



UNIVERSITÀ  
DEGLI STUDI  
DI TRIESTE

# MODULO 2: Quicksort

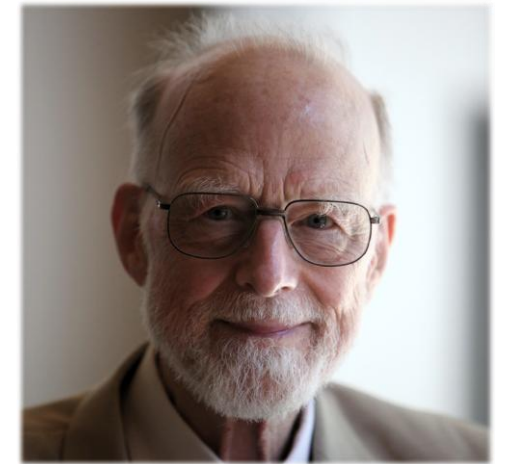
**Prof.ssa Giulia Cisotto**

[giulia.cisotto@units.it](mailto:giulia.cisotto@units.it)

Trieste, 15 maggio 2025

# ALGORITMO: IDEA

- ▶ Il quick sort è un algoritmo “**divide et impera**”
- ▶ L’idea di base è quella di scegliere in un array di  $n$  elementi un **pivot**, spostare gli elementi più piccoli prima del pivot e quelli più grandi dopo il pivot.
- ▶ Se applichiamo **ricorsivamente** lo stesso algoritmo ai due sotto-array risultanti (elementi minori e maggiori del pivot) otteniamo un array ordinato
- ▶ **La scelta del pivot è critica.** Lo si vorrebbe, idealmente, *mediano*\* in modo che ad ogni ricorsione si spezzi la sequenza a metà e si abbia il numero minimo di passi ricorsivi ( $\log_2 n$ ). Spesso si sceglie a caso!
- ▶ Il vantaggio di quicksort sta nella sua **efficienza nel caso medio**.



Ideato da Tony Hoare (vincitore del premio Turing nel 1980) nel 1959-60

\*Trovare il mediano in un array costa  $O(n)$  e bisognerebbe farlo ad ogni passo.

# ESEMPIO

Array da ordinare



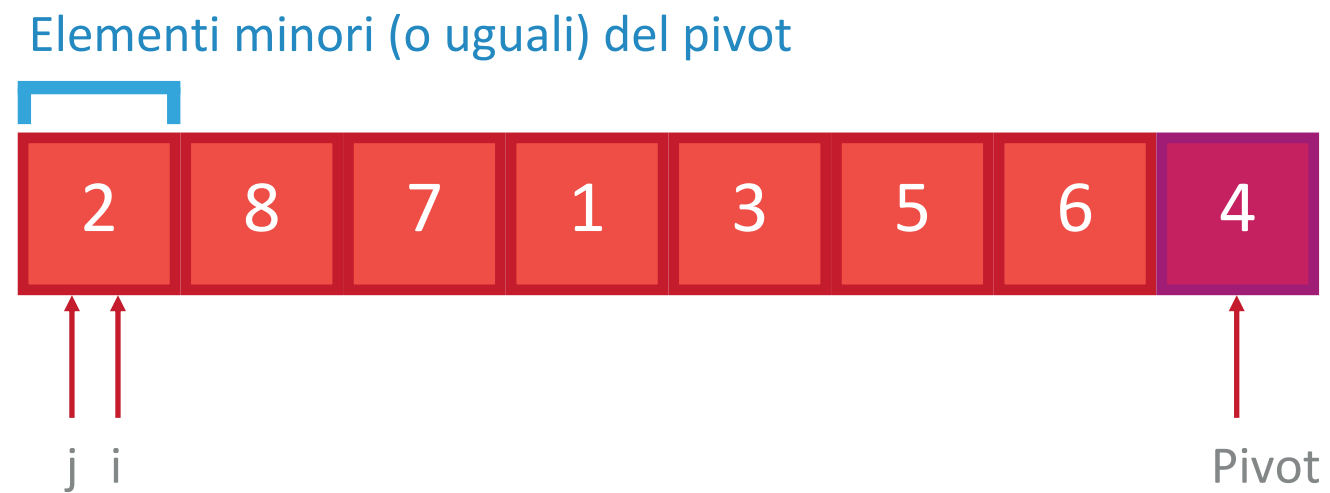
*i* indica l'ultimo degli elementi minori del pivot  
*j* viene utilizzato per scorrere l'array

## ESEMPIO



Se  $A[j]$  è minore (o uguale) del pivot  
incrementiamo  $i$  e scambiamo  $A[i]$  e  $A[j]$

