Sommario

L'algoritmo per l'ordinamento di Hoare, rivisitato per esaminare lo spazio di memoria impegnato dallo stack delle chiamate.

[CLRS] cap. 7, problema 7.4

QuickSort

```
QuickSort (A,p,r)
  if p < r then
  q ← Partizione(A,p,r)
  QuickSort(A,p,q-1)
  QuickSort(A,q+1,r)
```

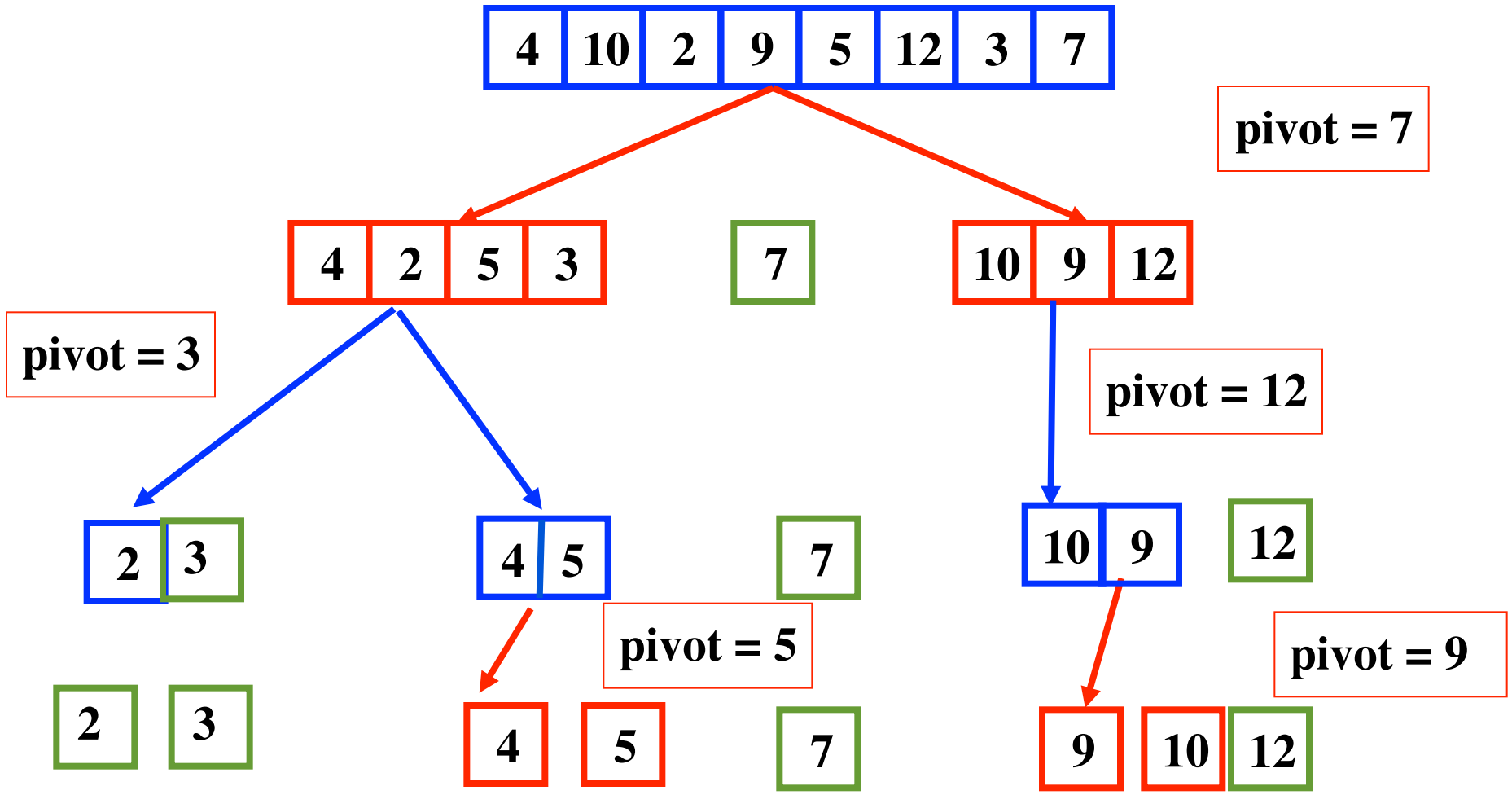
Partizione(A,p,r)

input A è una lista di interi e p ed r sono interi positivi

prec: 0≤p≤r≤n-1, se n è il numero degli elementi di A.

postc: restituisce la posizione che il pivot avrebbe se A fosse ordinata, dopo aver spostato il pivot in quella posizione e sposta gli elementi più piccoli o uguali del pivot a sinistra di questa posizione e quelli maggiori a destra.

QuickSort Partition semplificata

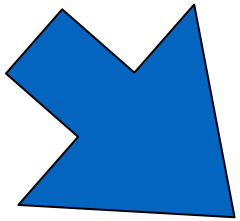


Fine chiamate sulla metà sinistra.

