In questa lezione

- **La funzione FONDI: fonde due arrays ordinati in un unico array ordinato**

- **La funzione PARTIZIONE: prende in input un array A non ordinato di interi e individua la posizione che dovrebbe assumere un elemento scelto, detto pivot, di A, se fosse ordinato. Gli elementi minori o uguali del pivot sono spostati in modo da precedere il pivot stesso e tutti quelli maggiori sono spostati in modo da seguirlo.**

Fondi

Fondi(L,R,A)

Input: L,R ed A sono array di m, n e n+m elementi

prec: L e R sono ordinati, cioè

L[1] ≤ ...≤ L[m-1] e R[1]≤…≤R[n-1]

postc: A contiene gli elementi di L e di R e

A[0] ≤ ...≤ A[m+n-1]

Fondi

Fondi(L,R,A)

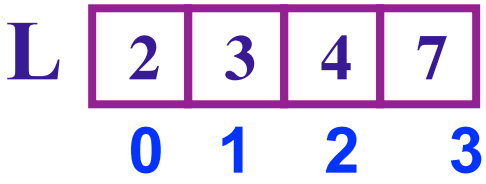
Input: L,R ed A sono array di m, n e n+m elementi

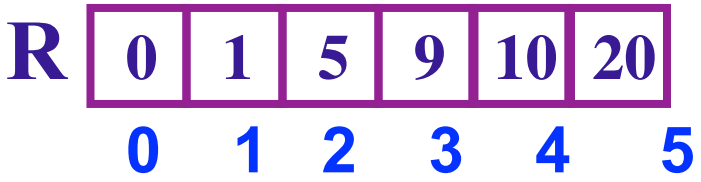
prec: L e R sono ordinati, cioè

L[1] ≤ ...≤ L[m-1] e R[1]≤…≤R[n-1]

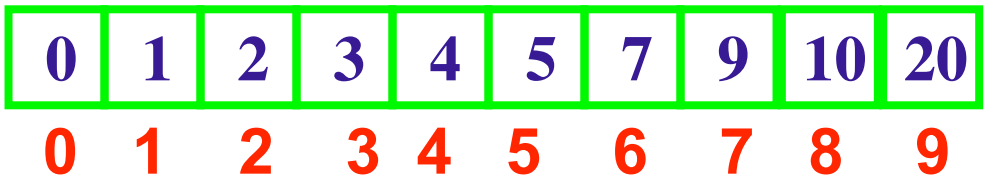
postc: A contiene gli elementi di L e di R e

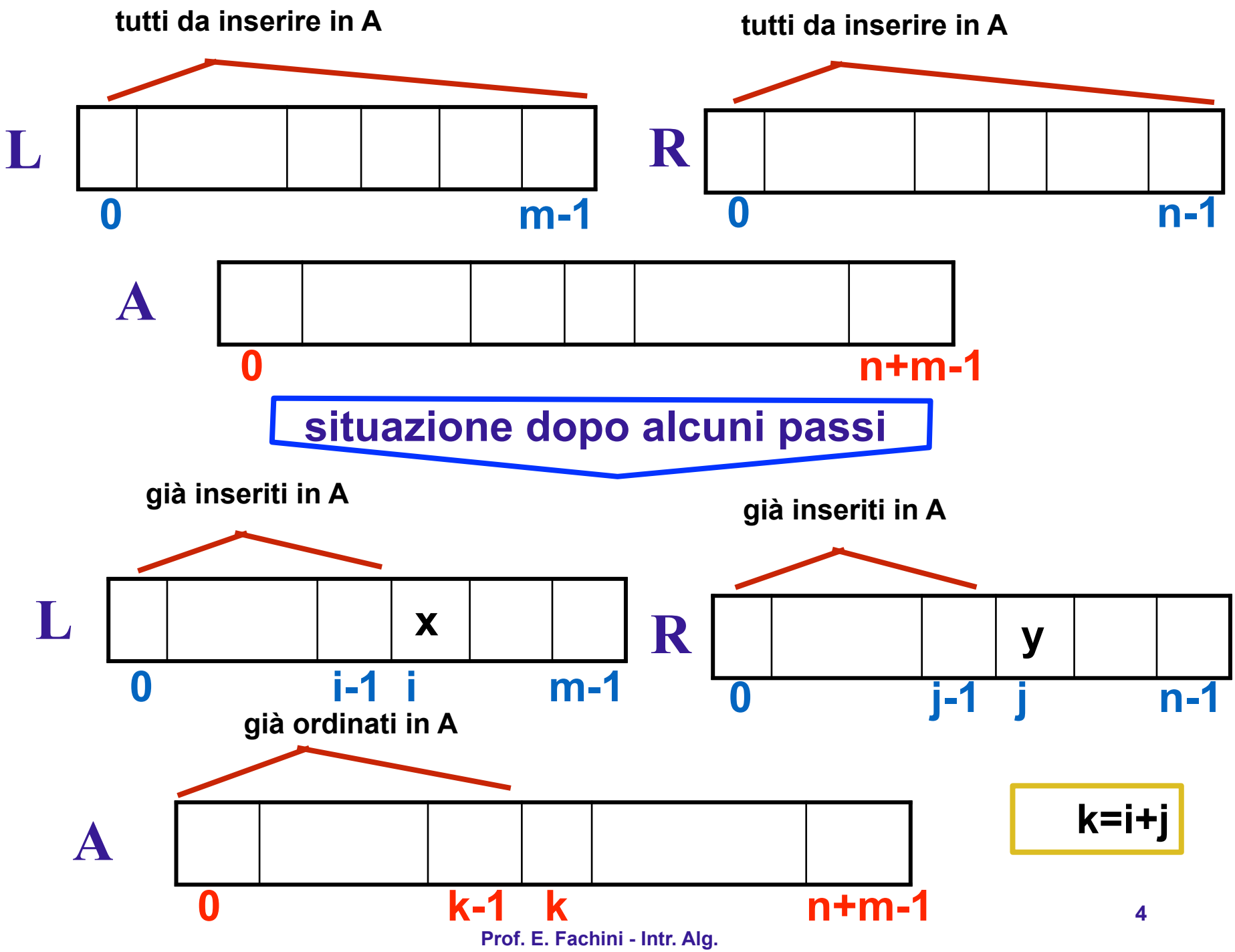
A[0] ≤ ...≤ A[m+n-1]