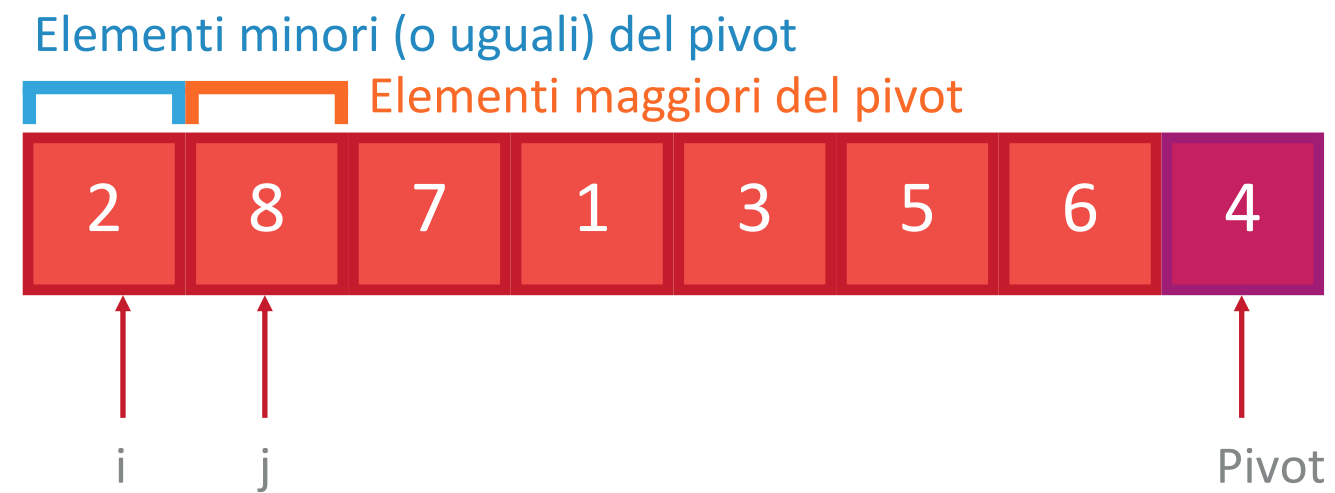
## ESEMPIO



Se  $A[j]$  è minore (o uguale) del pivot  
incrementiamo  $i$  e scambiamo  $A[i]$  e  $A[j]$

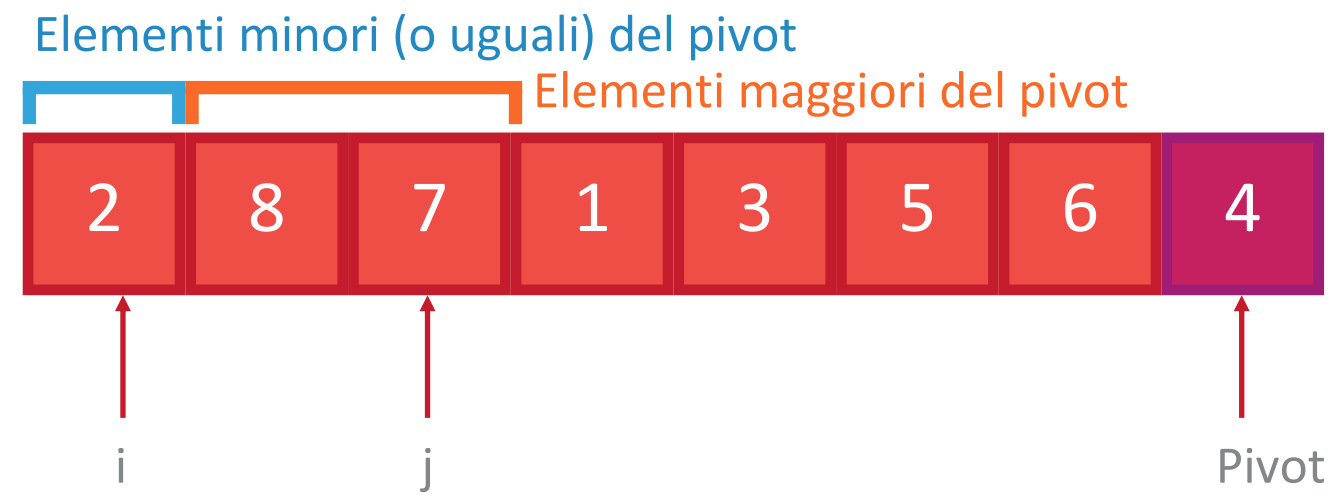
In questo caso non cambia nulla ( $i$  è uguale a  $j$ )

