1}$
2. $\mathbf{2 \leq k \leq 5}$
3. $\mathbf{k = 8}$



Inizio ciclo 7

Fine ciclo 7

Partiziona - Correttezza

Alg.: PARTIZIONA(A, p, r)

```
1. x = A[r]
2.  i = p-1
3.  for j = p to r-1
4.        if A[ j ] ≤ x
5.              i =i + 1
6.              Scambia A[i] con A[j]
7. Scambia A[i+1] con A[r]
8. return i + 1
```

Invariante

1. **Se** $p \leq k \leq i$, **allora** $A[k] \leq x$
2. **Se** $i+1 \leq k \leq j - 1$, **allora** $A[k] > x$
3. **Se** $k = r$, **allora** $A[k] = x$

Da dimostrare:

1. **Inizializzazione**: invariante vera all'inizio del primo ciclo

2. **Mantenimento**: se vera prima di un ciclo, allora vera prima del prossimo

3. **Terminazione**: vera alla fine del ciclo

Partiziona - Correttezza

Alg.: PARTIZIONA(A, p, r)

```
1. x = A[r]
2.  i = p-1
3.  for j = p to r-1
4.        if A[ j ] ≤ x
5.              i =i + 1
6.              Scambia A[i] con A[j]
7. Scambia A[i+1] con A[r]
8. return i + 1
```

Invariante

1. **Se** $p \leq k \leq i$, **allora** $A[k] \leq x$
2. **Se** $i+1 \leq k \leq j - 1$, **allora** $A[k] > x$
3. **Se** $k = r$, **allora** $A[k] = x$

1. **Inizializzazione**: invariante vera all'inizio del primo ciclo

????

Partiziona - Correttezza

Invariante

Alg.: PARTIZIONA(A, p, r)

```
1. x = A[r]
2.  i = p-1
3.  for j = p to r-1
4.        if A[ j ] ≤ x
5.              i =i + 1
6.              Scambia A[i] con A[j]
7. Scambia A[i+1] con A[r]
8. return i + 1
```

1. **Se** $p \leq k \leq i$, **allora** $A[k] \leq x$
2. **Se** $i+1 \leq k \leq j - 1$, **allora** $A[k] > x$
3. **Se** $k = r$, **allora** $A[k] = x$

1. **Inizializzazione**: invariante vera all'inizio del primo ciclo

All'inizio, $i = p-1$ e $j = p$

Nessun valore tra p e i, e tra $i+1$ e $j-1$

Terza condizione soddisfatta dalla linea 1

Partiziona - Correttezza

Invariante

Alg.: PARTIZIONA(A, p, r)

```
1. x = A[r]
2.  i = p-1
3.  for j = p to r-1
4.        if A[ j ] ≤ x
5.              i =i + 1
6.              Scambia A[i] con A[j]
7. Scambia A[i+1] con A[r]
8. return i + 1
```

1. **Se** $p \leq k \leq i$, **allora** $A[k] \leq x$
2. **Se** $i+1 \leq k \leq j - 1$, **allora** $A[k] > x$
3. **Se** $k = r$, **allora** $A[k] = x$

2. **Mantenimento**: se vera prima di un ciclo, allora vera prima del prossimo

????

Partiziona - Correttezza

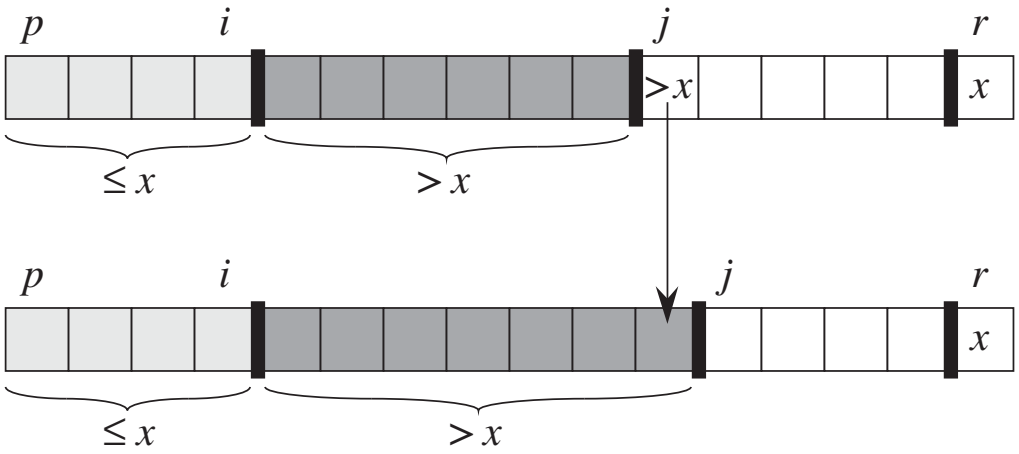
Invariante

Alg.: PARTIZIONA(A, p, r)

```
1. x = A[r]
2.  i = p-1
3.  for j = p to r-1
4.        if A[ j ] ≤ x
5.              i =i + 1
6.              Scambia A[i] con A[j]
7. Scambia A[i+1] con A[r]
8. return i + 1
```

1. **Se** $p \le k \le i$, **allora** $A[k] \le x$
2. **Se** $i+1 \le k \le j - 1$, **allora** $A[k] > x$
3. **Se** $k = r$, **allora** $A[k] = x$

2. **Mantenimento**: se vera prima di un ciclo, allora vera prima del prossimo



Caso A[j] > x

Partiziona - Correttezza

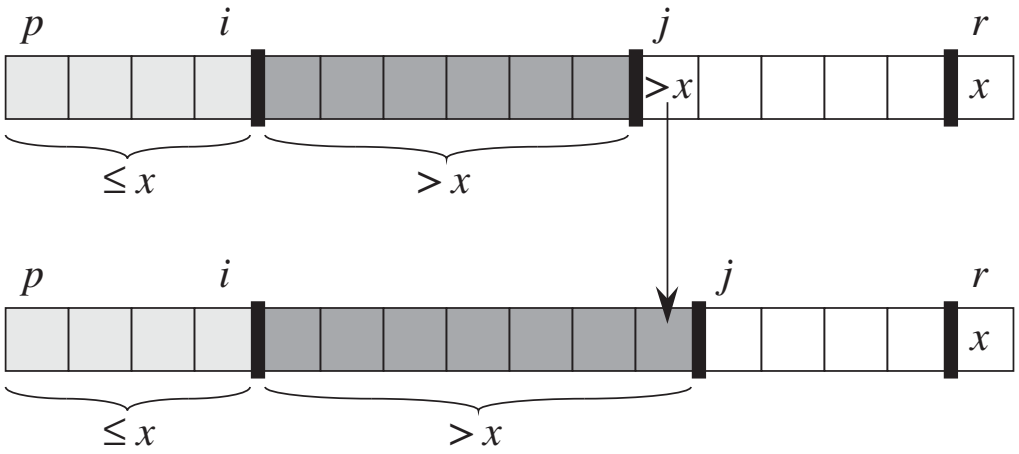
Invariante

Alg.: PARTIZIONA(A, p, r)

1. x = A[r]
2. i = p-1
3. **for** j = p **to** r-1
4. **if** A[j] ≤ x
5. i =i + 1
6. Scambia A[i] con A[j]
7. Scambia A[i+1] con A[r]
8. **return** i + 1

1. **Se** $p \le k \le i$, **allora** $A[k] \le x$
2. **Se** $i+1 \le k \le j - 1$, **allora** $A[k] > x$
3. **Se** $k = r$, **allora** $A[k] = x$

2. **Mantenimento**: se vera prima di un ciclo, allora vera prima del prossimo



Caso A[j] > x

OK!!!!

Partiziona - Correttezza

Invariante

Alg.: PARTIZIONA(A, p, r)

1. x = A[r]
2. i = p-1
3. **for** j = p **to** r-1
4. **if** A[j] ≤ x
5. i =i + 1
6. Scambia A[i] con A[j]
7. Scambia A[i+1] con A[r]
8. **return** i + 1

1. **Se** $p \leq k \leq i$, **allora** $A[k] \leq x$
2. **Se** $i+1 \leq k \leq j - 1$, **allora** $A[k] > x$
3. **Se** $k = r$, **allora** $A[k] = x$

2. **Mantenimento**: se vera prima di un ciclo, allora vera prima del prossimo

Caso $A[j] \leq x$

Inizio ciclo j

Partiziona - Correttezza

Invariante

Alg.: PARTIZIONA(A, p, r)

1. x = A[r]
2. i = p-1
3. **for** j = p **to** r-1
4. **if** A[j] ≤ x
5. i =i + 1
6. Scambia A[i] con A[j]
7. Scambia A[i+1] con A[r]
8. **return** i + 1

1. **Se** $p \leq k \leq i$, **allora** $A[k] \leq x$
2. **Se** $i+1 \leq k \leq j - 1$, **allora** $A[k] > x$
3. **Se** $k = r$, **allora** $A[k] = x$

2. **Mantenimento**: se vera prima di un ciclo, allora vera prima del prossimo

Caso $A[j] \leq x$

Inizio ciclo j

Partiziona - Correttezza

Invariante

Alg.: PARTIZIONA(A, p, r)

1. x = A[r]
2. i = p-1
3. **for** j = p **to** r-1
4. **if** A[j] ≤ x
5. i =i + 1
6. Scambia A[i] con A[j]
7. Scambia A[i+1] con A[r]
8. **return** i + 1

1. **Se** $p \leq k \leq i$, **allora** $A[k] \leq x$
2. **Se** $i+1 \leq k \leq j - 1$, **allora** $A[k] > x$
3. **Se** $k = r$, **allora** $A[k] = x$

2. **Mantenimento**: se vera prima di un ciclo, allora vera prima del prossimo

Caso $A[j] \leq x$

Inizio ciclo j

Partiziona - Correttezza

Invariante

Alg.: PARTIZIONA(A, p, r)

1. x = A[r]
2. i = p-1
3. **for** j = p **to** r-1
4. **if** A[j] ≤ x
5. i =i + 1
6. Scambia A[i] con A[j]
7. Scambia A[i+1] con A[r]
8. **return** i + 1

1. **Se** $p \leq k \leq i$, **allora** $A[k] \leq x$
2. **Se** $i+1 \leq k \leq j - 1$, **allora** $A[k] > x$
3. **Se** $k = r$, **allora** $A[k] = x$

2. **Mantenimento**: se vera prima di un ciclo, allora vera prima del prossimo

Caso $A[j] \leq x$

Fine ciclo j

Partiziona - Correttezza

Invariante

Alg.: PARTIZIONA(A, p, r)

1. x = A[r]
2. i = p-1
3. **for** j = p **to** r-1
4. **if** A[j] ≤ x
5. i =i + 1
6. Scambia A[i] con A[j]
7. Scambia A[i+1] con A[r]
8. **return** i + 1

1. **Se** $p \leq k \leq i$, **allora** $A[k] \leq x$
2. **Se** $i+1 \leq k \leq j - 1$, **allora** $A[k] > x$
3. **Se** $k = r$, **allora** $A[k] = x$

2. **Mantenimento**: se vera prima di un ciclo, allora vera prima del prossimo

Caso $A[j] \leq x$

Fine ciclo j

OK!!!!