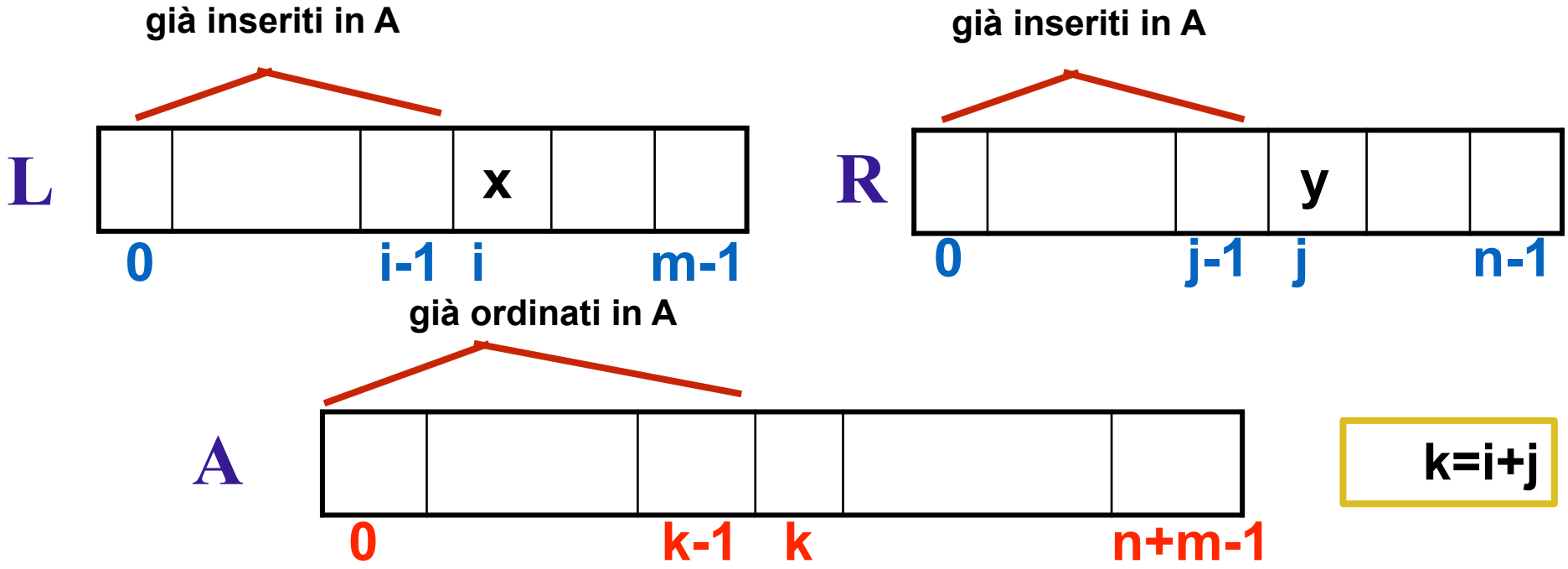


obiettivo:




tutti da inserire in A
L
0
m-1
tutti da inserire in A
R
0
n-1
A
0
n+m-1
situazione dopo alcuni passi
già inseriti in A
L
x
0
i-1
i
m-1
già inseriti in A
R
y
0
j-1
j
n-1
già ordinati in A
A
0
k-1
k
n+m-1
k=i+j

Passo ulteriore



Osserviamo che gli elementi già inseriti sono più piccoli sia di x = L[i] che di y = R[j] e sono ordinati in A. Se inseriamo il più piccolo tra x e y, otteniamo un array ordinato in A?