## ESEMPIO



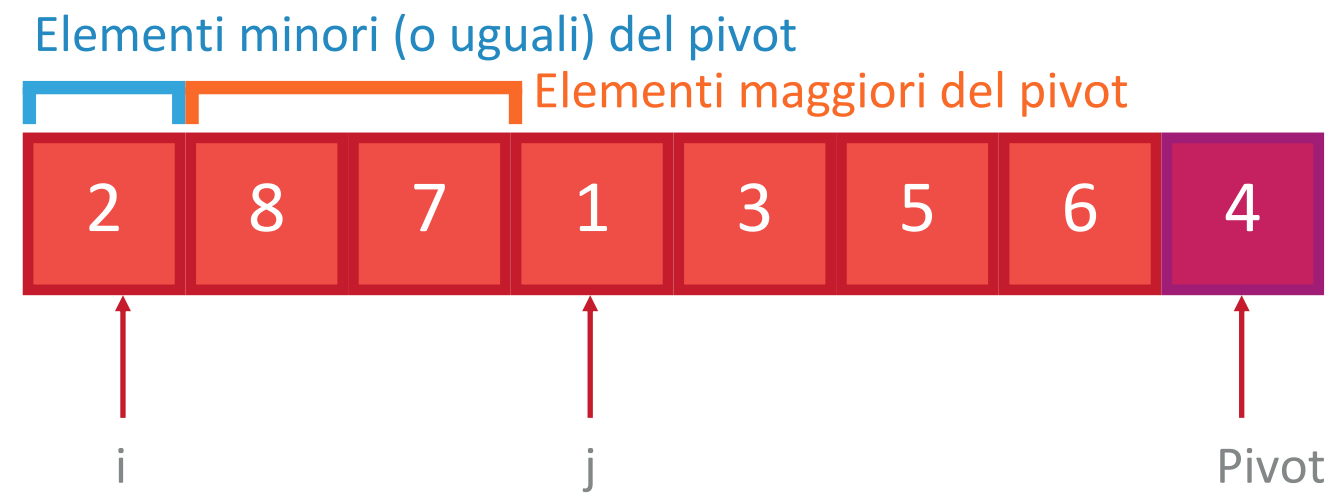
Se  $A[j]$  è maggiore del pivot  
non facciamo nulla

## ESEMPIO



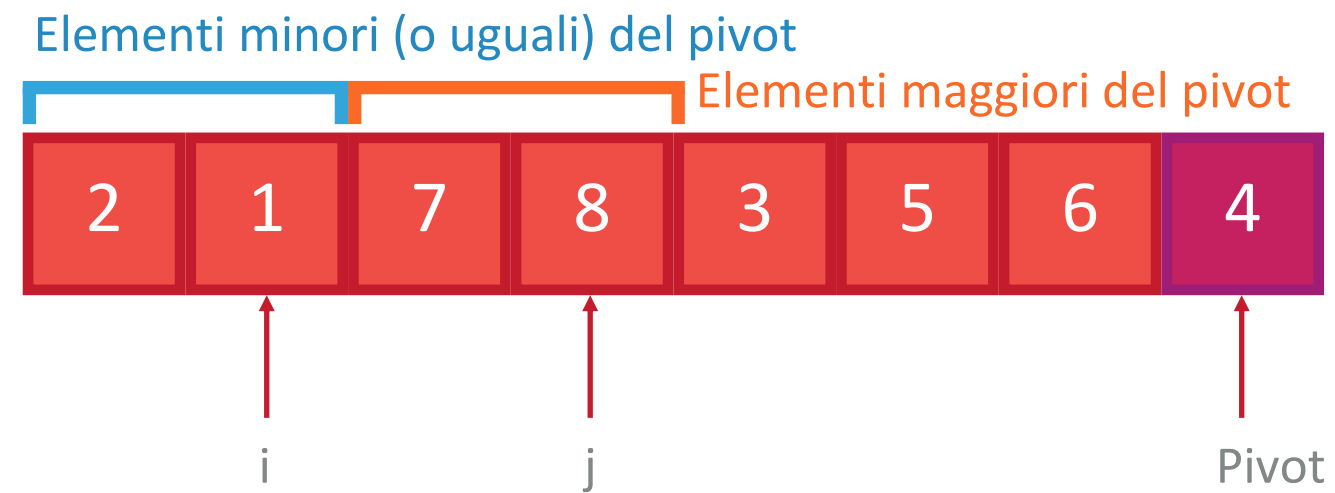
Se  $A[j]$  è maggiore del pivot  
non facciamo nulla

## ESEMPIO



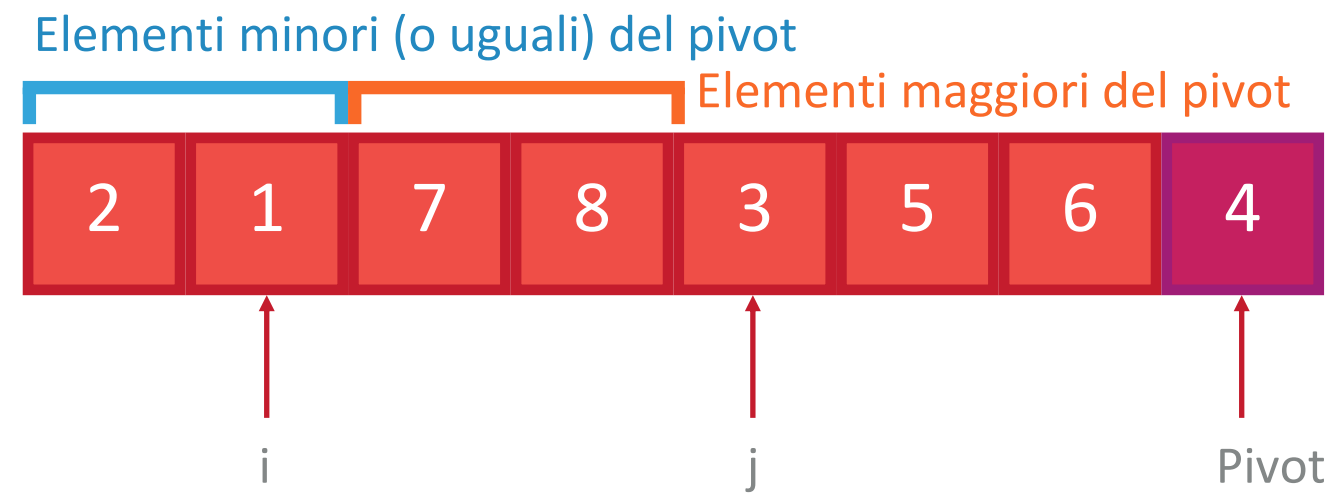
Se  $A[j]$  è minore del pivot  
incrementiamo  $i$  e scambiamo  $A[i]$  e  $A[j]$