Partiziona - Correttezza

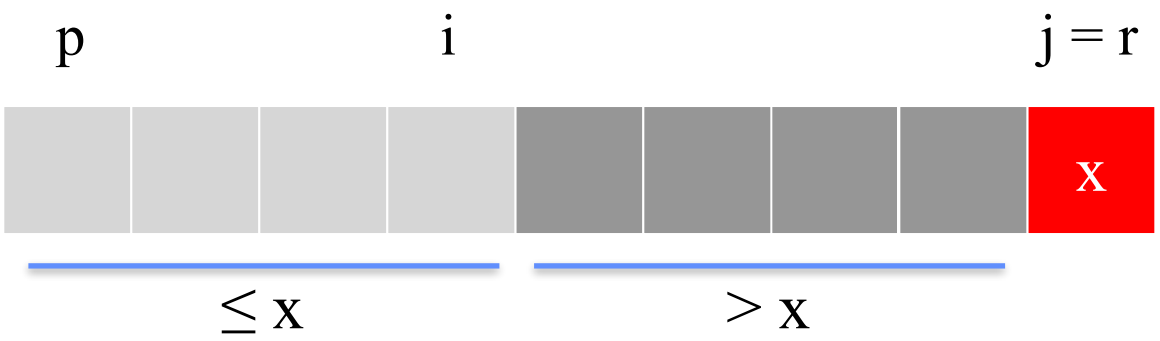
Invariante

Alg.: PARTIZIONA(A, p, r)

1. x = A[r]
2. i = p-1
3. **for** j = p **to** r-1
4. **if** A[j] ≤ x
5. i =i + 1
6. Scambia A[i] con A[j]
7. Scambia A[i+1] con A[r]
8. **return** i + 1

1. **Se** $p \leq k \leq i$, **allora** $A[k] \leq x$
2. **Se** $i+1 \leq k \leq j - 1$, **allora** $A[k] > x$
3. **Se** $k = r$, **allora** $A[k] = x$

3. **Terminazione**: vera alla fine del ciclo



Alla fine, $j = r$

Partiziona - Correttezza

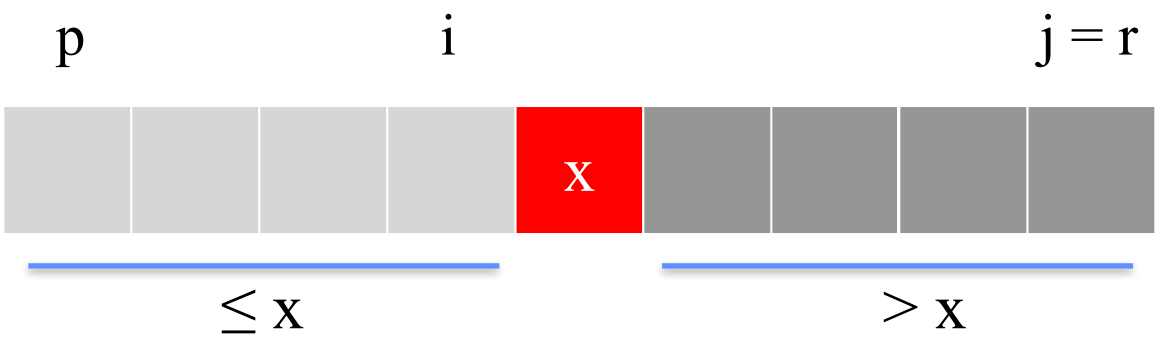
Invariante

Alg.: PARTIZIONA(A, p, r)

1. x = A[r]
2. i = p-1
3. **for** j = p **to** r-1
4. **if** A[j] ≤ x
5. i =i + 1
6. Scambia A[i] con A[j]
7. Scambia A[i+1] con A[r]
8. **return** i + 1

1. **Se** $p \leq k \leq i$, **allora** $A[k] \leq x$
2. **Se** $i+1 \leq k \leq j - 1$, **allora** $A[k] > x$
3. **Se** $k = r$, **allora** $A[k] = x$

3. **Terminazione**: vera alla fine del ciclo



Alla fine, $j = r$

OK!!!!

Partiziona - Costo

Invariante

Alg.: PARTIZIONA(A, p, r)

```
1. x = A[r]
2.  i = p-1
3.  for j = p to r-1
4.        if A[ j ] ≤ x
5.              i =i + 1
6.              Scambia A[i] con A[j]
7. Scambia A[i+1] con A[r]
8. return i + 1
```

????

Partiziona - Costo

Invariante

Alg.: PARTIZIONA(A, p, r)

1. x = A[r]
2. i = p-1
3. **for** j = p **to** r-1
4. **if** A[j] ≤ x
5. i =i + 1
6. Scambia A[i] con A[j]
7. Scambia A[i+1] con A[r]
8. **return** i + 1

$\Theta(n)$

QuickSort - Costo

Alg.: QUICK-SORT(*A*, p, r)

```
if p < r
  then q = PARTIZIONA(A, p, r)        ▷  Θ(n)
       QUICK-SORT(A, p, q - 1)        ▷
       QUICK-SORT(A, q + 1, r)        ▷
```

Caso migliore?

QuickSort - Costo

Alg.: QUICK-SORT(*A*, p, r)

```
if p < r
  then q = PARTIZIONA(A, p, r)        ▷  Θ(n)
       QUICK-SORT(A, p, q - 1)        ▷
       QUICK-SORT(A, q + 1, r)        ▷
```

Caso migliore? $T(n) = 2T(n/2) + \Theta(n)$

QuickSort - Costo

Alg.: QUICK-SORT(*A*, p, r)

```
if p < r
  then q = PARTIZIONA(A, p, r)        ▷  Θ(n)
       QUICK-SORT(A, p, q - 1)        ▷
       QUICK-SORT(A, q + 1, r)        ▷
```

Caso migliore? $T(n) = 2T(n/2) + \Theta(n)$

Master Th. 2: Θ(nlg n)

QuickSort - Costo

Alg.: QUICK-SORT(*A*, p, r)

if p < r

then q = PARTIZIONA(A, p, r)

QUICK-SORT(*A*, p, q - 1)

QUICK-SORT(*A*, q + 1, r)

Caso peggiore?

Alg.: PARTIZIONA(A, p, r)

1. x = A[r]
2. i = p-1
3. **for** j = p **to** r-1
4. **if** A[j] ≤ x
5. i =i + 1
6. Scambia A[i] con A[j]
7. Scambia A[i+1] con A[r]
8. **return** i + 1

▷ **Θ(n)**

▷

▷

Provate!!!!

QuickSort - Costo

Alg.: QUICK-SORT(*A*, p, r)

if p < r

then *q* = PARTIZIONA(A, p, r) ▷ $\Theta(n)$

QUICK-SORT(*A*, p, *q* - 1) ▷

QUICK-SORT(*A*, *q* + 1, r) ▷

Alg.: PARTIZIONA(A, p, r)

1. x = A[r]
2. i = p-1
3. **for** j = p **to** r-1
4. **if** A[j] ≤ x
5. i =i + 1
6. Scambia A[i] con A[j]
7. Scambia A[i+1] con A[r]
8. **return** i + 1

Caso peggiore?

Primo giro di PARTIZIONA **restituisce:** ????

QuickSort - Costo

Alg.: QUICK-SORT(*A*, p, r)

if p < r

then q = PARTIZIONA(A, p, r) ▷ $\Theta(n)$

QUICK-SORT(*A*, p, q - 1) ▷

QUICK-SORT(*A*, q + 1, r) ▷

Alg.: PARTIZIONA(A, p, r)

```
1. x = A[r]
2.  i = p-1
3.  for j = p to r-1
4.        if A[ j ] ≤ x
5.              i =i + 1
6.              Scambia A[i] con A[j]
7.    Scambia A[i+1] con A[r]
8.  return i + 1
```

0	1	2	3	4	5

Caso peggiore?

Primo giro di PARTIZIONA restituisce: **5**

Dimensioni Left e Right delle ricorsioni? **n-1, 0**

QuickSort - Costo

Alg.: QUICK-SORT(*A*, p, r)

if p < r

then q = PARTIZIONA(A, p, r) ▷ $\Theta(n)$

QUICK-SORT(*A*, p, q - 1) ▷

QUICK-SORT(*A*, q + 1, r) ▷

Alg.: PARTIZIONA(A, p, r)

```
1. x = A[r]
2.  i = p-1
3.  for j = p to r-1
4.        if A[ j ] ≤ x
5.              i =i + 1
6.              Scambia A[i] con A[j]
7.    Scambia A[i+1] con A[r]
8.  return i + 1
```

Caso peggiore?

Secondo giro di PARTIZIONA **restituisce:** ?

Dimensioni Left e Right delle ricorsioni? ?, ?

QuickSort - Costo

Alg.: QUICK-SORT(*A*, p, r)

if p < r

then q = PARTIZIONA(A, p, r) ▷ $\Theta(n)$

QUICK-SORT(*A*, p, q - 1) ▷

QUICK-SORT(*A*, q + 1, r) ▷

Alg.: PARTIZIONA(A, p, r)

1. x = A[r]
2. i = p-1
3. **for** j = p **to** r-1
4. **if** A[j] ≤ x
5. i =i + 1
6. Scambia A[i] con A[j]
7. Scambia A[i+1] con A[r]
8. **return** i + 1

Caso peggiore?

Secondo giro di PARTIZIONA restituisce: 4

Dimensioni Left e Right delle ricorsioni? ?, ?

QuickSort - Costo

Alg.: QUICK-SORT(*A*, p, r)

if p < r

then q = PARTIZIONA(A, p, r) ▷ $\Theta(n)$

QUICK-SORT(*A*, p, q - 1) ▷

QUICK-SORT(*A*, q + 1, r) ▷

Alg.: PARTIZIONA(A, p, r)

```
1. x = A[r]
2.  i = p-1
3.  for j = p to r-1
4.        if A[ j ] ≤ x
5.              i =i + 1
6.              Scambia A[i] con A[j]
7.    Scambia A[i+1] con A[r]
8.  return i + 1
```

Caso peggiore?

Secondo giro di PARTIZIONA restituisce: **4**

Dimensioni Left e Right delle ricorsioni? **n-2, 0**

QuickSort - Costo

Alg.: QUICK-SORT(*A*, p, r)

if p < r

then q = PARTIZIONA(A, p, r) ▷ $\Theta(n)$

QUICK-SORT(*A*, p, q - 1) ▷

QUICK-SORT(*A*, q + 1, r) ▷

Alg.: PARTIZIONA(A, p, r)

1. x = A[r]
2. i = p-1
3. **for** j = p **to** r-1
4. **if** A[j] ≤ x
5. i =i + 1
6. Scambia A[i] con A[j]
7. Scambia A[i+1] con A[r]
8. **return** i + 1

Caso peggiore?

Terzo giro di PARTIZIONA **restituisce:** ?

Dimensioni Left e Right delle ricorsioni? ?, ?

QuickSort - Costo

Alg.: QUICK-SORT(*A*, p, r)

if p < r

then q = PARTIZIONA(A, p, r) ▷ $\Theta(n)$

QUICK-SORT(*A*, p, q - 1) ▷

QUICK-SORT(*A*, q + 1, r) ▷

Alg.: PARTIZIONA(A, p, r)

```
1. x = A[r]
2.  i = p-1
3.  for j = p to r-1
4.        if A[ j ] ≤ x
5.              i =i + 1
6.              Scambia A[i] con A[j]
7.    Scambia A[i+1] con A[r]
8.  return i + 1
```

Caso peggiore?

Terzo giro di PARTIZIONA restituisce: **3**

Dimensioni Left e Right delle ricorsioni? **n-3, 0**

QuickSort - Costo

Alg.: QUICK-SORT(*A*, p, r)

if p < r

then *q* = PARTIZIONA(A, p, r)

QUICK-SORT(*A*, p, *q* - 1)

QUICK-SORT(*A*, *q* + 1, r)

Alg.: PARTIZIONA(A, p, r)

```
1. x = A[r]
2.  i = p-1
3.  for j = p to r-1
4.        if A[ j ] ≤ x
5.              i =i + 1
6.              Scambia A[i] con A[j]
7.    Scambia A[i+1] con A[r]
8.  return i + 1
```

▷ **Θ(n)**

▷

▷

Caso peggiore?