Se l'elemento inserito è il minimo tra i due è anche il minimo tra tutti quelli ancora da inserire, inoltre è maggiore o uguale di tutti quelli già inseriti, quindi è inserito nel posto giusto per estendere l'ordinamento ai k +1 elementi in A.

Se uno dei due array si svuota gli elementi rimanenti dell'altro possono essere copiati in A in coda a quelli già inseriti, perché sono maggiori di questi e già ordinati.

Esempio di esecuzione

Lo pseudocodice

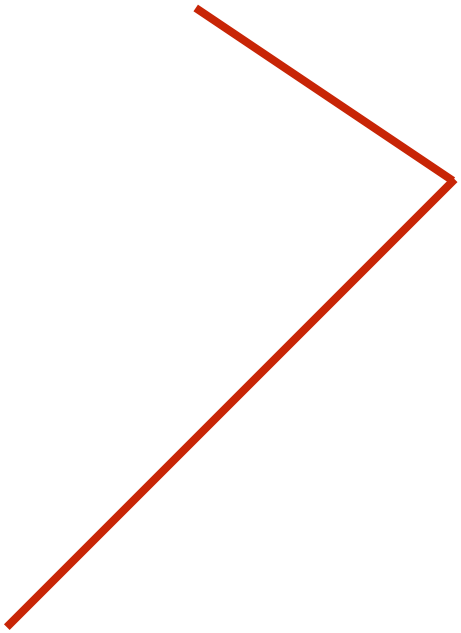
```
Fondi(L,R,A)
Input: L,R ed A  sono array di m, n e n+m elementi
prec: L e R sono ordinati, cioè
L[1] ≤ ...≤ L[m-1] e R[1]≤…≤R[n-1]
postc: A contiene gli elementi di L e di R e A[0] ≤ ...≤ A[m+n-1]
m =  L.size
n = R.size
i=j=k=0
while  i < m and  j < n   do
In ogni passo di esecuzione del ciclo while A contiene gli elementi
L[0],...,L[i-1] e gli elementi R[0],...,R[j-1], in modo che A[0] ≤ A[1] ≤...≤
A[k-1],con k=i+j
          if (L[i] ≤ R[j]) then A[k] = L[i]; i =i +1
               else A[k] = R[j]; j = j+1
          k = k+1
 while  i<m  do % uscita per j=n, copia eventuali elementi rimanenti di L,
       A[k] = L[i]; k=k+1; i=i+1
 while  j<n  do  % uscita per i=m, copia eventuali elementi rimanenti di R,
      A[k] = R[j]; k=k+1; j=j+1
```

Complessità

Fondi(L,R,A)

Input: L,R ed A sono array di m, n rispettivamente n+m elementi
prec: L e R sono ordinati, cioè
L[1] ≤ ...≤ L[m-1] e R[1]≤…≤R[n-1]
postc: A contiene gli elementi di L e di R e A[0] ≤ ...≤ A[m+n-1]