## ESEMPIO



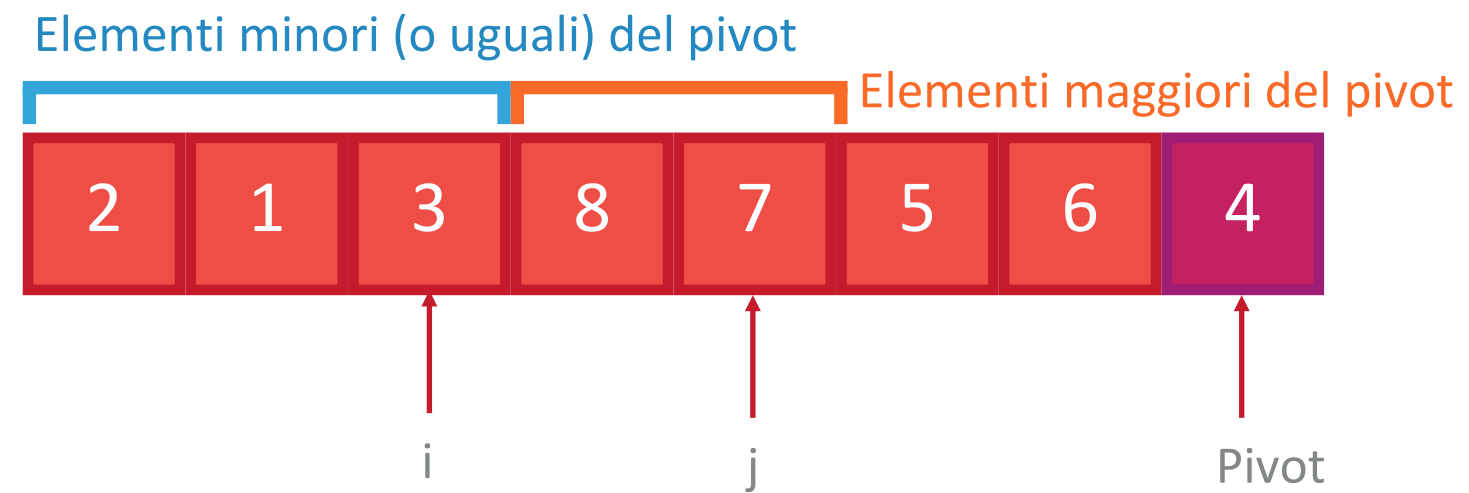
Se  $A[j]$  è minore del pivot  
incrementiamo  $i$  e scambiamo  $A[i]$  e  $A[j]$

## ESEMPIO



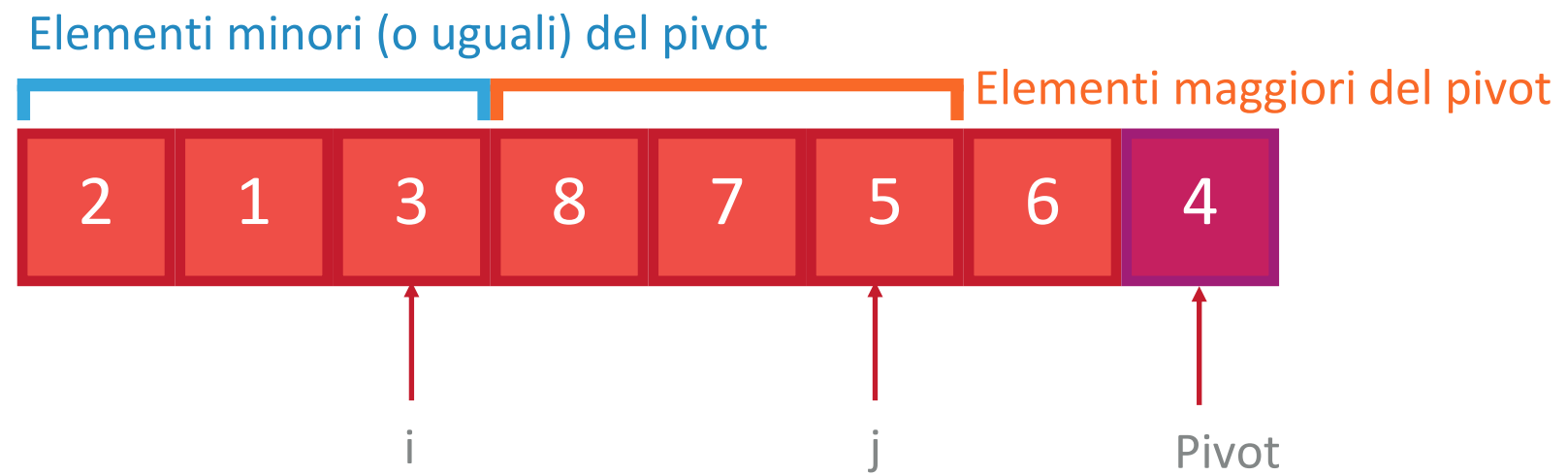
Se  $A[j]$  è minore del pivot  
incrementiamo  $i$  e scambiamo  $A[i]$  e  $A[j]$