Quindi, in generale T(n) = ?

QuickSort - Costo

Alg.: QUICK-SORT(*A*, p, r)

if p < r

then q = PARTIZIONA(A, p, r) ▷ $\Theta(n)$

QUICK-SORT(*A*, p, q - 1) ▷

QUICK-SORT(*A*, q + 1, r) ▷

Alg.: PARTIZIONA(A, p, r)

1. x = A[r]
2. i = p-1
3. **for** j = p **to** r-1
4. **if** A[j] ≤ x
5. i =i + 1
6. Scambia A[i] con A[j]
7. Scambia A[i+1] con A[r]
8. **return** i + 1

Caso peggiore?

$$T(n) = T(n-1) + T(0) + \Theta(n)$$

$\Theta(?)$

QuickSort - Costo

Alg.: QUICK-SORT(*A*, p, r)

if p < r

then q = PARTIZIONA(A, p, r) ▷ $\Theta(n)$

QUICK-SORT(*A*, p, q - 1) ▷

QUICK-SORT(*A*, q + 1, r) ▷

Alg.: PARTIZIONA(A, p, r)

1. x = A[r]
2. i = p-1
3. **for** j = p **to** r-1
4. **if** A[j] ≤ x
5. i =i + 1
6. Scambia A[i] con A[j]
7. Scambia A[i+1] con A[r]
8. **return** i + 1

Caso peggiore?

$$T(n) = T(n-1) + T(0) + \Theta(n)$$

$\Theta(n^2)$

Peggio di Insertion Sort nel caso migliore (array già ordinato)!!!!

Caso peggiore?

$$T(n) = T(n-1) + T(0) + \Theta(n)$$

$\Theta(n^2)$

QuickSort - Analisi Costo Medio

L'idea è basata sulla **distribuzione** dei dati

Infatti, è **piuttosto improbabile** che i dati vengano partizionati sempre nel modo peggiore ad ogni chiamata ricorsiva

In media, infatti, ci aspettiamo che **alcune** partizioni siano **buone** e **altre meno buone**

Partiziona - Pivot Casuale

Per cercare di evitare sempre il caso meno buono, **mischiamo** un pò le carte usando **randomizzazione**

Non scegliamo sempre ultimo come pivot….

Partiziona - Pivot Casuale

Alg.: PARTIZIONA(A, p, r)

```
1.  x = A[r]
2.   i = p-1
3.   for j = p to r-1
4.         if A[ j ] ≤ x
5.               i =i + 1
6.               Scambia A[i] con A[j]
7.     Scambia A[i+1] con A[r]
8.  return i + 1
```

Per cercare di evitare sempre il caso meno buono, **mischiamo** un pò le carte usando **randomizzazione**

Non scegliamo sempre ultimo come pivot….

Partiziona - Pivot Casuale

Alg.: PARTIZIONA(A, p, r)

```
1.  x = A[r]
2.   i = p-1
3.   for j = p to r-1
4.         if A[ j ] ≤ x
5.               i =i + 1
6.               Scambia A[i] con A[j]
7.     Scambia A[i+1] con A[r]
8.  return i + 1
```

Per cercare di evitare sempre il caso meno buono, **mischiamo** un pò le carte usando **randomizzazione**

Non scegliamo sempre ultimo come pivot….

Alg.: PARTIZIONA-RANDOMIZZATO(A, p, r)

```
1.  i = RANDOM(p, r)
2.  Scambia A[i] con A[r]
3.  return PARTIZIONA(A, p, r)
```

QuickSort Randomizzato

Alg.: QUICK-SORT-RANDOMIZZATO(A, p, r)

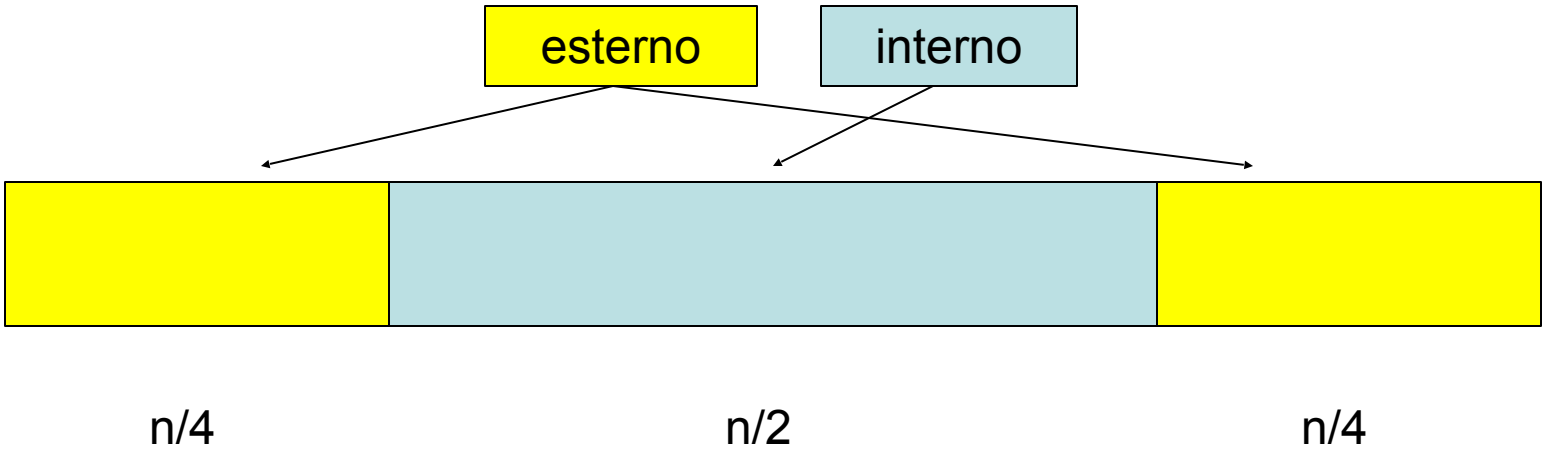
```
if p < r
   then q = PARTIZIONA-RANDOMIZZATO(A, p, r)
        QUICK-SORT-RANDOMIZZATO(A, p, q - 1)
        QUICK-SORT-RANDOMIZZATO(A, q + 1, r)
```

QuickSort Randomizzato - Analisi

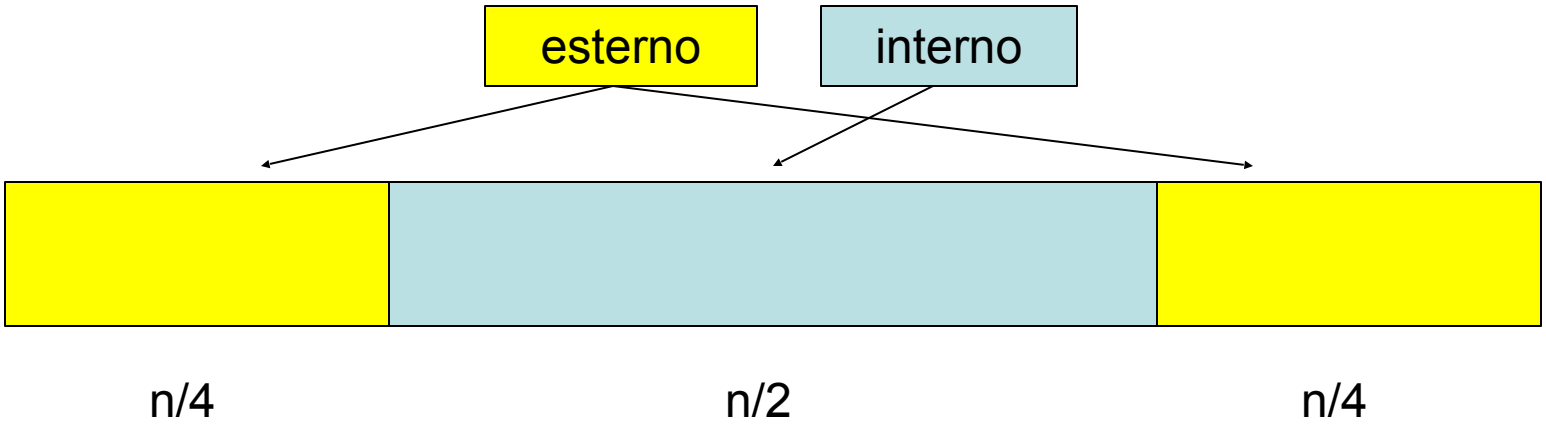
Seleziona ogni volta il pivot in modo uniforme e casuale nell'intervallo [p, r]

Definiamo due eventi **equiprobabili** (ossia con probabilità ½ ciascuno)

1. Pivot esterno (***dopo** aver partizionato*)
2. Pivot interno (***dopo** aver partizionato*)



QuickSort Randomizzato - Analisi



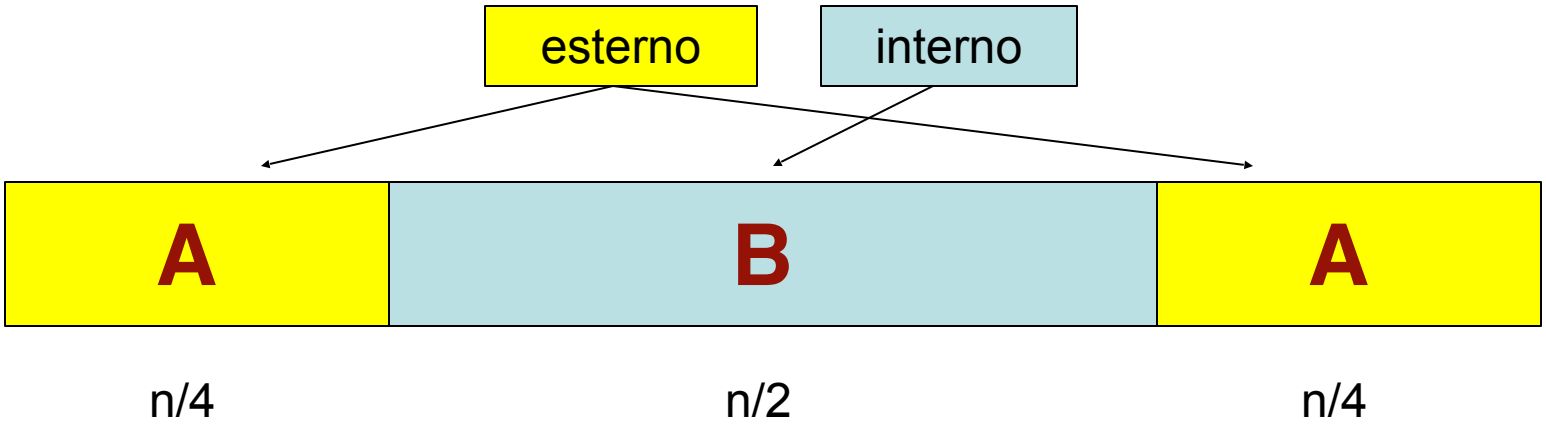
T(n) = costo medio del QS randomizzato

Chiamiamo:

A= T(n) quando il perno è **esterno**

B= T(n) quando il perno è **interno**

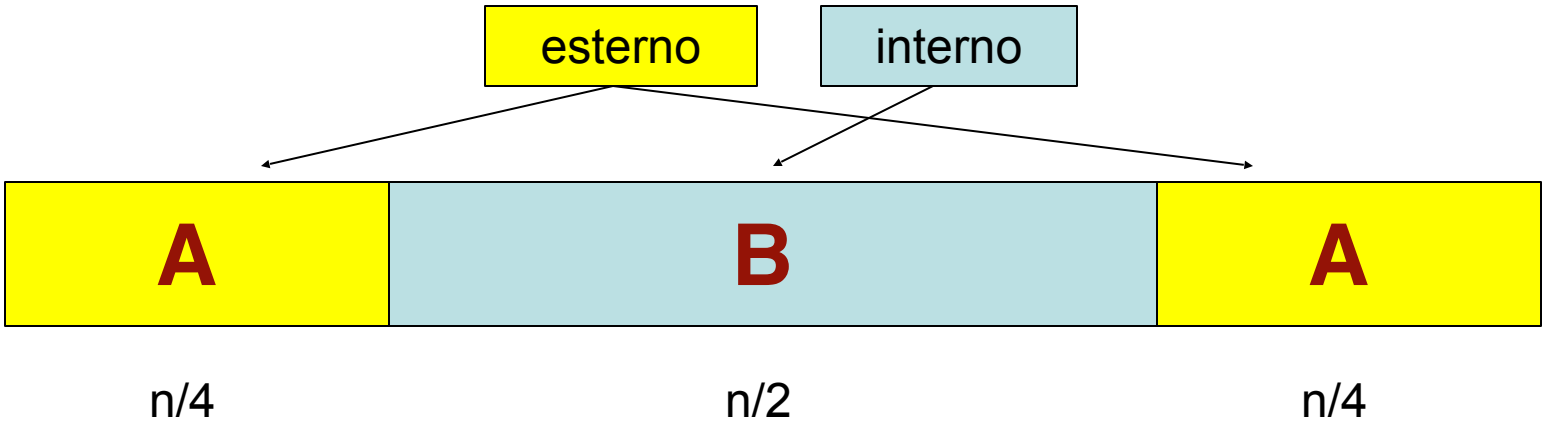
QuickSort Randomizzato - Analisi



A: Quando si verifica la situazione peggiore quando il perno è **esterno**?

B: Quando si verifica la situazione peggiore quando il perno è **interno**?

QuickSort Randomizzato - Analisi

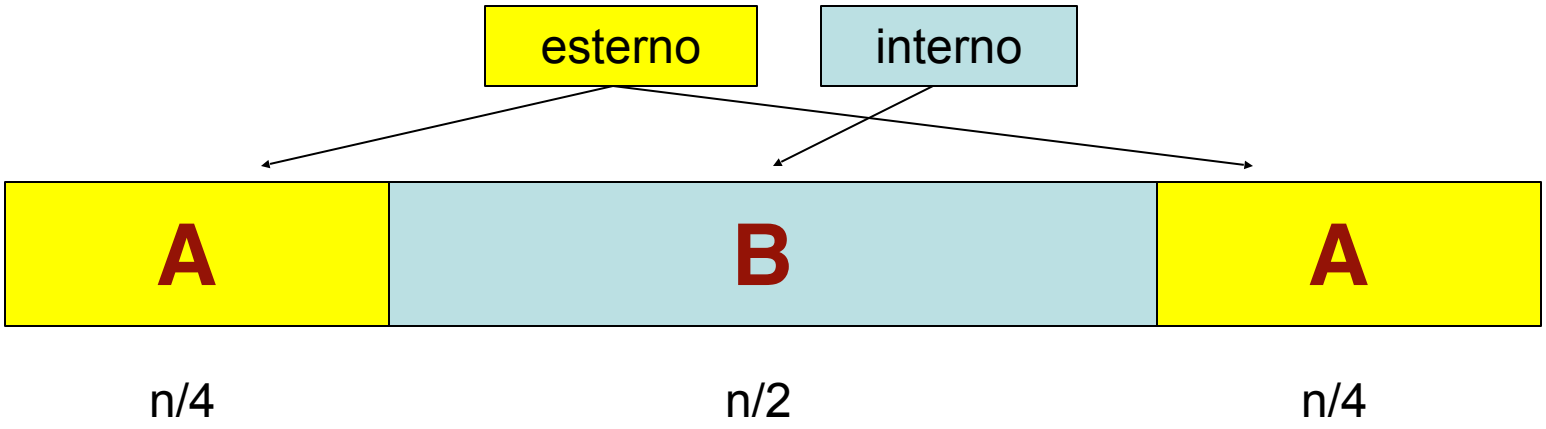


A: Quando si verifica la situazione peggiore quando il perno è **esterno**?

$$T(n) = T(n-1) + O(n)$$

B: Quando si verifica la situazione peggiore quando il perno è **interno**?

QuickSort Randomizzato - Analisi



A: Quando si verifica la situazione peggiore quando il perno è **esterno**?

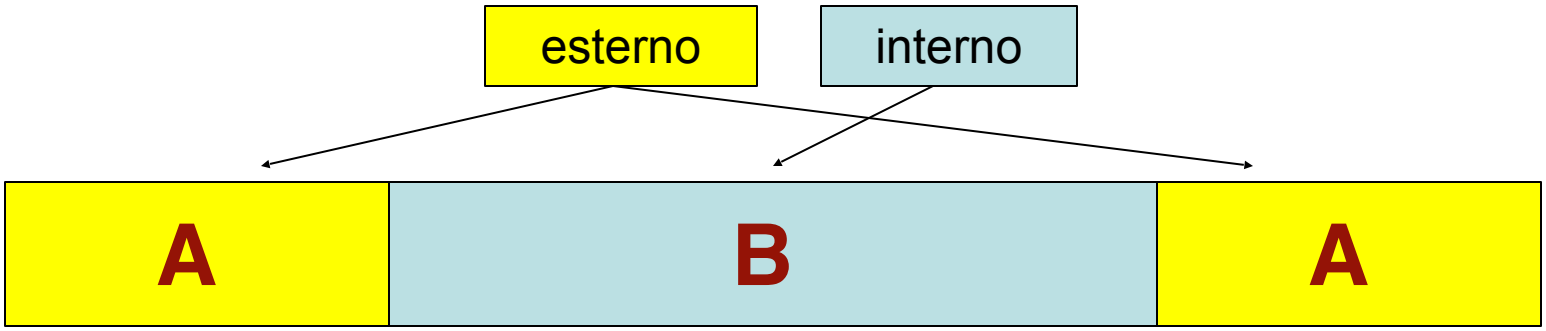
$$T(n) = T(n-1) + O(n)$$

B: Quando si verifica la situazione peggiore quando il perno è **interno**?

$$T(n) = T(\tfrac{1}{4}\, n) + T(\tfrac{3}{4}\, n) + O(n)$$

Perno è uno dei due estremi del segmento centrale

QuickSort Randomizzato - Analisi



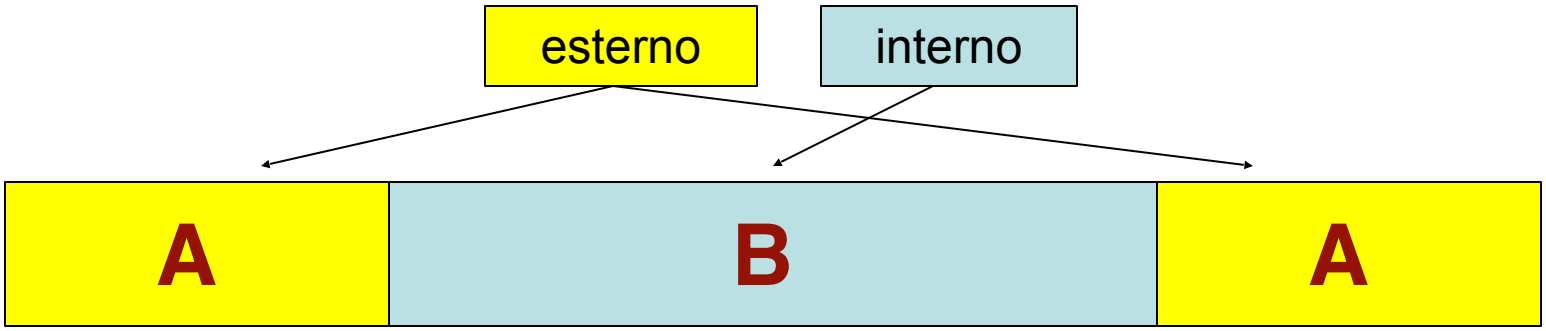
n/4 n/2 n/4

A **B**

$T(n) = T(n-1) + O(n)$ **$T(n) = T(¼\ n) + T(¾\ n) + O(n)$**

Abbiamo assunto A e B equiprobabili

QuickSort Randomizzato - Analisi



n/4 n/2 n/4

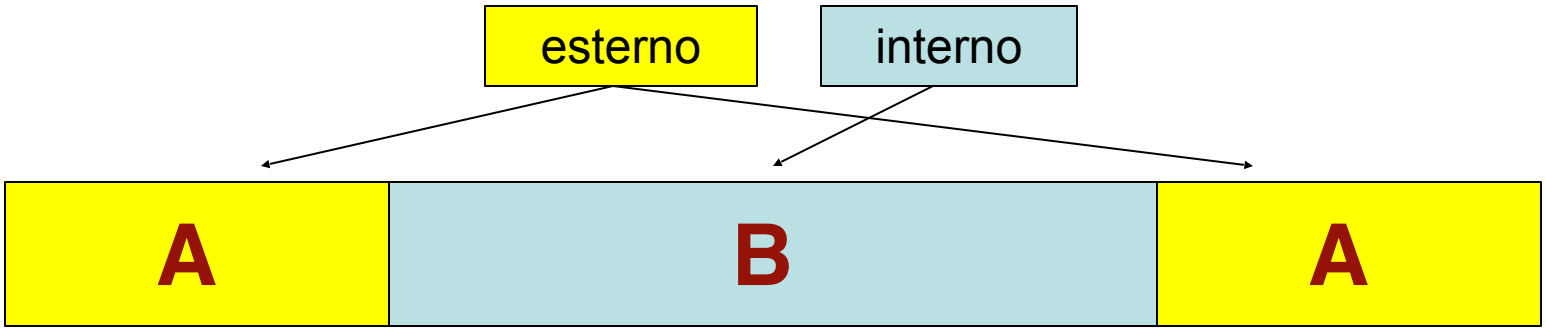
A **B**

$T(n) = T(n-1) + O(n)$ **$T(n) = T(\frac{1}{4}\, n) + T(\frac{3}{4}\, n) + O(n)$**