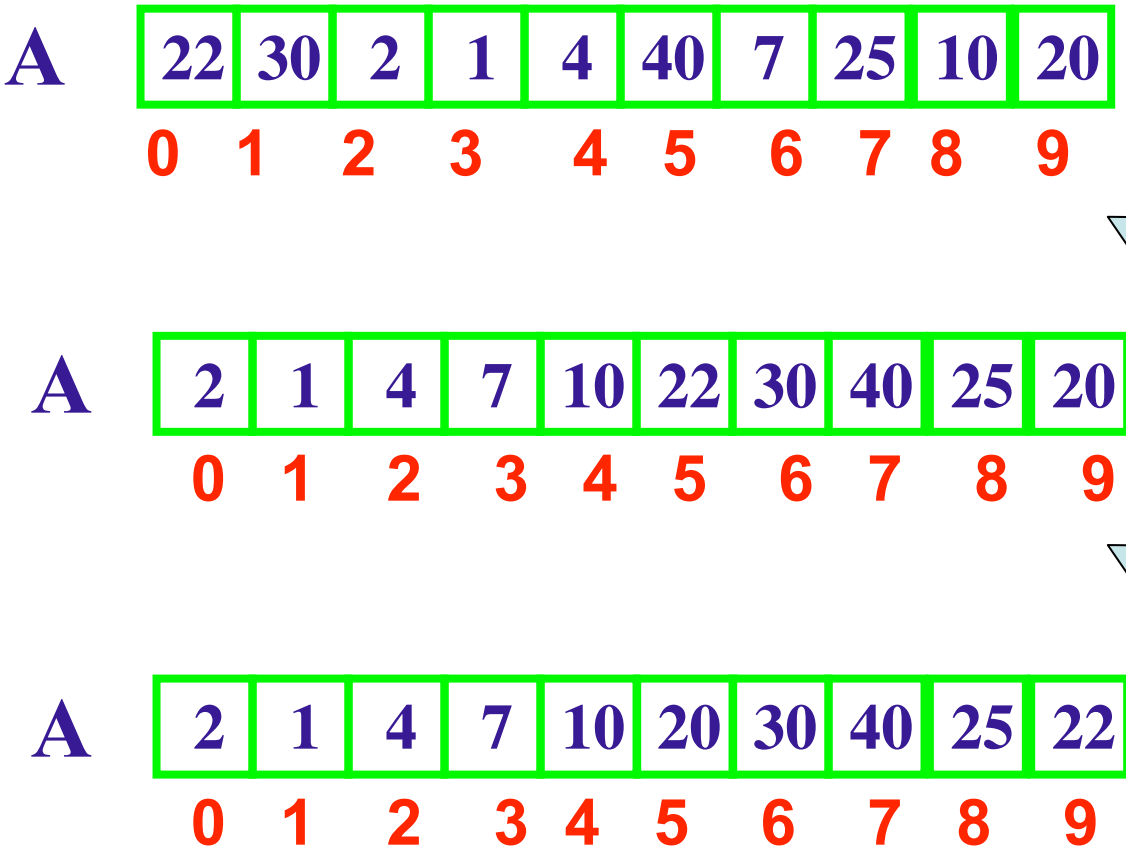
```
m =  L.size
n = R.size
i=j=k=0
while  i<m and  j<n   do
       if L[i] ≤ R[j]
              then A[k] = L[i]
                   i=i+1
             else A[k] = R[j]
                  j=j+1
         k=k+1
while  i<m  do
      A[k] = L[i]
       k=k+1
       i=i+1
while  j<n  do
      A[k] = R[j]
       k=k+1
       j=j+1
```



Θ(n+m)

Partizione

Bisogna individuare la posizione giusta del pivot e spostare gli elementi minori o uguali a sinistra di questa posizione, inserire il pivot e spostare quelli maggiori alla sua destra pivot. Nell'esempio prendiamo come pivot l'ultimo elemento, 20. Vogliamo ottenere il risultato senza usare memoria aggiuntiva e in tempo lineare nel numero degli elementi.