## ESEMPIO



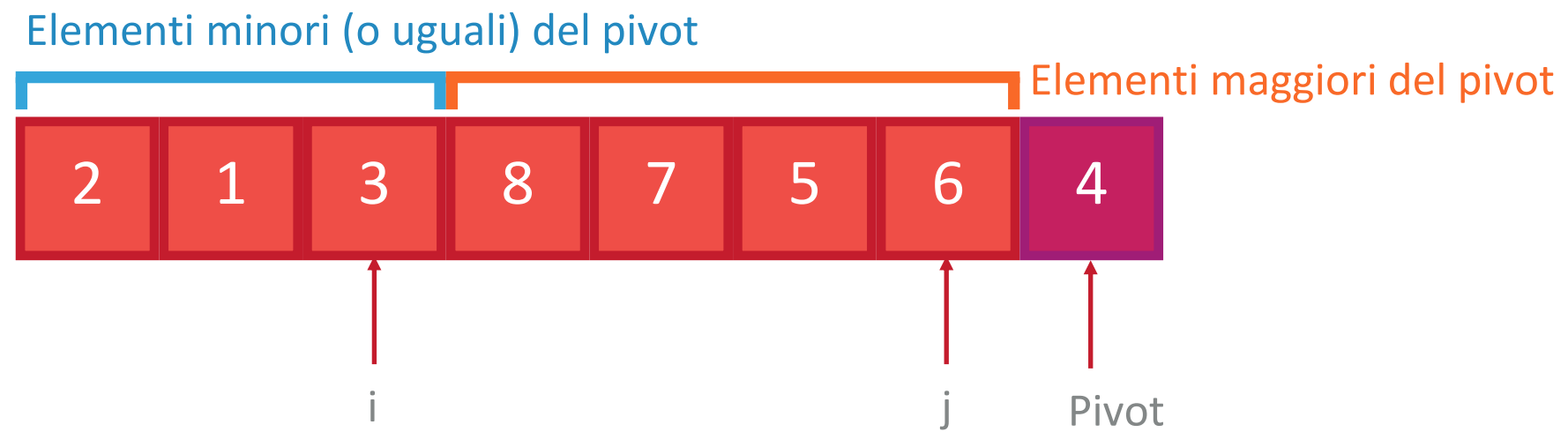
Se  $A[j]$  è minore del pivot  
incrementiamo  $i$  e scambiamo  $A[i]$  e  $A[j]$

## ESEMPIO



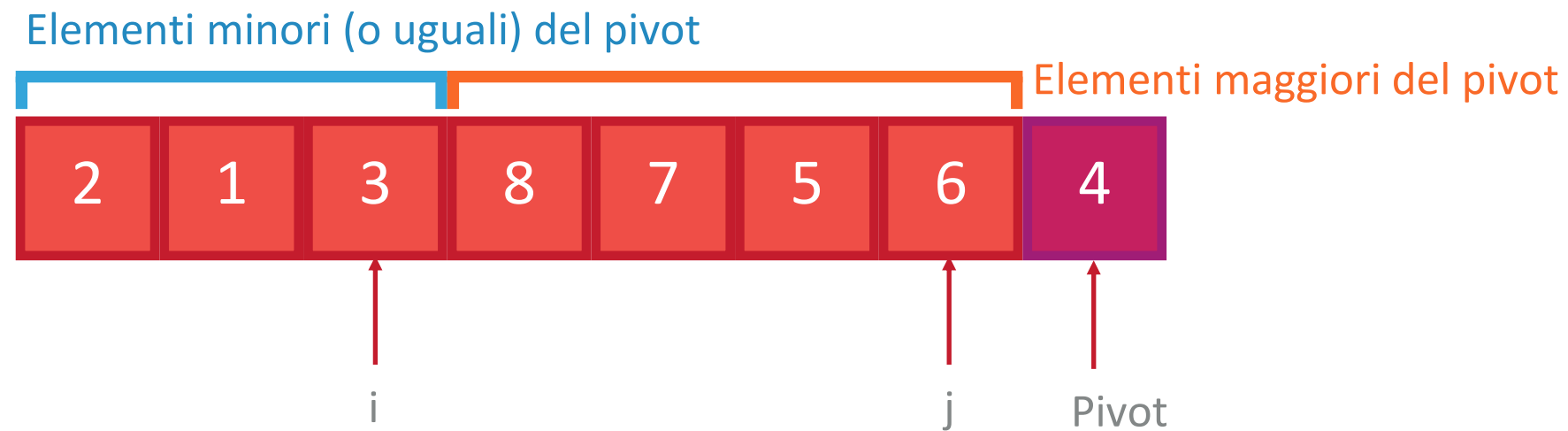
Se  $A[j]$  è maggiore del pivot  
non facciamo nulla