Abbiamo assunto A e B equiprobabili

$T(n)$

QuickSort Randomizzato - Analisi



n/4 n/2 n/4

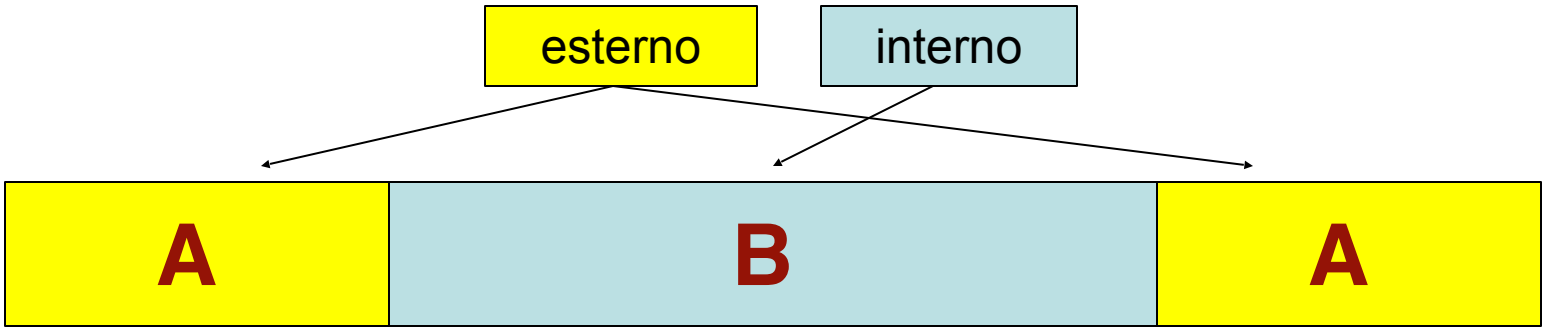
A **B**

$T(n) = T(n-1) + O(n)$ **$T(n) = T(¼\ n) + T(¾\ n) + O(n)$**

Abbiamo assunto A e B equiprobabili

$T(n) \leq ½\ A + ½\ B = ½\ [A+B]$

QuickSort Randomizzato - Analisi



n/4 n/2 n/4

A **B**

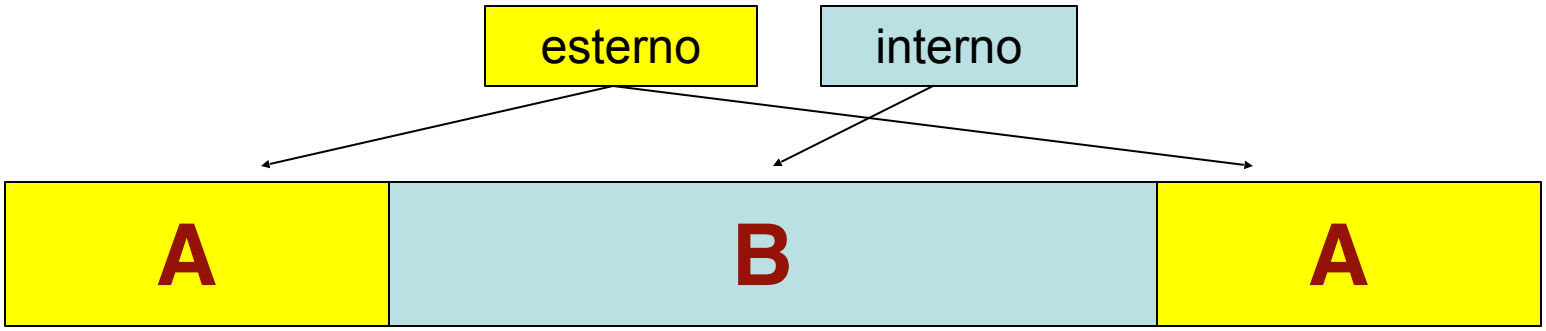
$T(n) = T(n-1) + O(n)$ **$T(n) = T(¼\ n) + T(¾\ n) + O(n)$**

Abbiamo assunto A e B equiprobabili

$T(n) \leq ½\ A + ½\ B = ½\ [A+B]$

$= ½\ [T(n-1) + T(¼\ n) + T(¾\ n) + O(n)]$

QuickSort Randomizzato - Analisi



n/4 n/2 n/4

A **B**

$T(n) = T(n-1) + O(n)$ **$T(n) = T(¼\ n) + T(¾\ n) + O(n)$**

Abbiamo assunto A e B equiprobabili

$T(n) \leq ½\ A + ½\ B = ½\ [A+B]$

$= ½\ [T(n-1) + T(¼\ n) + T(¾\ n) + O(n)]$

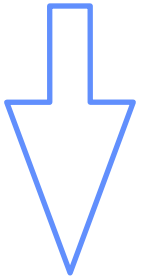
$\leq ½\ [T(n) + T(¼\ n) + T(¾\ n) + O(n)]$

QuickSort Randomizzato - Analisi

$$T(n) \leq \tfrac{1}{2} [T(n) + T(\tfrac{1}{4}\, n) + T(\tfrac{3}{4}\, n) + O(n)]$$

QuickSort Randomizzato - Analisi

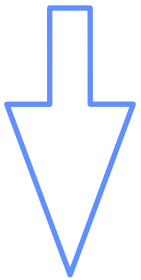
$$T(n) \leq \tfrac{1}{2} [T(n) + T(\tfrac{1}{4} n) + T(\tfrac{3}{4} n) + O(n)]$$



Moltiplichiamo per 2 a dx e sx

QuickSort Randomizzato - Analisi

$$T(n) \leq \tfrac{1}{2} [T(n) + T(\tfrac{1}{4} n) + T(\tfrac{3}{4} n) + O(n)]$$

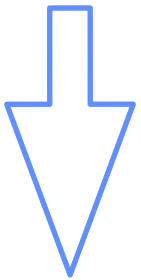


Moltiplichiamo per 2 a dx e sx

$$2T(n) \leq T(n) + T(\tfrac{1}{4} n) + T(\tfrac{3}{4} n) + O(n)$$

QuickSort Randomizzato - Analisi

$$T(n) \leq \tfrac{1}{2} [T(n) + T(\tfrac{1}{4} n) + T(\tfrac{3}{4} n) + O(n)]$$



Moltiplichiamo per 2 a dx e sx

$$2T(n) \leq T(n) + T(\tfrac{1}{4} n) + T(\tfrac{3}{4} n) + O(n)$$

$$T(n) \leq T(\tfrac{1}{4} n) + T(\tfrac{3}{4} n) + O(n)$$

QuickSort Randomizzato - Analisi

$$T(n) \leq T(\tfrac{1}{4}\, n) + T(\tfrac{3}{4}\, n) + O(n)$$

Come risolviamo?

QuickSort Randomizzato - Analisi

$$T(n) \leq T(\tfrac{1}{4}\, n) + T(\tfrac{3}{4}\, n) + O(n)$$

Come risolviamo?

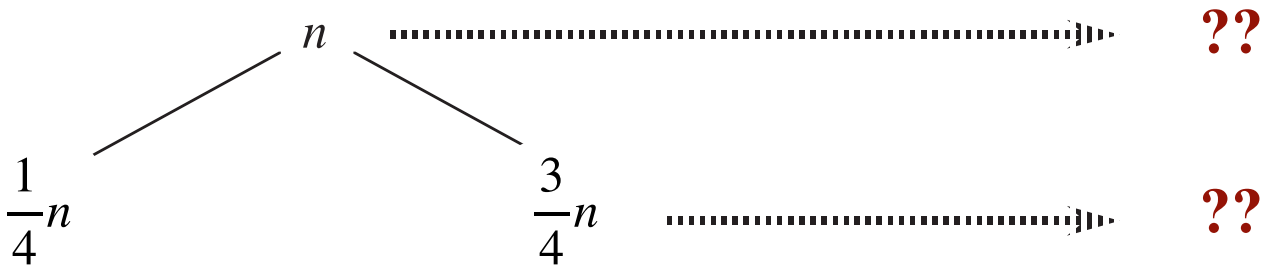
Albero!

QuickSort - Costo

Primi 2 livelli albero di ricorrenza?

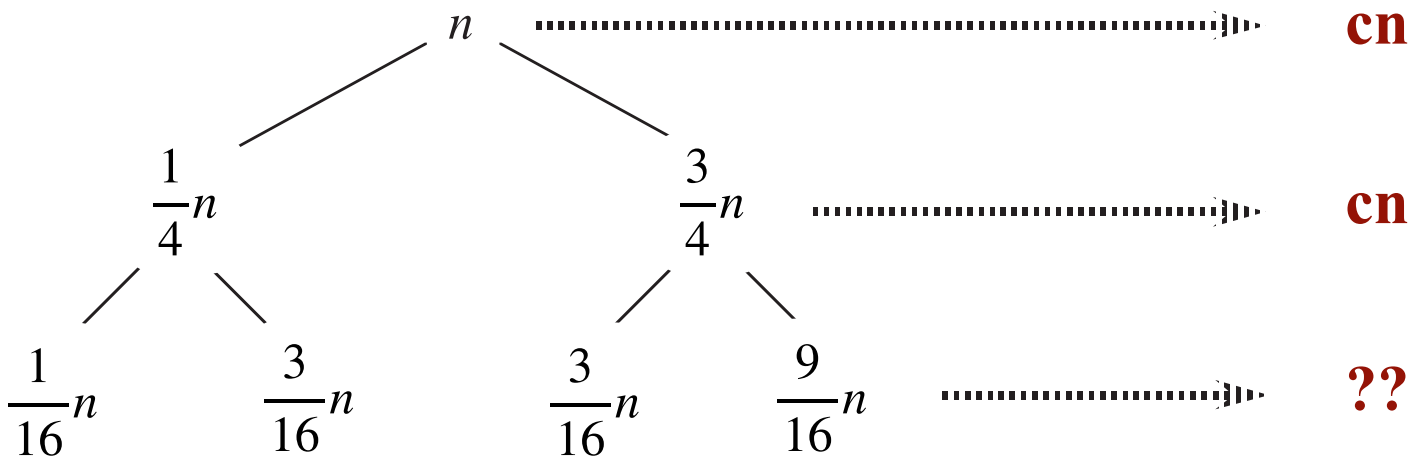
$$T(n) = T\left(\frac{n}{4}\right) + T\left(\frac{3}{4}n\right) + n$$

QuickSort - Costo



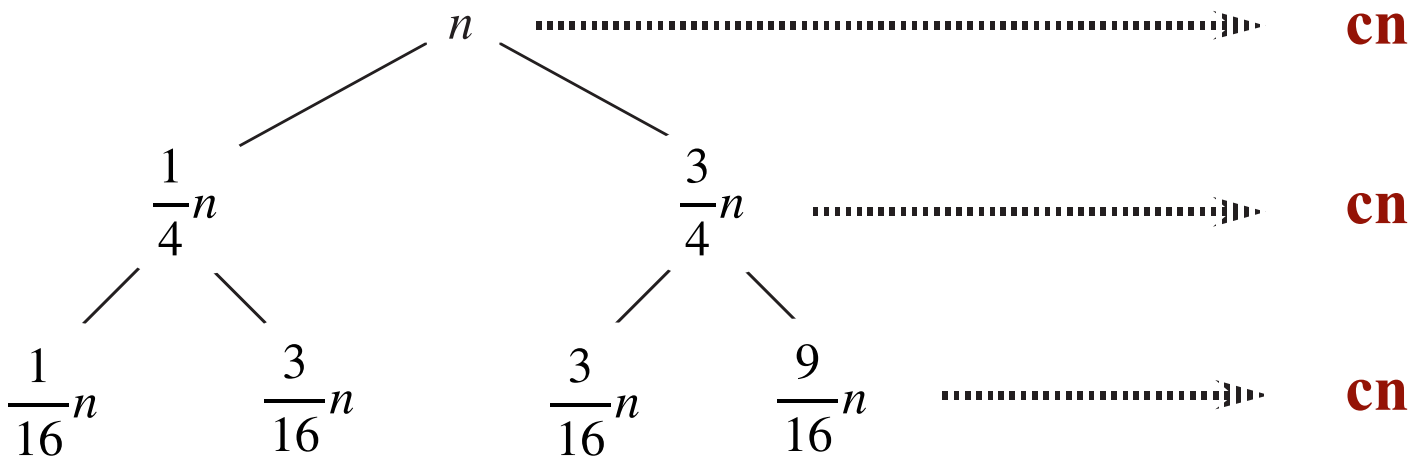
$$T(n) = T\left(\frac{n}{4}\right) + T\left(\frac{3}{4}n\right) + n$$