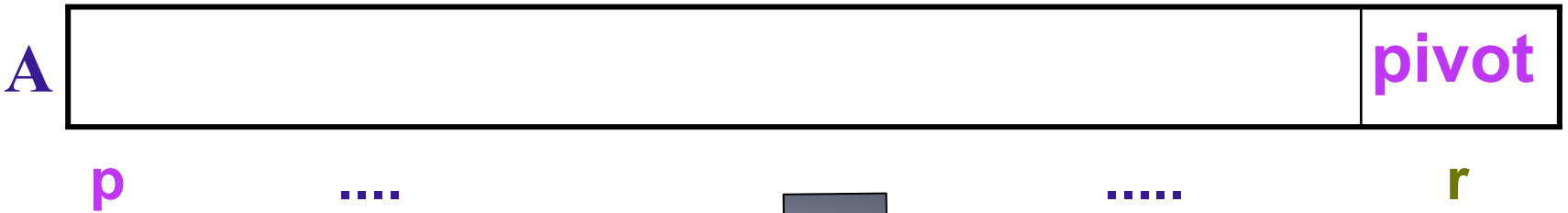


gli elementi minori o uguali a 20 vanno a sinistra, a seguire i maggiori

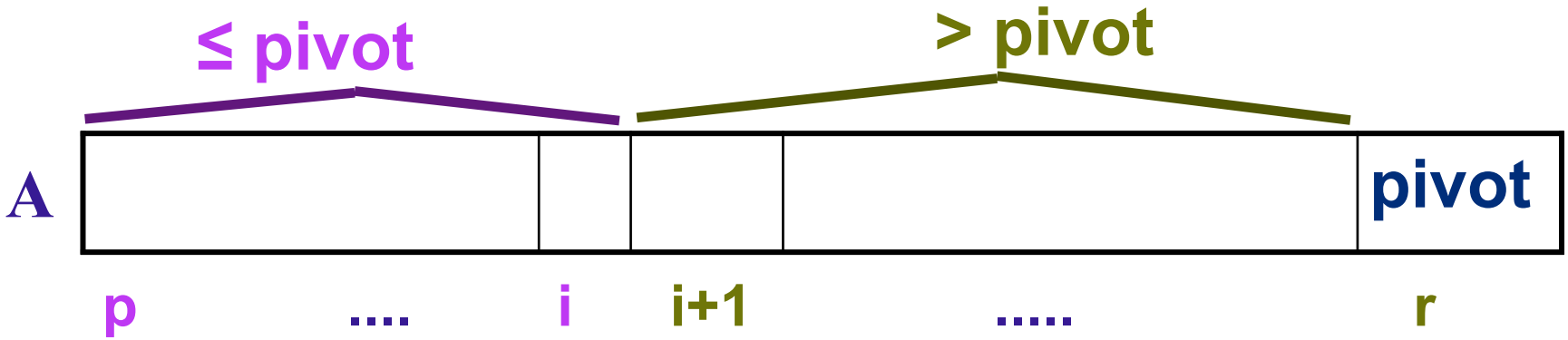
scambio del pivot con il primo dei più grandi

Partizione di un array

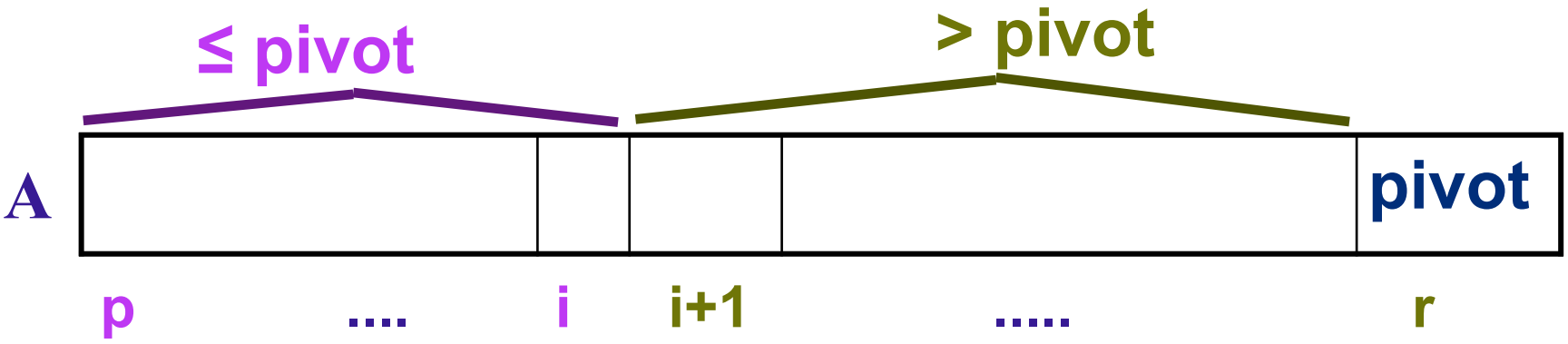
situazione inziale:



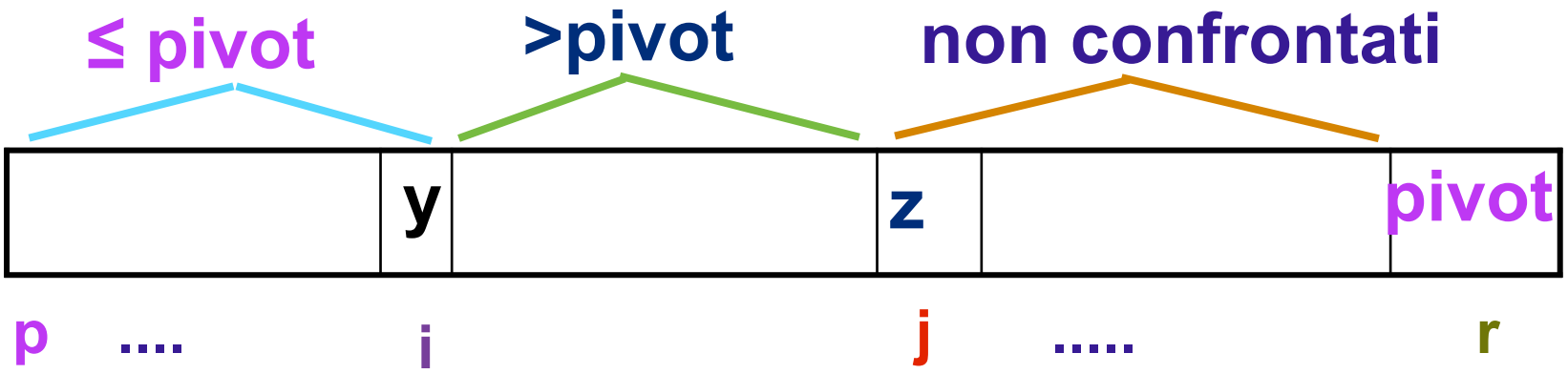
obiettivo:



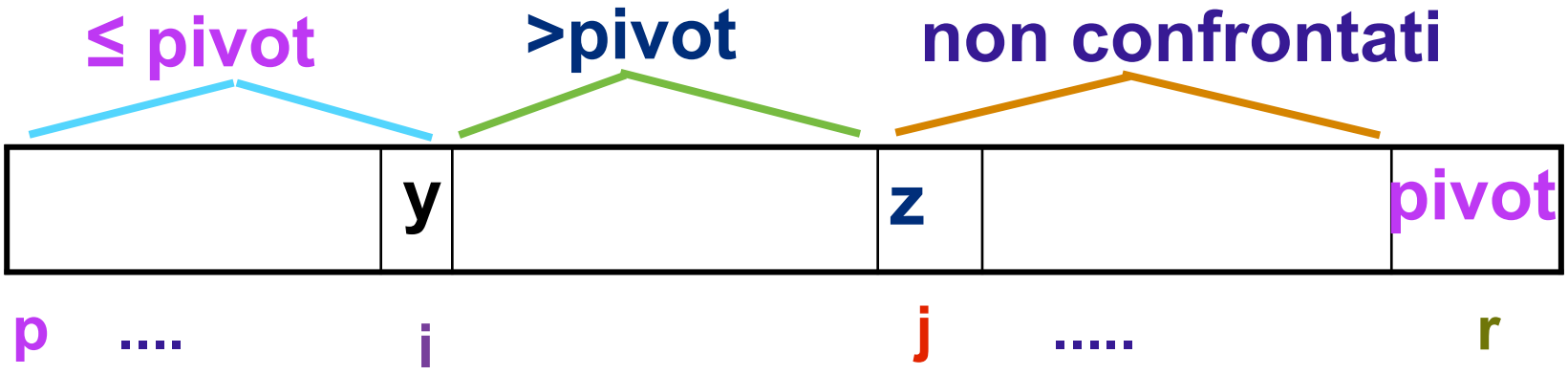
obiettivo:



possibile stadio intermedio:



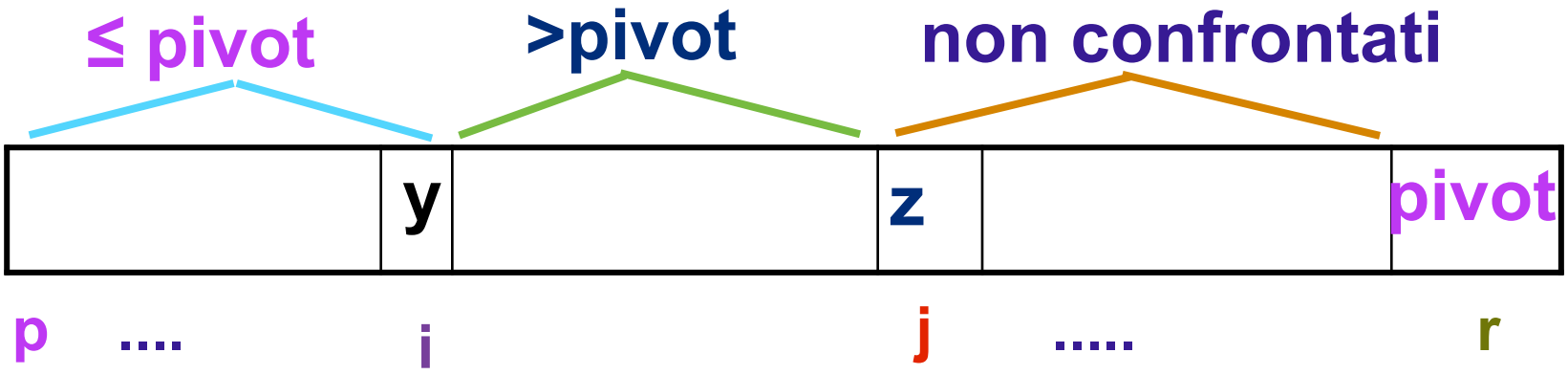
Che operazione su A[j]?