Eliminazione II chiamata

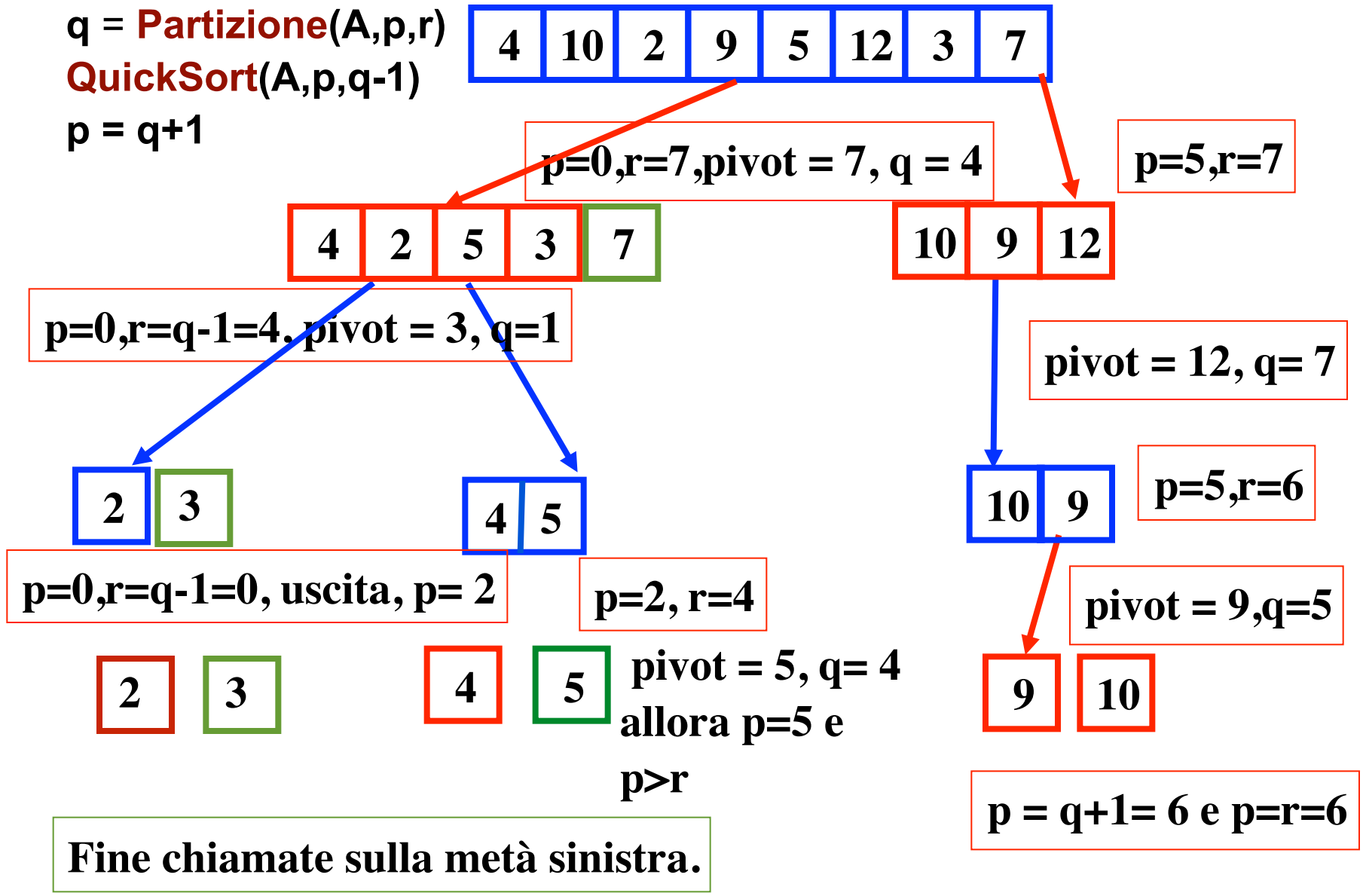
```
QuickSort (A,p,r)
  if p < r then
   q = Partizione(A,p,r)
  QuickSort(A,p,q-1)
  QuickSort(A,q+1,r)
```

E' possibile eliminare la seconda chiamata, sostituendola con una struttura iterativa, perché non ci sono istruzioni che la seguono



```
QuickSort (A,p,r)
 while (p < r):
      q = Partizione(A,p,r)
      QuickSort(A,p,q-1)
      p = q+1
```

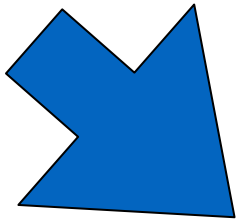
```
QuickSort (A,p,r)
while (p < r):
    q = Partizione(A,p,r)
    QuickSort(A,p,q-1)
    p = q+1
```



QuickSort

```
QuickSort (A,p,r)
  if p < r then
 q = Partizione(A,p,r)
QuickSort(A,p,q-1)
QuickSort(A,q+1,r)
```

E' possibile eliminare la seconda chiamata, sostituendola con una struttura iterativa, perché non ci sono istruzioni che la seguono



```
QuickSort' (A,p,r)
 while (p < r):
      q = Partizione(A,p,r)
      QuickSort'(A,p,q-1)
      p = q+1
```