QuickSort - Costo



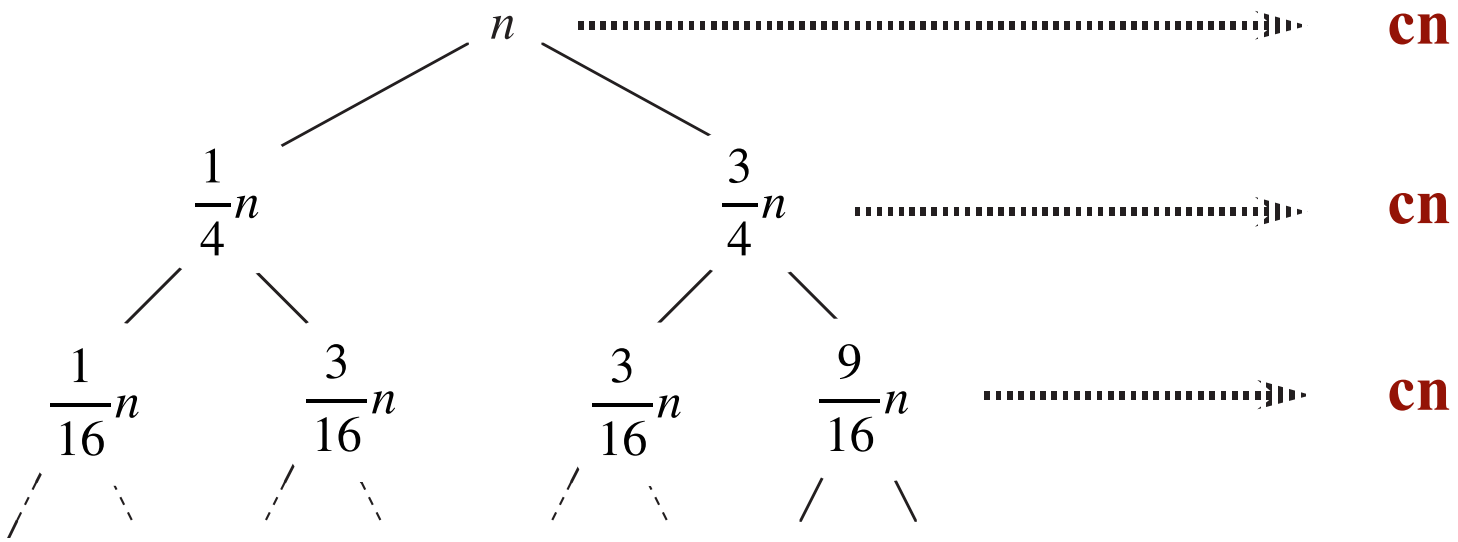
$$T(n) = T\left(\frac{n}{4}\right) + T\left(\frac{3}{4}n\right) + n$$

QuickSort - Costo



$$T(n) = T\left(\frac{n}{4}\right) + T\left(\frac{3}{4}n\right) + n$$

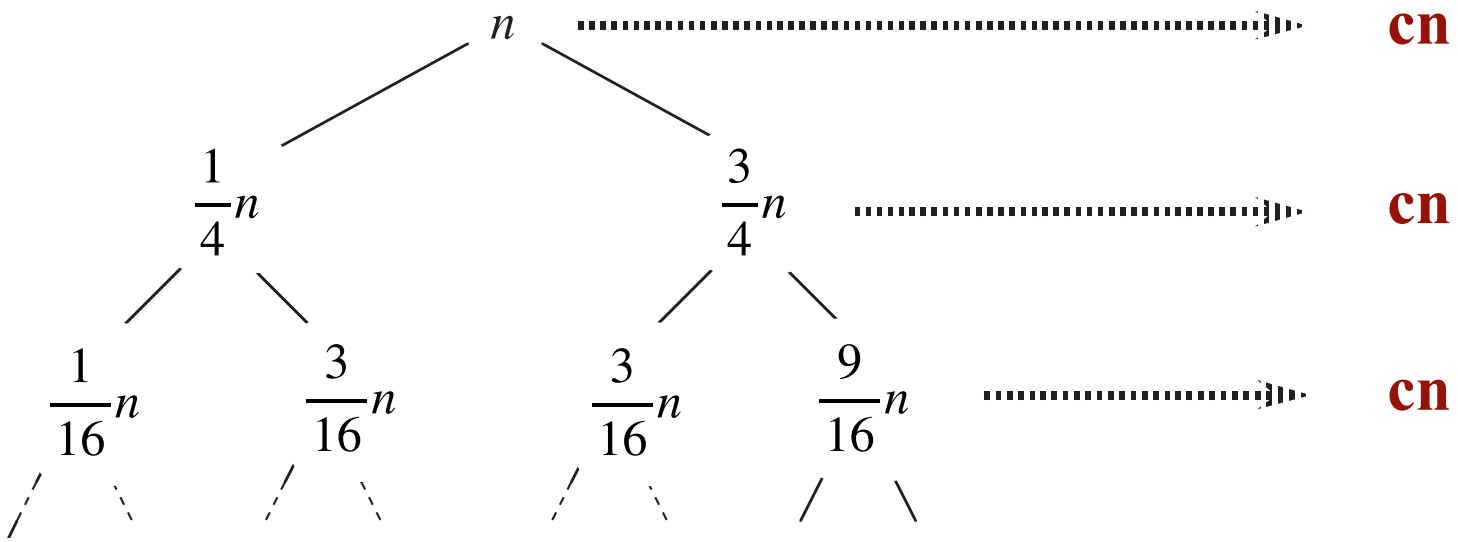
QuickSort - Costo



L'albero verrà bilanciato?

$$T(n) = T\left(\frac{n}{4}\right) + T\left(\frac{3}{4}n\right) + n$$

QuickSort - Costo

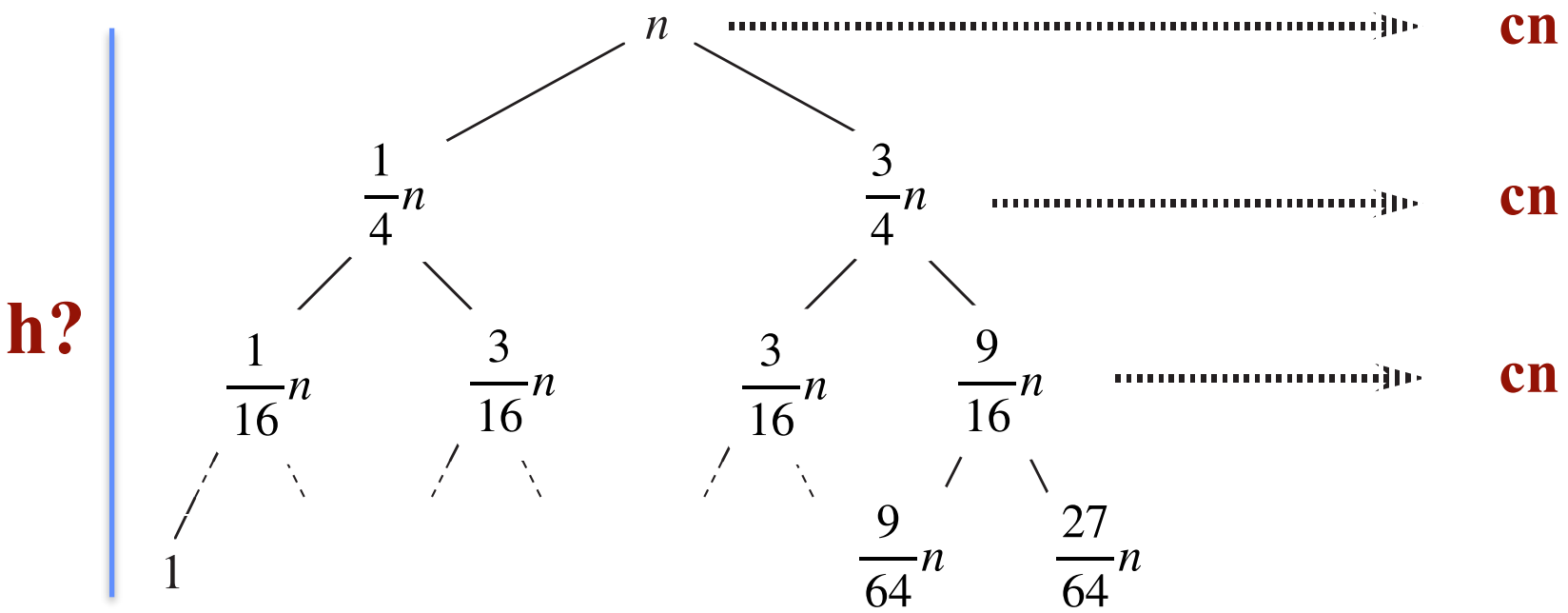


No!

Da quale lato arriveremo prima alle foglie?

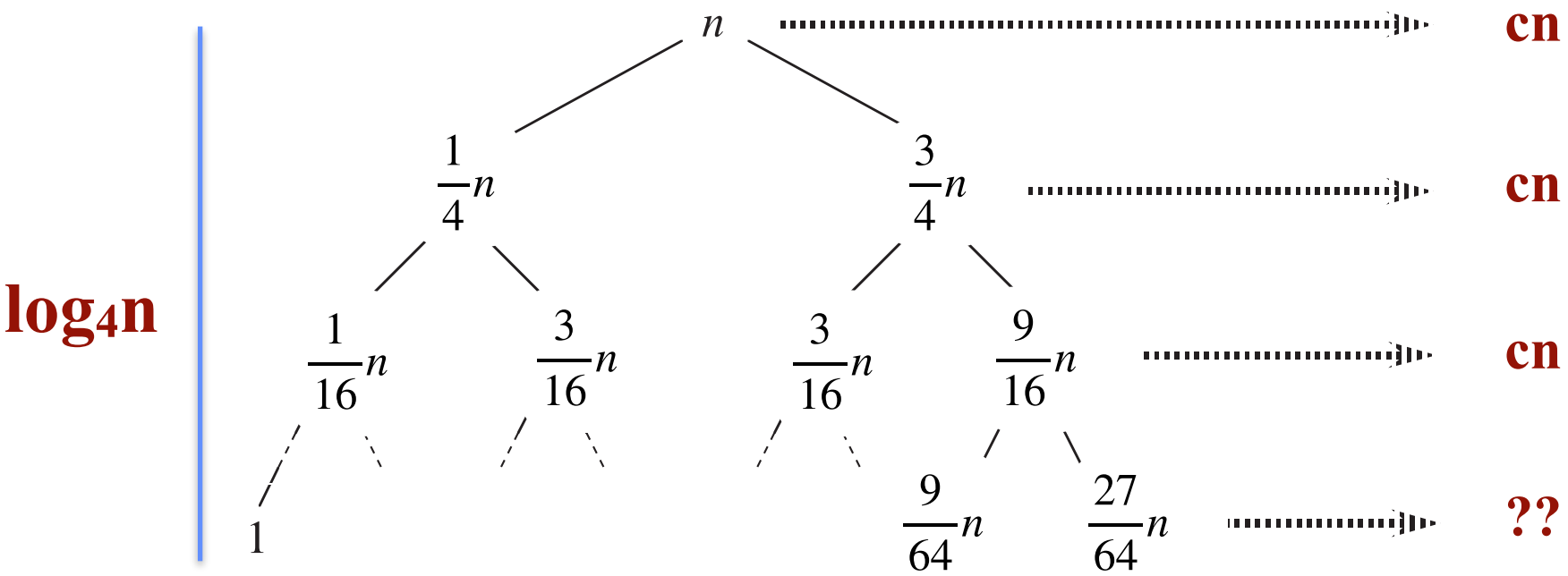
$$T(n) = T\left(\frac{n}{4}\right) + T\left(\frac{3}{4}n\right) + n$$