L'**indice j è quello di scorrimento dell'array. Supponiamo che tra gli elementi già confrontati con il pivot i più piccoli siano nella parte sinistra dell'array, dall'indice iniziale a un indice i e che gli elementi a seguire, dall'indice i+1 fino a j-1 siano più grandi del pivot.**

Ora ci chiediamo: cosa deve essere vero quando si considera l'elemento A[j]?

Che operazione su A[j]?



Se A[j] > pivot basta incrementare j e controllare così l'elemento successivo.

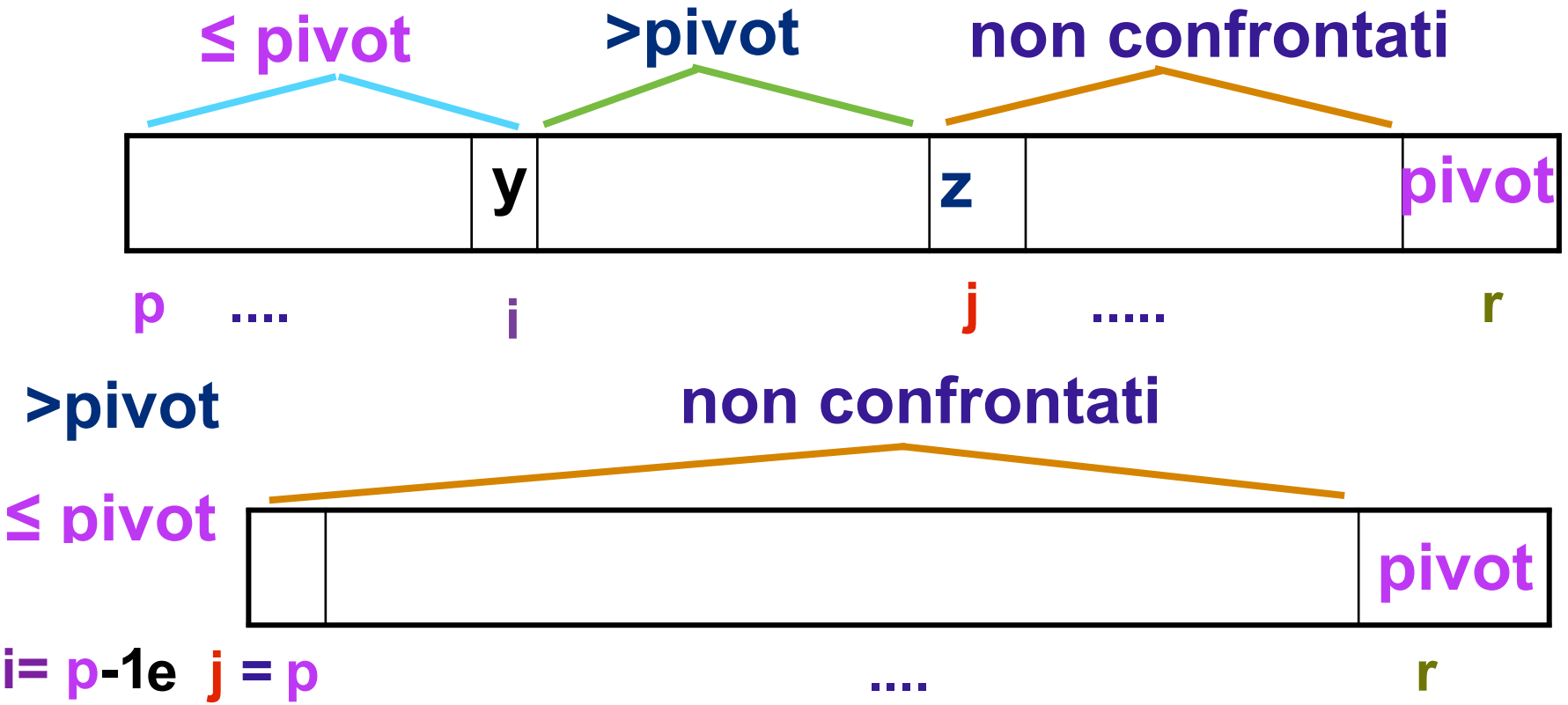
Se A[j] ≤ pivot , allora A[j] deve andare nella "zona" dei più piccoli. Per ottenere questo basta scambiarlo con A[i+1], che è maggiore del pivot, e incrementare sia i che j.

in ogni passo di esecuzione:

Gli elementi di indice tra p e i sono ≤ pivot

gli elementi di indice tra i+1 e j-1 sono > pivot

Inizializzazione indici



Prima di ogni confronto l'insieme degli elementi ≤ pivot deve essere vuoto quindi poniamo i = p - 1, anche l'insieme degli elementi > pivot deve essere vuoto, e quindi poniamo j = p.