## ESEMPIO



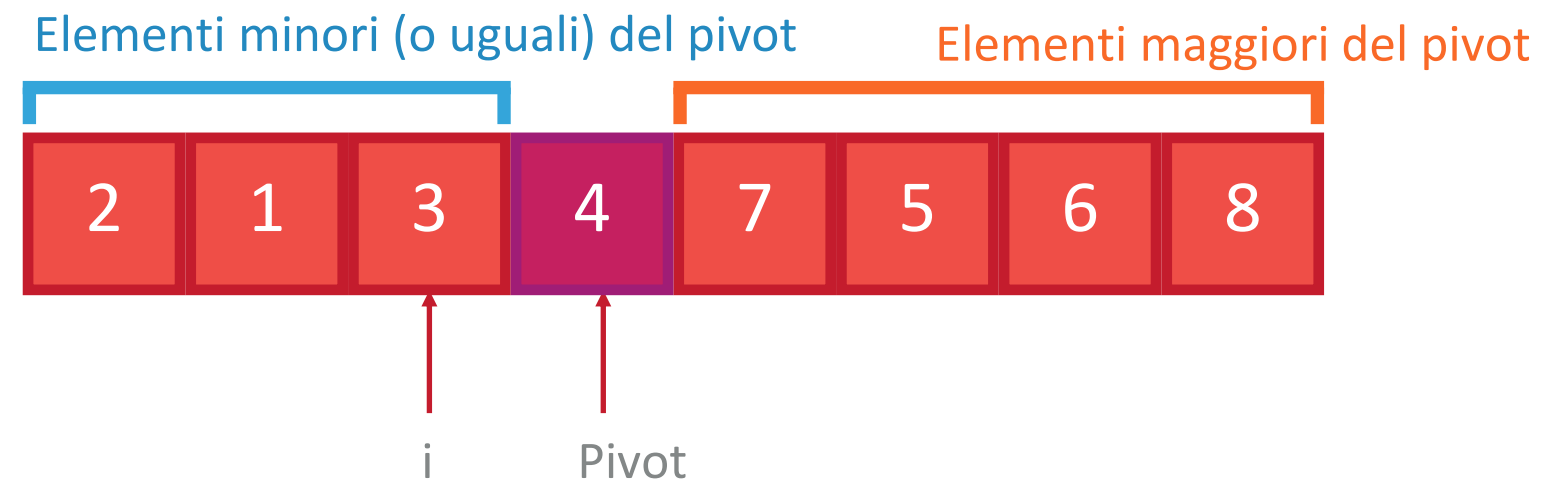
Se  $A[j]$  è maggiore del pivot  
non facciamo nulla

## ESEMPIO



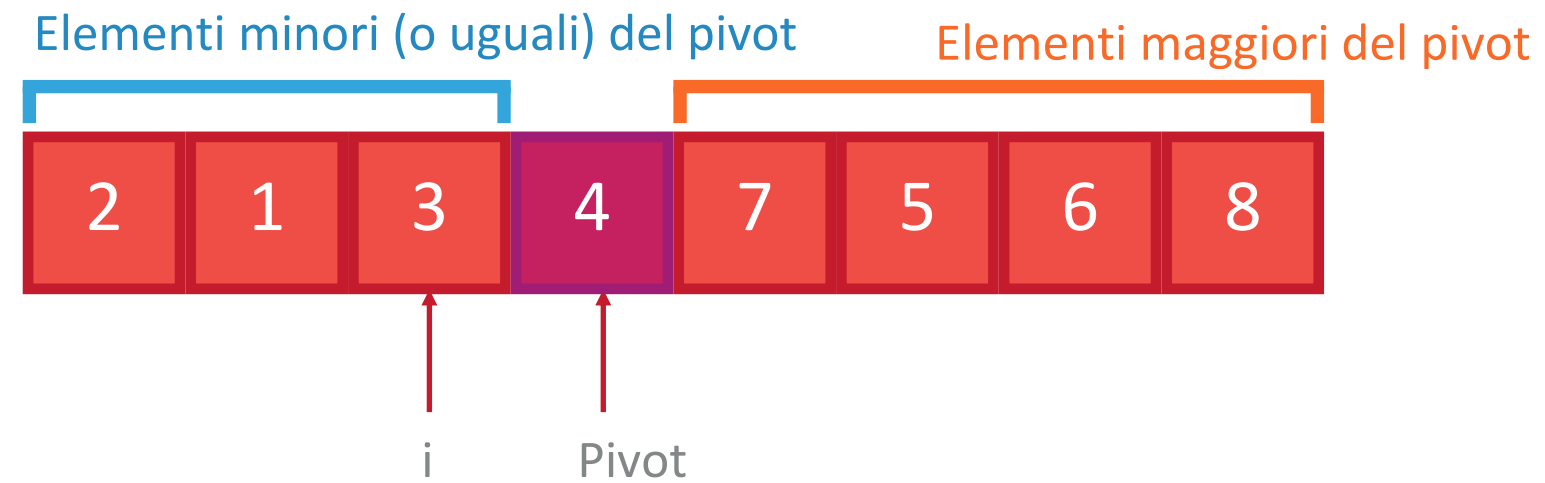
Abbiamo effettuato una scansione di tutto l'array e abbiamo raccolto tutti gli elementi minori o uguali del pivot negli indici  $[0, i]$ .  
Dobbiamo solo posizionare il pivot tra le due partizioni

## ESEMPIO



Ci basta scambiare l'elemento in posizione  $i + 1$  (che è necessariamente maggiore del pivot o il pivot stesso) con il pivot

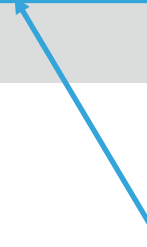
## ESEMPIO



Questa stessa procedura va poi ripetuta su tutti i sottoarray (ricorsione).

# PSEUDOCODICE (MAIN)

```
QUICKSORT (A, p, r)    //input: array A, p=inizio array, r=fine array
if p ≥ r:                //caso base: array di lunghezza 0 o 1
    return
q = partiziona(A, p, r) // indice del pivot dopo il partizionamento
quicksort(A, p, q-1) // chiamata ricorsiva sugli elementi minori o uguali
quicksort(A, q+1, r) // chiamata ricorsiva sugli elementi maggiori
```



Chiamate ricorsive

## PSEUDOCODICE (PARTIZIONAMENTO)

```
PARTIZIONA (A, p, r)           //input: array A, p=inizio array, r=fine array
x = A[r]      //il pivot è l'ultimo elemento dell'array (tra gli indici p e r)
i = p-1      //posizione iniziale degli elementi minori del pivot
for j=p to r-1 //per tutti gli elementi dell'array tranne il pivot
    if A[j] ≤ x      //se troviamo un elemento minore del pivot...
        i = i+1
        scambia A[i] e A[j] //...lo spostiamo nella prima parte dell'array
scambia A[i+1] con A[r] //mettiamo il pivot nella sua posizione finale
return i+1
```

# CORRETTEZZA DELL'ALGORITMO

**Invariante** (condizione che rimane vera ad ogni ciclo):

- ▶ Gli elementi tra l'inizio dell'array e  $i$  sono tutti minori o uguali del pivot
- ▶ Gli elementi tra  $i + 1$  e  $j$  sono tutti maggiori del pivot

**Ogni iterazione continua a far rispettare questa condizione:**

- ▶ Se il nuovo elemento è maggiore del pivot viene mantenuto nella sua posizione, rispettando quindi la condizione
- ▶ Se il nuovo elemento è minore o uguale del pivot,  $i$  viene incrementato, l'elemento in posizione  $j$  viene scambiato con quello in posizione  $i$  (che, dato che  $i$  è stato incrementato, è maggiore del pivot)

## PRESTAZIONI: CALCOLO NEL CASO MIGLIORE

Costo partizionamento:  $\Theta(n)$  (ogni elemento è visitato 1 volta e, se necessario, spostato)

**Pivot: mediano**

Costo totale:

$$T(n) = 2T\left(\frac{n}{2}\right) + \Theta(n) = (\text{teorema dell'esperto}) = \Theta(n^{\log_2 2} \log n) = \Theta(n \cdot \log n)$$

Costo chiamata ricorsiva  
(su 2 sottoarray  
contenente  $n/2$  elementi)

Costo partizionamento

## PRESTAZIONI: CALCOLO NEL CASO PEGGIORE

Costo partizionamento:  $\Theta(n)$  (ogni elemento è visitato 1 volta e, se necessario, spostato)

**Pivot: primo (o ultimo) elemento dell'array**

Costo totale:

$$T(n) = \boxed{T(n-1) + T(1)} + \boxed{\Theta(n)} = T(n-1) + \Theta(n)$$

Risolvo iterativamente  $\longrightarrow$

$$\begin{aligned} T(n-1) &= T(n-2) + \Theta(n-1) \\ T(n-2) &= T(n-3) + \Theta(n-2) \\ &\vdots \\ T(2) &= T(1) + \Theta(1) \end{aligned}$$

Costo chiamata ricorsiva (su 1 sottoarray contenente 1 elemento e l'altro n-1 elementi)      Costo partizionamento

$$= \Theta(1) + \Theta(2) + \dots + \Theta(n) = \sum_{i_1}^n \Theta(i) = \Theta\left(\frac{n(n+1)}{2}\right) = \Theta(n^2)$$

## PRESTAZIONI: CALCOLO NEL CASO MEDIO

Costo partizionamento:  $\Theta(n)$  (ogni elemento è visitato 1 volta e, se necessario, spostato)

**Pivot: elemento a caso**  **d** = numero di livelli di partizionamento

Ad esempio: se partizioniamo  $\frac{1}{4} \cdot n$  e  $\frac{3}{4} \cdot n$

Il ramo più lungo delle chiamate ricorsive sarà quello che continua a prendere la parte più lunga della partizione. Ad un certo punto si arriverà ad un array di lunghezza 1 (dopo d chiamate ricorsive). A quale d corrisponde questo momento?

$$\left(\frac{3}{4}\right)^d n = 1$$



$$d = \frac{\log_2 n}{\log_2(3/4)} = 2.5 \log_2 n$$

Costo totale:  $T(n) = d \cdot \Theta(n) = O(\log n) \cdot \Theta(n) = O(n \log n)$

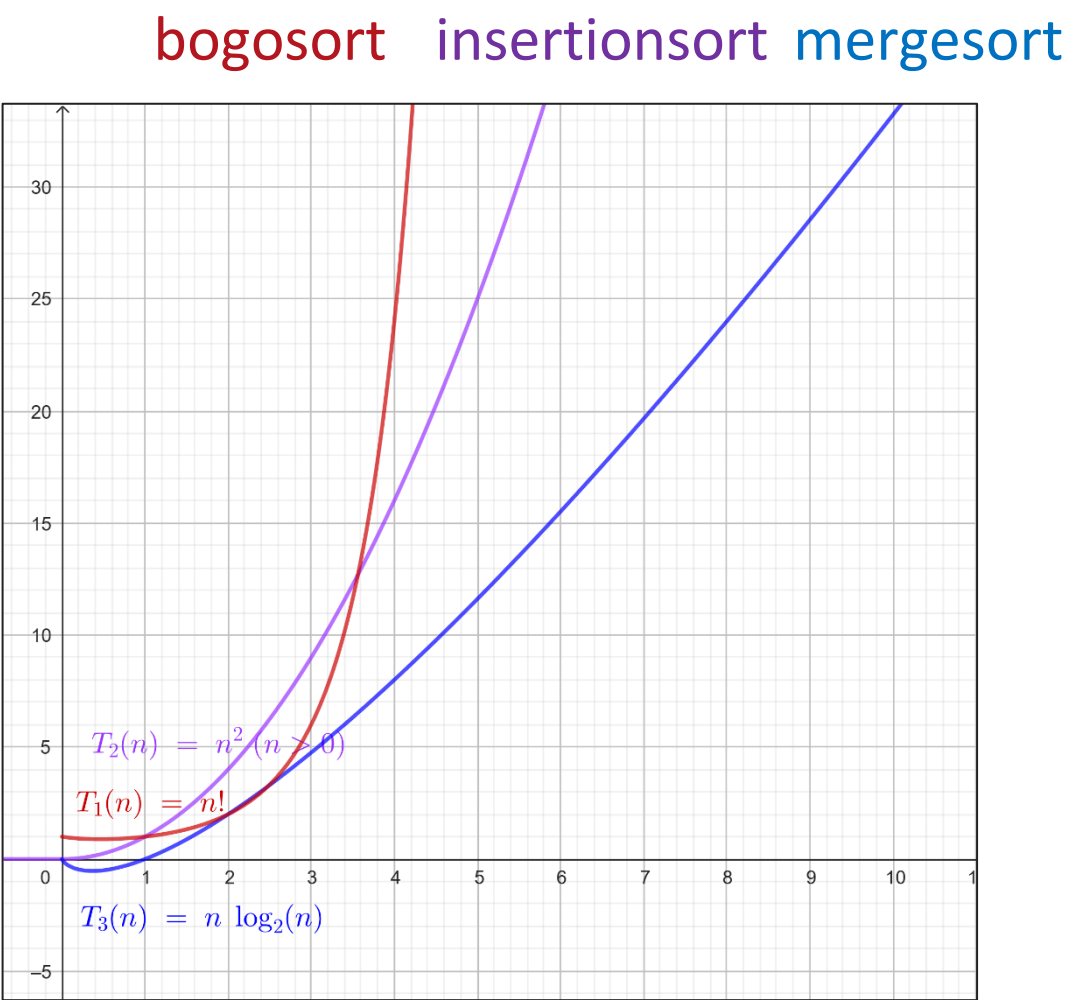
# CONFRONTO CON ALTRI ALGORITMI DI ORDINAMENTO

Caso migliore (array già ordinato):  $O(n \log n)$

Caso peggiore:  $O(n^2)$

Caso medio:  $O(n \log n)$

| Algoritmo      | $T(n)$         |
|----------------|----------------|
| Bogosort       | $n!$           |
| Insertion sort | $n * n = n^2$  |
| Mergesort      | $n * \log_2 n$ |
| Quicksort      |                |



Potete usare [Geogebra online](#) per creare lo stesso grafico e poi aggiungere l'andamento dell'algoritmo assegnato.



UNIVERSITÀ  
DEGLI STUDI  
DI TRIESTE

*Prossimo laboratorio: 16 maggio, h.11:00, aula 5B*