QuickSort - Costo



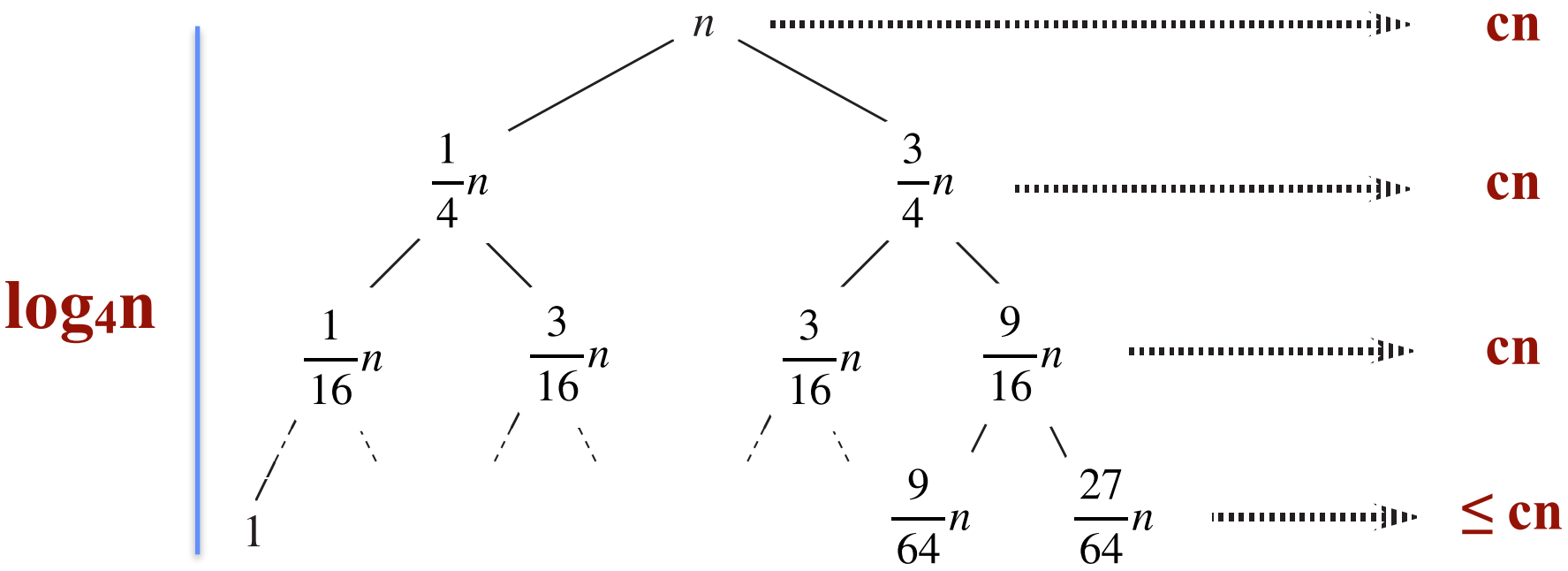
$$T(n) = T\left(\frac{n}{4}\right) + T\left(\frac{3}{4}n\right) + n$$

QuickSort - Costo



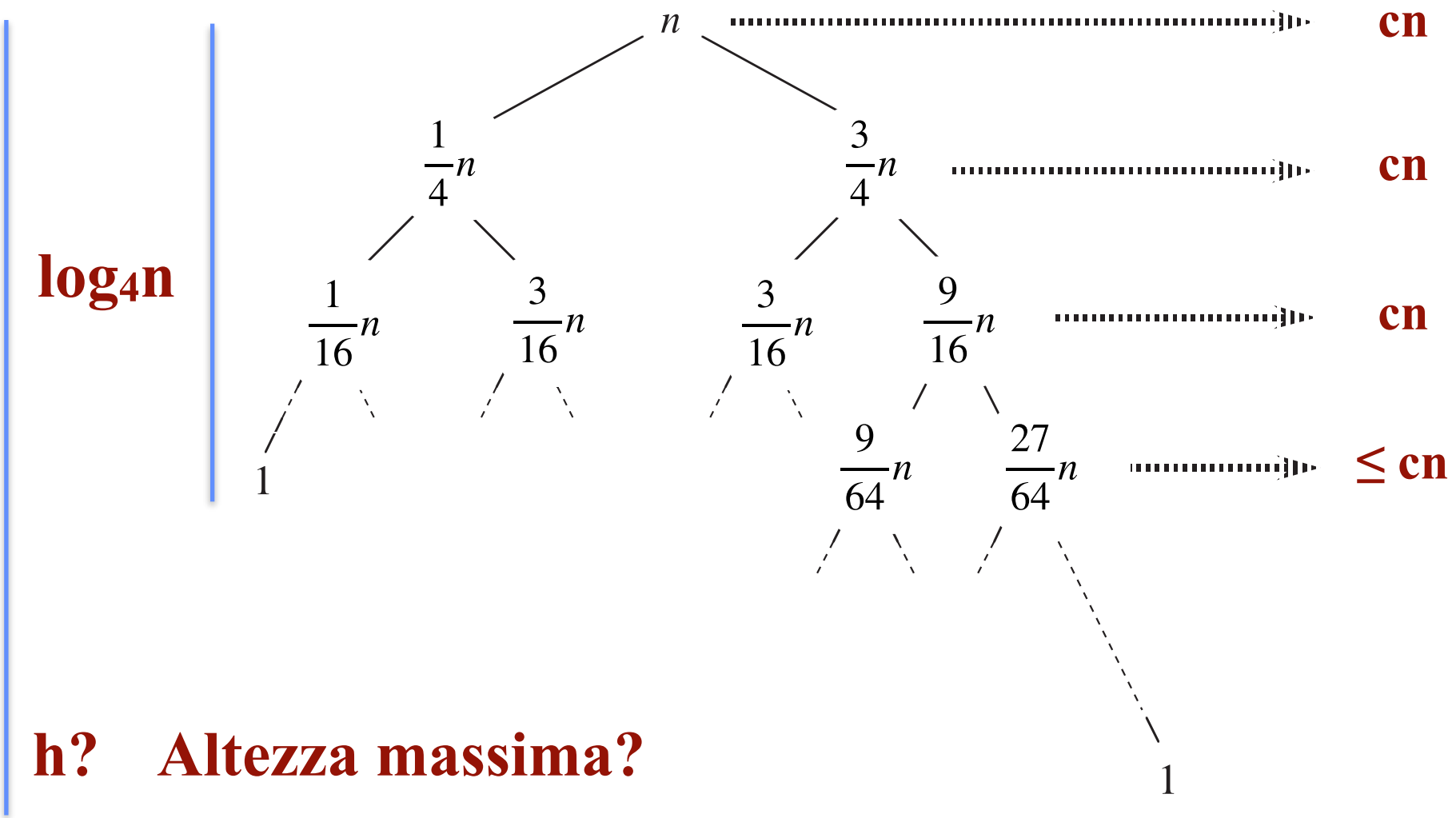
$$T(n) = T\left(\frac{n}{4}\right) + T\left(\frac{3}{4}n\right) + n$$

QuickSort - Costo



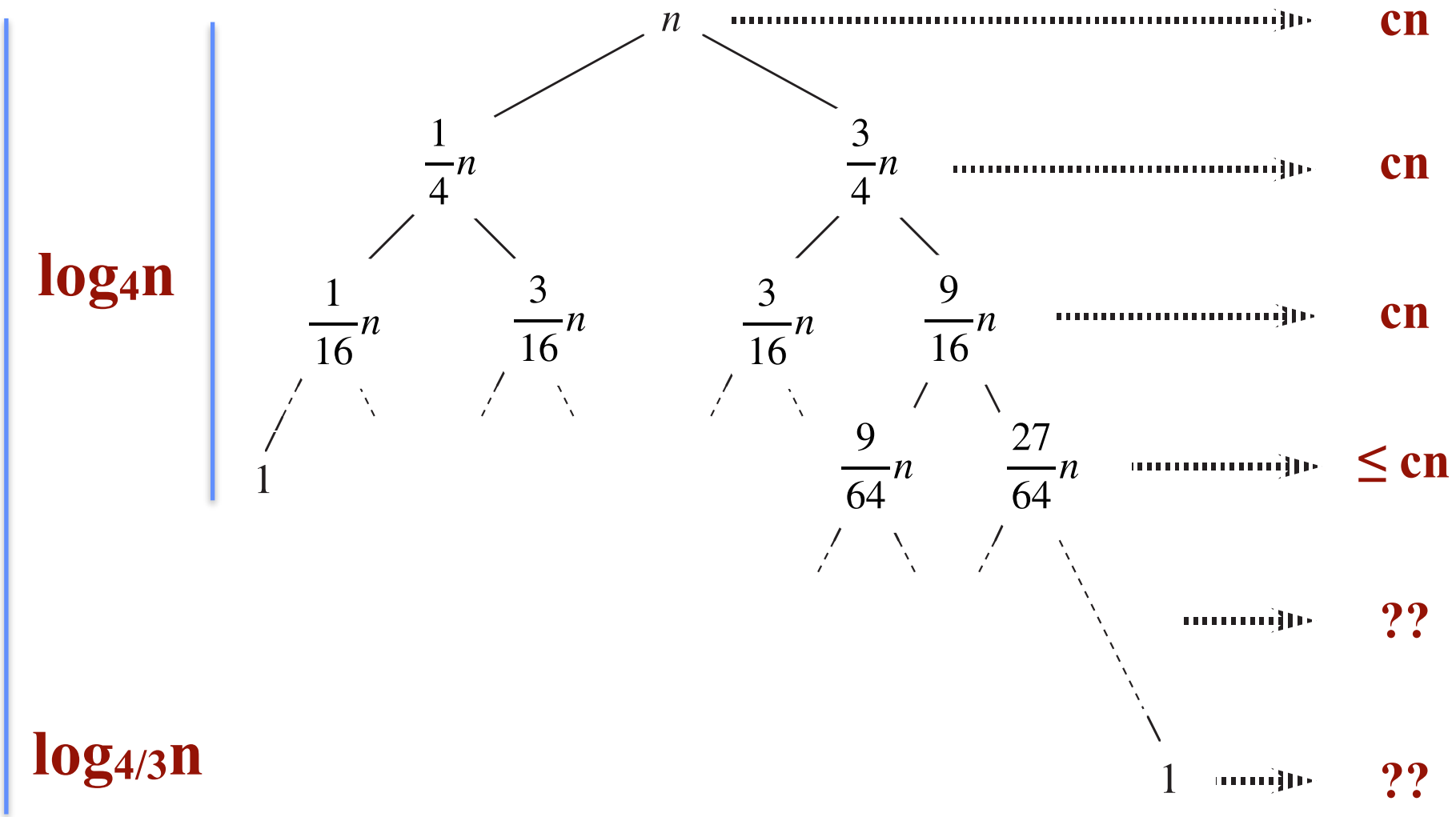
$$T(n) = T\left(\frac{n}{4}\right) + T\left(\frac{3}{4}n\right) + n$$

QuickSort - Costo



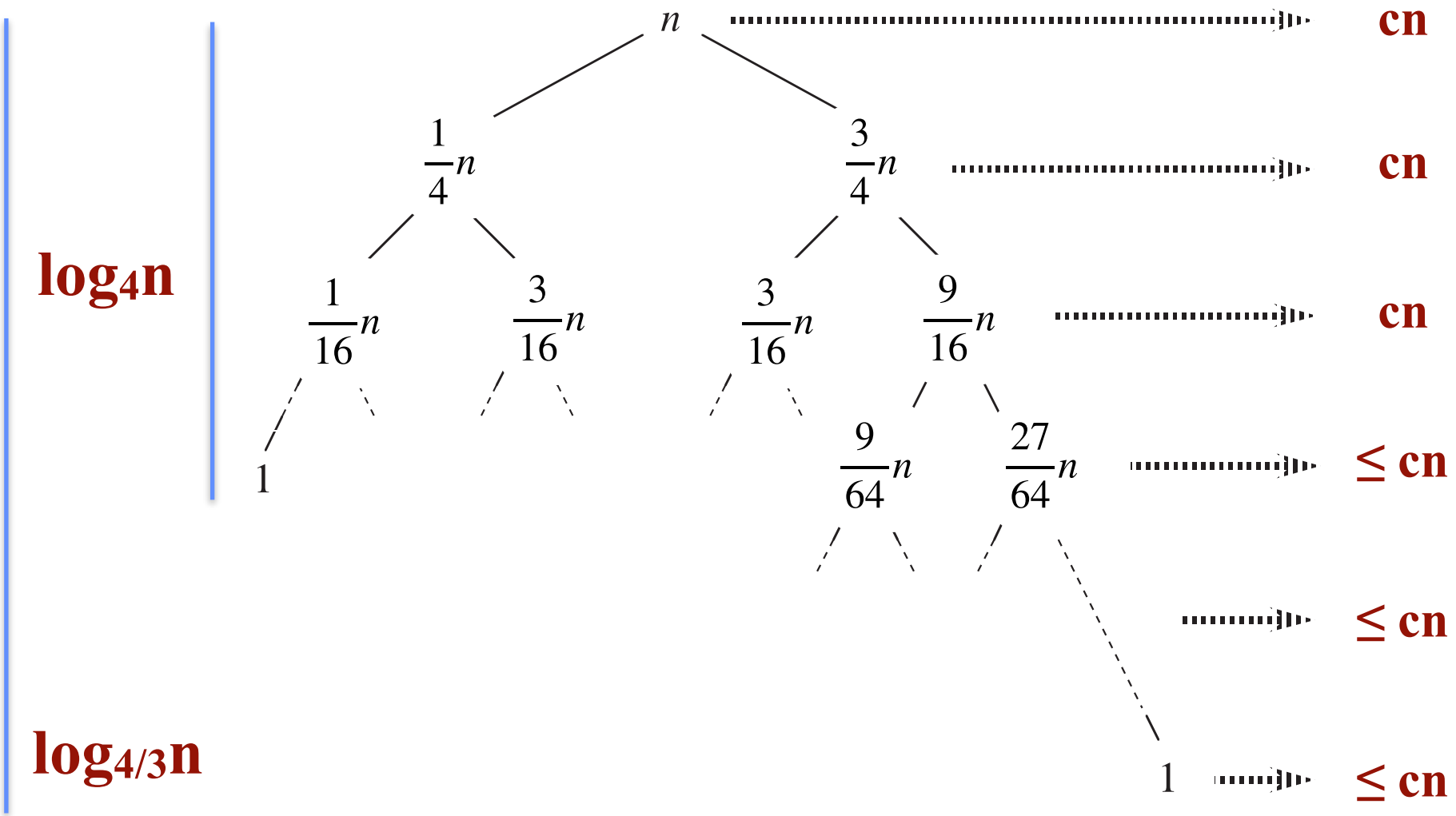
$$T(n) = T\left(\frac{n}{4}\right) + T\left(\frac{3}{4}n\right) + n$$

QuickSort - Costo



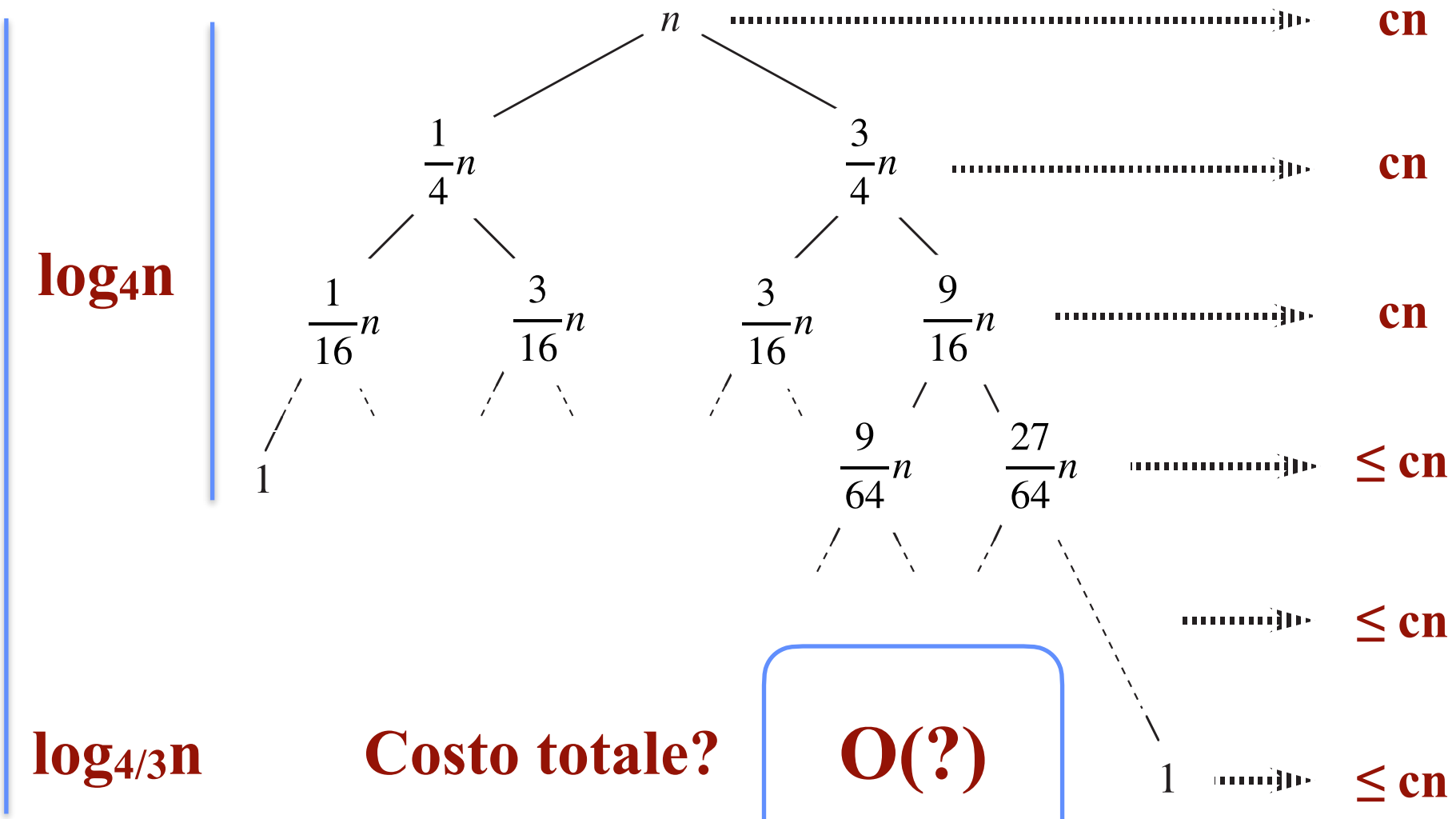
$$T(n) = T\left(\frac{n}{4}\right) + T\left(\frac{3}{4}n\right) + n$$

QuickSort - Costo



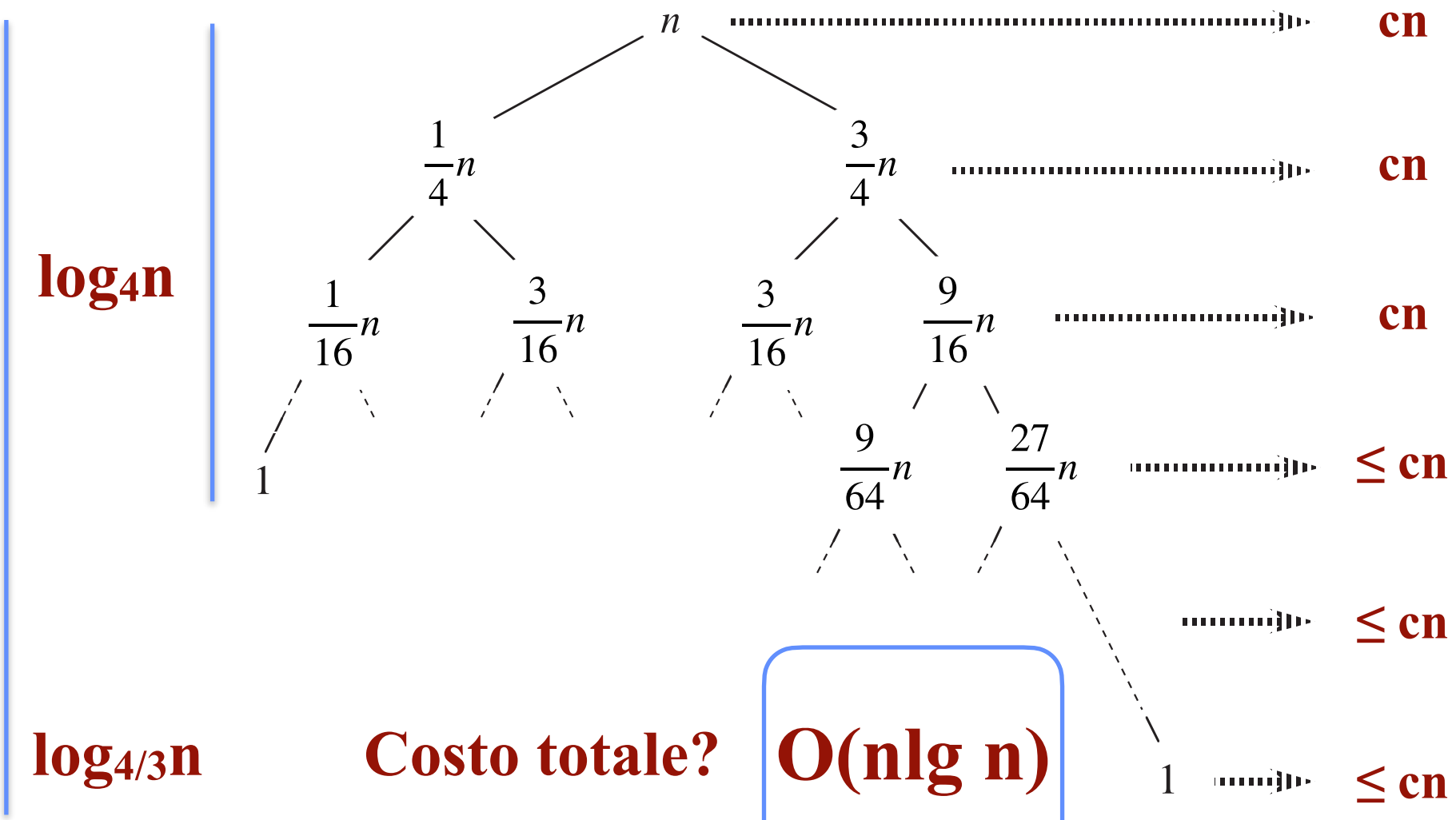
$$T(n) = T\left(\frac{n}{4}\right) + T\left(\frac{3}{4}n\right) + n$$

QuickSort - Costo



$$T(n) = T\left(\frac{n}{4}\right) + T\left(\frac{3}{4}n\right) + n$$

QuickSort - Costo



$$T(n) = T\left(\frac{n}{4}\right) + T\left(\frac{3}{4}n\right) + n$$

QuickSort Randomizzato - Analisi

In pratica funziona molto bene

Nelle librerie, usato in forma ibrida ottenuta eseguendo insertion sort o merge sort in alcune situazioni