QuickSort' esegue gli stessi passi di QuickSort ed è quindi corretto

QuickSort e stack delle chiamatate

```
QuickSort' (A,p,r)
 while (p < r):
      q = Partizione(A,p,r)
      QuickSort'(A,p,q-1)
      p = q+1
```

Se l'array è già ordinato crescente e prendiamo come pivot l'ultimo elemento:

QuickSort' (A,0,n-1) , q = n-1 e la chiamata è

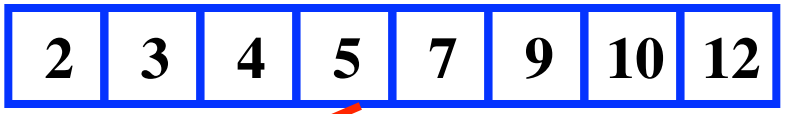
QuickSort' (A,0,n-2), poi di nuovo q = n-2 e

QuickSort' (A,0,n-3), …

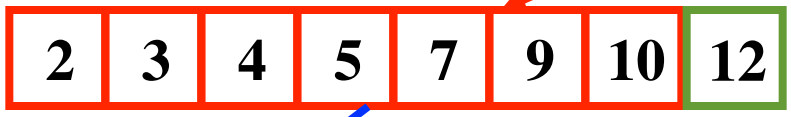
QuickSort' (A,0,0),

QuickSort'

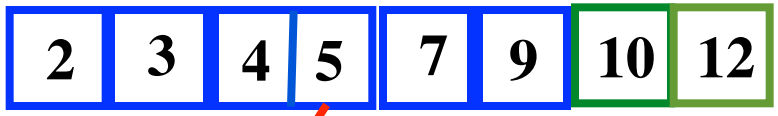
```
QuickSort' (A,p,r)
while (p < r):
    q = Partizione(A,p,r)
    QuickSort'(A,p,q-1)
    p = q+1
```



p = 0, r = 7, pivot = 12, q = 7



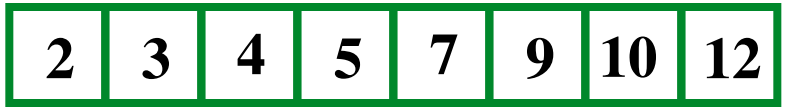
p = 0, r = 6, pivot = 10, q= 6



p = 0, r = 5, pivot = 9, q= 5

...

QuickSort'(A,0,1): p = 0, r = 1, pivot = 3, q= 1,



La chiamata QuickSort'(A,0,0) non ha effetto, si rientra nella chiamata QuickSort'(A,0,1) ponendo p = q+1 = 2 e poiché p > r il processo termina.

QuickSort' e stack delle chiamatate

```
QuickSort' (A,p,r)
 while (p < r):
      q = Partizione(A,p,r)
      QuickSort'(A,p,q-1)
      p = q+1
```

Se l'array è già ordinato crescente e prendiamo come pivot l'ultimo elemento:

lo stack delle chiamate ha profondità Θ(n).

Si può evitare?

Se facessimo in modo che ogni chiamata del QuickSort" sia sulla porzione di array più piccola, avrei profondità Θ(lg n), infatti questa chiamata al più riguarda la metà degli elementi!

QuickSort e stack delle chiamatate

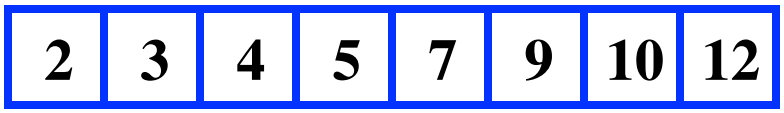
```
QuickSort”(A,p,r)
 while (p < r):
     q = Partizione(A,p,r)
     if (q - p < r - q)
          QuickSort”(A,p,q-1)
          p = q+1
     else
          QuickSort”(A,q+1,r)
          r = q-1
```

Ogni chiamata di QuickSort” è sulla porzione di array più piccola.

Poiché questa al più riguarda la metà degli elementi, lo stack ha profondità al più Θ(lg n)

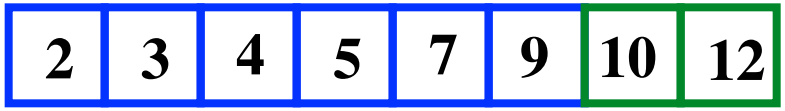
QuickSort”

```
QuickSort”(A,p,r)
while (p < r):
    q = Partizione(A,p,r)
    if (q - p < r - q)
        QuickSort”(A,p,q-1) p = q+1
    else
        QuickSort”(A,q+1,r) r = q-1
```



p = 0, r = 7, pivot = 12, q = 7
chiamata QuickSort”(A,8,7)
dalla quale si esce subito, r = q-1 = 6

p = 0, r = 6, pivot = 10, q= 6
chiamata QuickSort”(A,7,6)
dalla quale si esce subito, r= q-1= 5



La profondità dello stack è Θ(1)

...

p = 0, r = 1, pivot = 3, q= 1
chiamata QuickSort”(A,2,1)
dalla quale si esce subito, r= q-1= 0, uscita dalla prima chiamata, perché p=r.