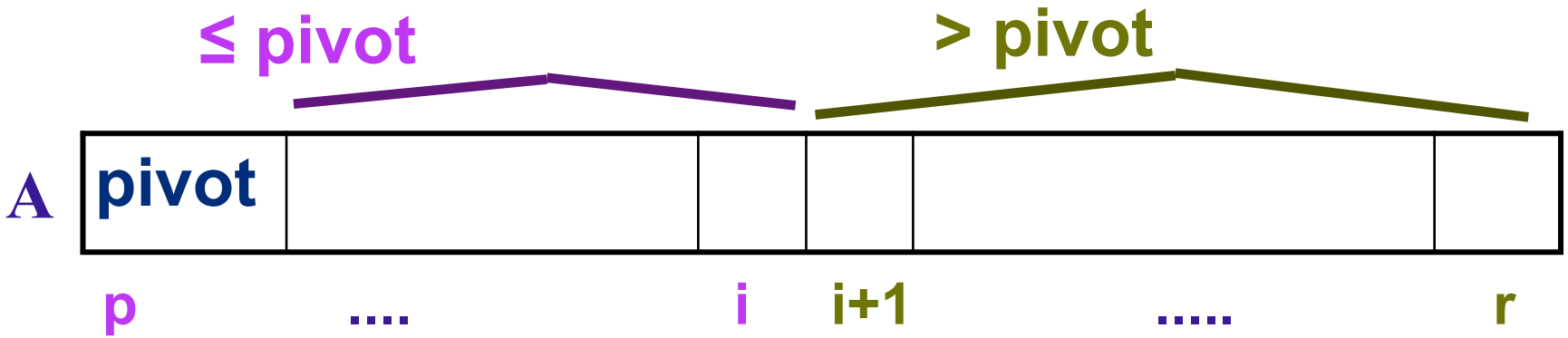
Pseudocodice di Partizione

```
Partizione(A,p,r)
 pivot = A[r]
 i = p-1   All'inizio non ci sono elementi ≤ pivot
 j = p     e nemmeno > pivot!
 while j ≤ r-1 do
       if  A[j] ≤ pivot then
             i =  i+1
             scambia A[i] con A[j]
       j= j+1
 scambia A[i+1] con A[r]
 i+1 è la posizione del pivot
 return i+1
```

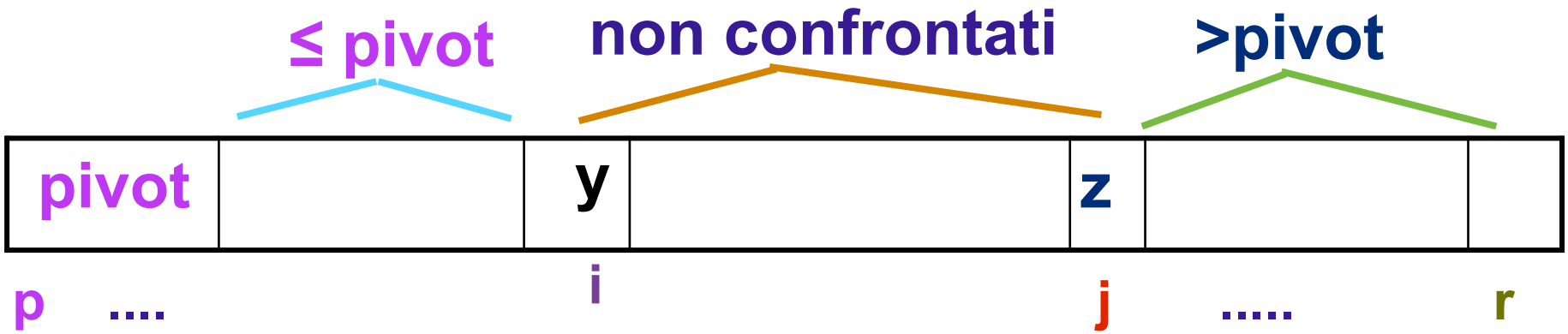
in ogni passo di esecuzione:
Gli elementi di indice tra p e i sono ≤ pivot
gli elementi di indice tra i+1 e j-1 sono > pivot

Partizione: l'idea di Hoare

obiettivo:



possibile stadio intermedio:



Esercizio

Si progetti un algoritmo che realizza una tripartizione degli elementi dell'array determinata da un pivot.
L'array finale deve avere tutti gli elementi minori del pivot a sinistra, seguiti da tutte le occorrenze del pivot a loro volta seguite dagli elementi più grandi del pivot.
Nell'esempio il pivot è l'ultimo elemento: