Sommario

L’algoritmo per l’ordinamento di Hoare.

[CLRS] cap. 7, par. 7.1, 7.2

QuickSort (1962, The Computer Journal)

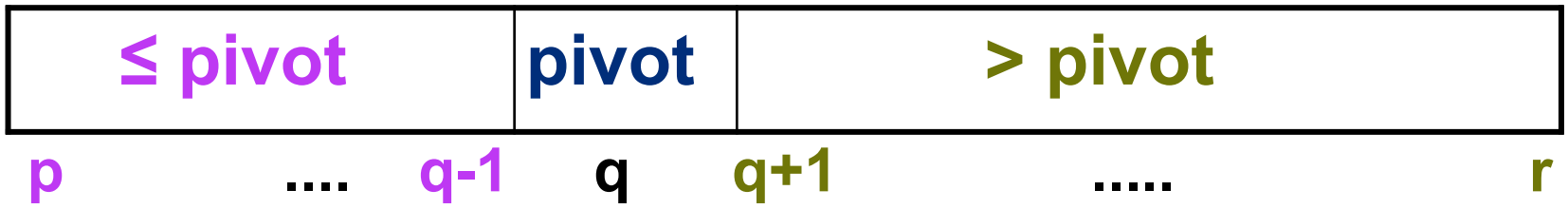
Hoare ha vinto nel 1980 il Turing Award, il premio più prestigioso dell' ACM (Association for Computing Machinery), per i fondamentali contributi alla definizione e al progetto dei linguaggi di programmazione (semantica assiomatica: logica di Hoare, Communicating Sequential Processes) .

Charles Antony Richard Hoare - 1934

Il Quicksort è stato inserito nell'elenco dei 10 algoritmi che hanno avuto la maggiore influenza sullo sviluppo e la pratica dell'informatica nel ventesimo secolo, in un articolo apparso su *Computing in Science & Engineering*, 2000, una pubblicazione dell' American Institute of Physics e dell' IEEE Computer Society. Gli autori della lista sono Jack Dongarra, Università del Tennessee e Francis Sullivan del Center for Computing Sciences at the Institute for Defense Analyses.

DIVIDE ET IMPERA E QUICKSORT

Divide: scegli un elemento pivot (perno) dell'array A[p...r] e partizionalo in due: uno con gli elementi ≤ pivot, A[p...q-1] e l'altra con gli elementi > pivot, A[q+1...r]



Impera: metti il pivot al suo posto e ordina le due sottosequenze ricorsivamente, usando il quicksort

Combina: i due sottoarray sono ordinati sul posto, quindi non c'è combinazione

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)

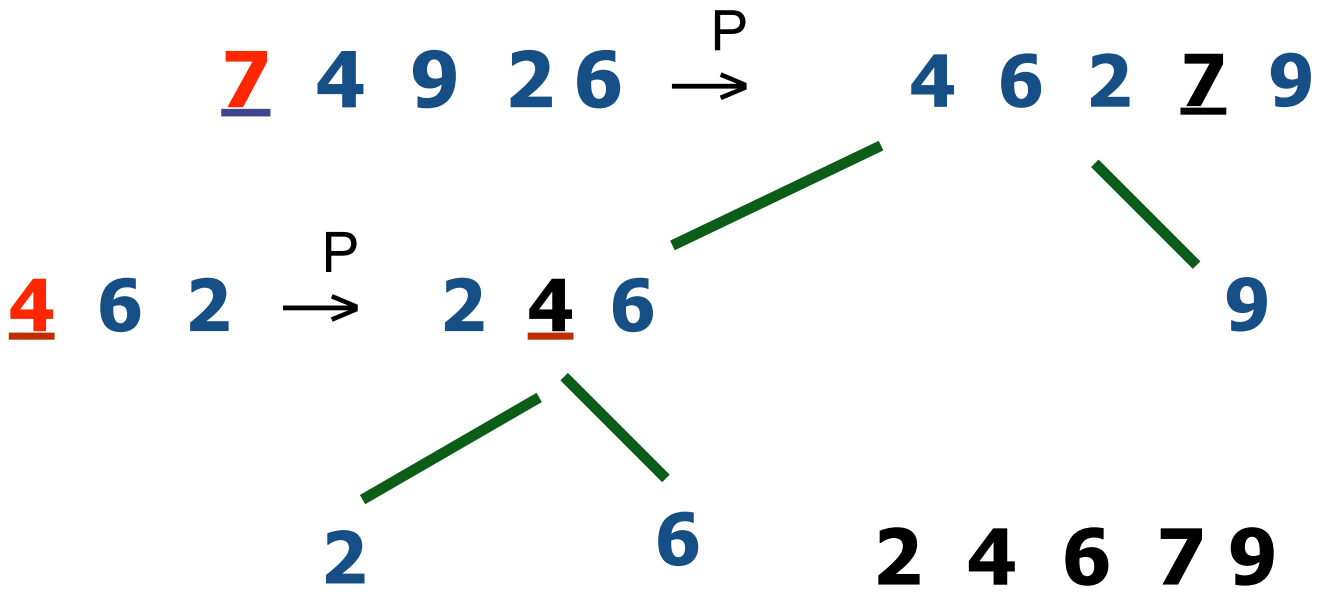
input A è una lista di interi e p ed r sono interi positivi

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

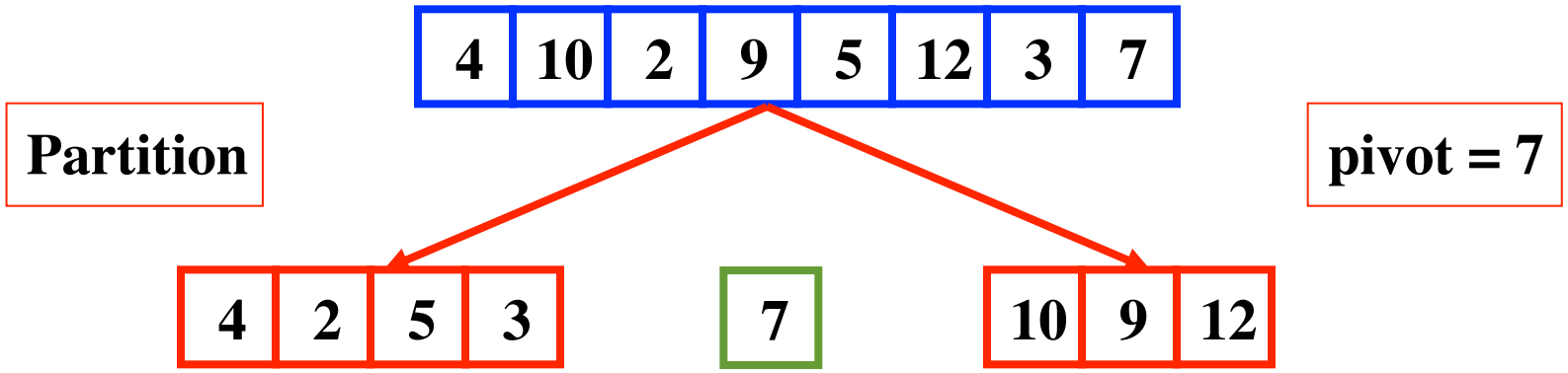
postc: restituisce la posizione che il pivot avrebbe se A fosse ordinata, dopo aver spostato il pivot in quella posizione e sposta gli elementi più piccoli o uguali del pivot a sinistra di questa posizione e quelli maggiori a destra.

QuickSort

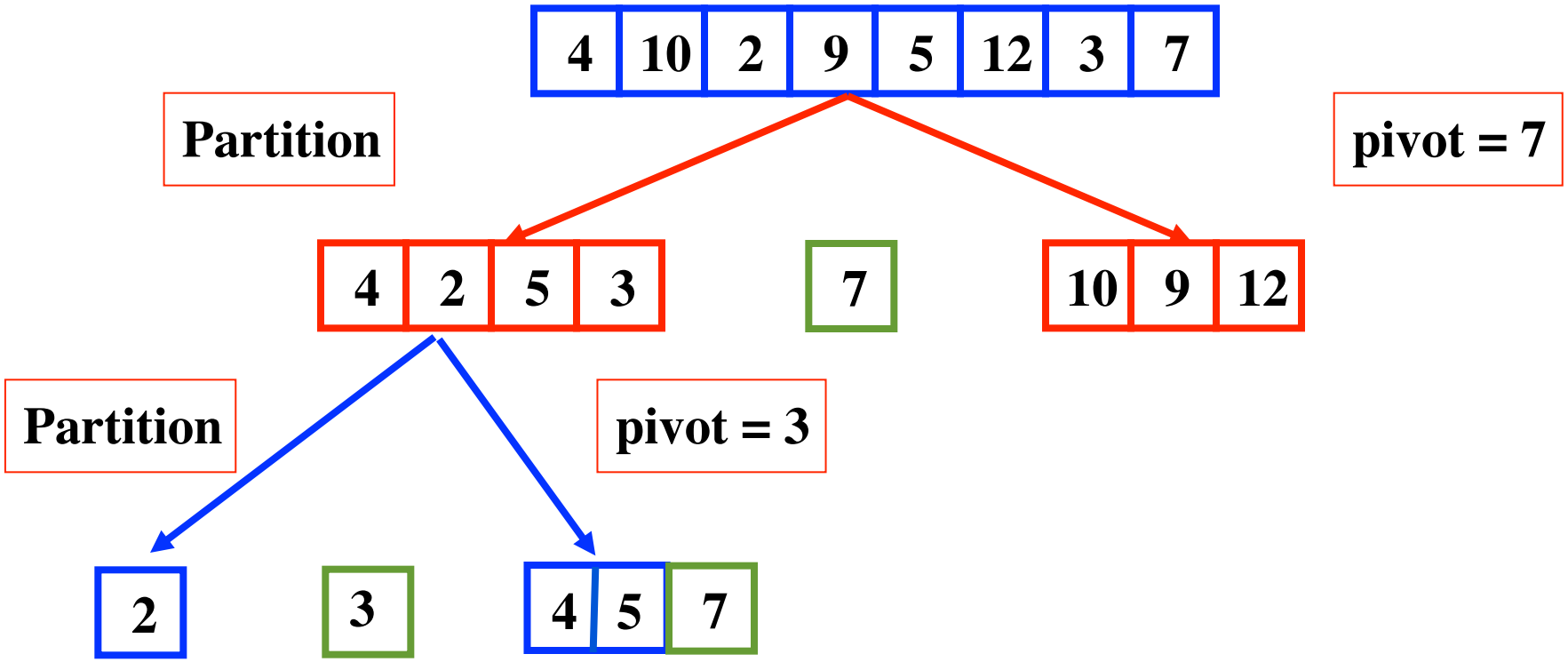
```
QuickSort (A,p,r)
  if p < r then
        q = Partizione(A,p,r)
        QuickSort(A,p,q-1)
        QuickSort(A,q+1,r)
```



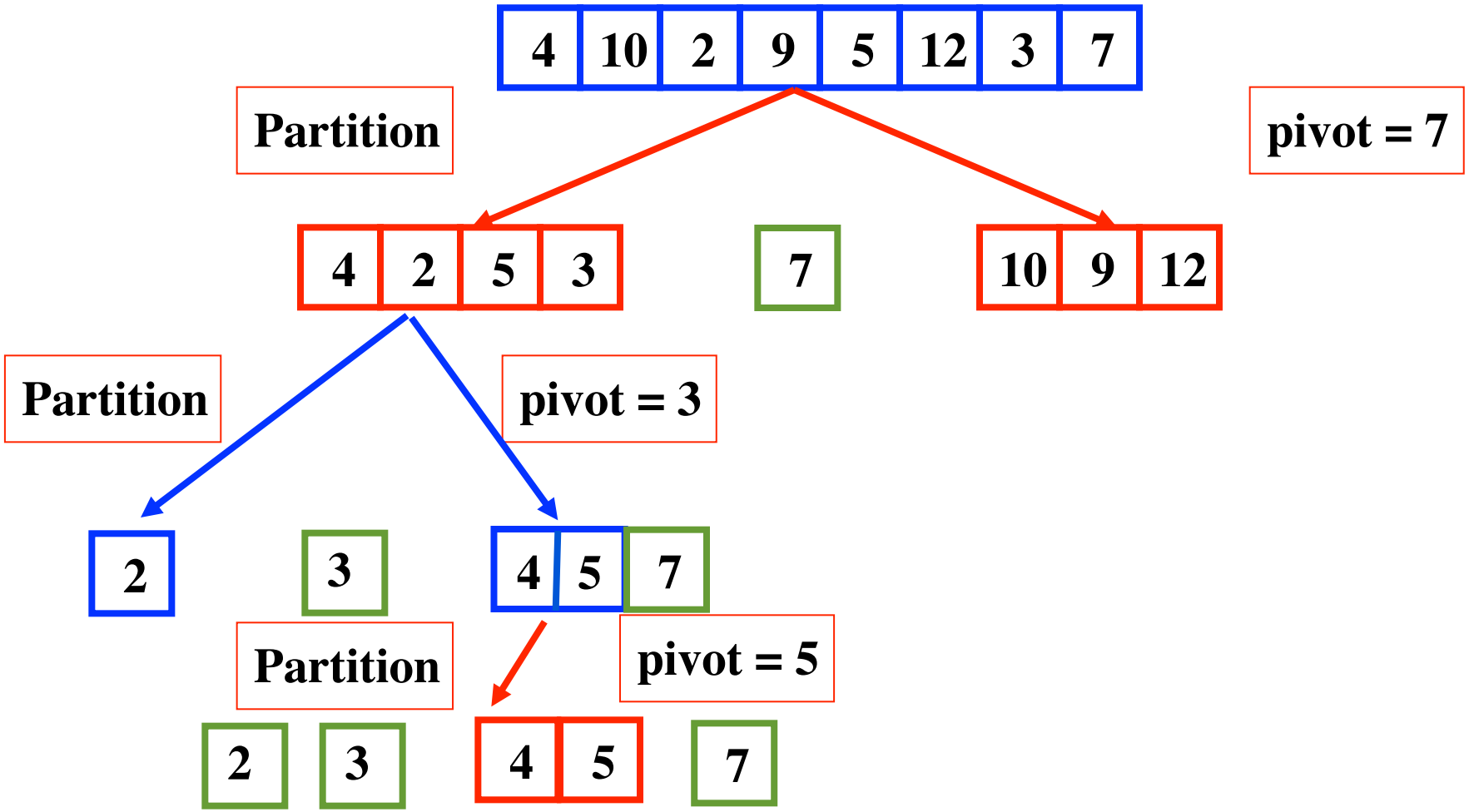
QuickSort



QuickSort

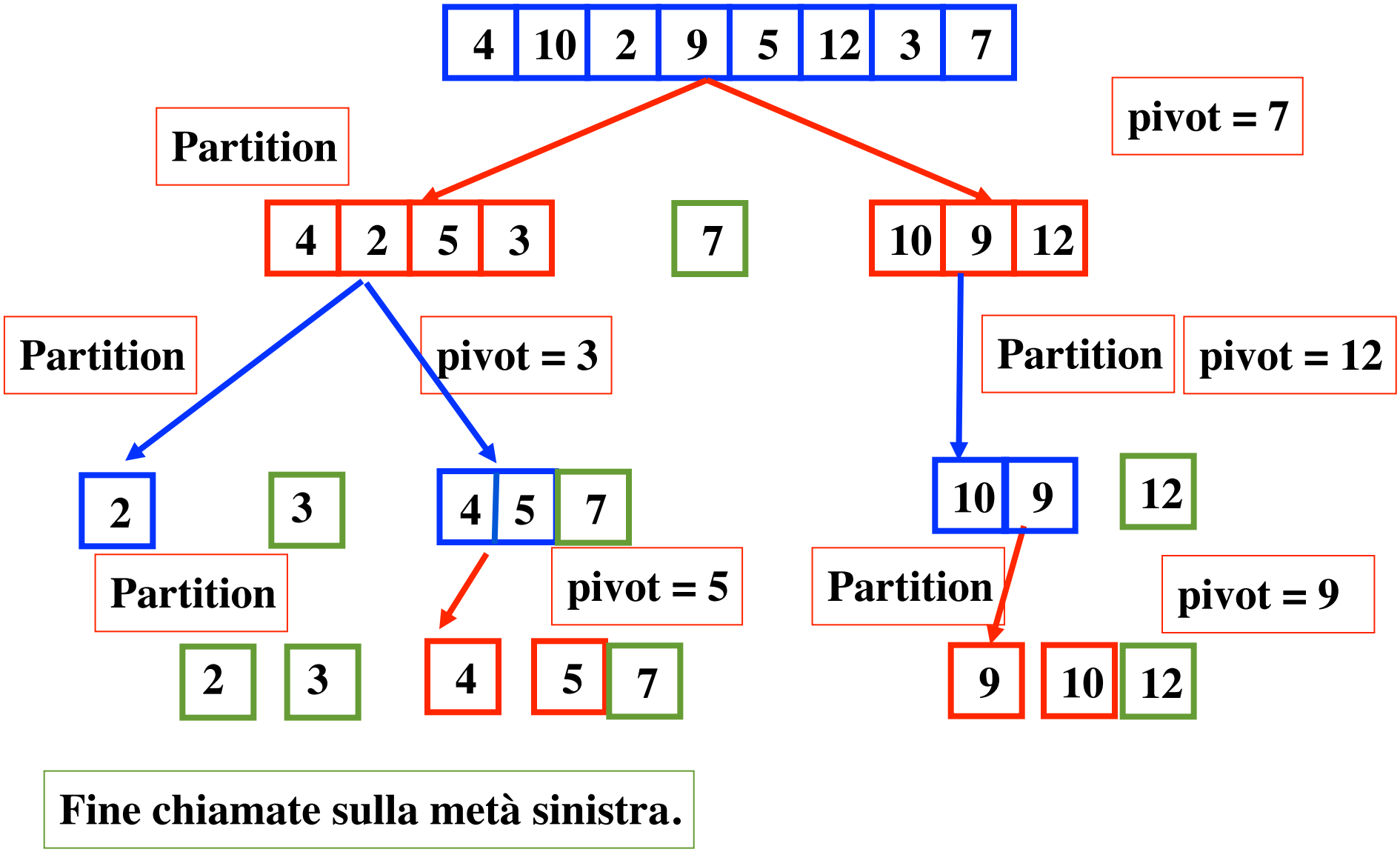

4 10 2 9 5 12 3 7
Partition
pivot = 7
4 2 5 3
7
10 9 12
Partition
pivot = 3
2
3
4 5 7

QuickSort



Fine chiamate sulla metà sinistra.

QuickSort



Fine chiamate sulla metà sinistra.

QuickSort: correttezza

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

Per induzione.

Base. Se il numero degli elementi, n, è ≤1, cioè p≥ r, l'algoritmo non fa nulla, correttamente. Se il numero degli elementi, n, è 2 allora p < r e il secondo elemento, il pivot, viene messo al posto giusto e questo già ordina l'array. Nei due casi le due chiamate lavorano su meno di 2 elementi non modificando l'array.

Passo induttivo. Sia corretto su array di m elementi con m<n. Poichè gli elementi da p a q-1 e quelli da q+1 a r sono due sottoarray con meno di n elementi, questi vengono correttamente ordinati, per ipotesi induttiva. Tutto l'array è ordinato perchè dopo l'esecuzione della partizione sappiamo che gli elementi che precedono A[q] sono tutti minori o uguali di A[q] e quelli che lo seguono sono maggiori

QuickSort: analisi

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

0 elementi

n-1 elementi

Il caso peggiore:

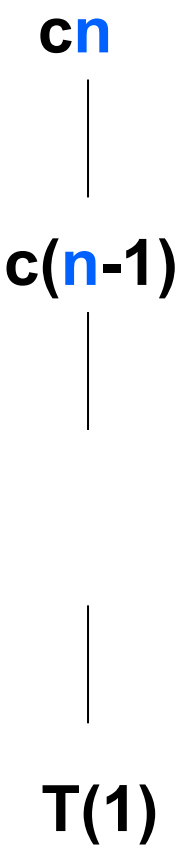
$T(n) = T(n-1) + T(0) + \Theta(n) = T(n-1) + \Theta(n)$

Si verifica se gli elementi sono ordinati in ordine decrescente, infatti se il pivot è l'ultimo elemento e va all'inizio, perchè è il minimo, la prima chiamata non fa nulla, ma la seconda "lavora" su n-1 elementi

La ricorrenza nel caso peggiore: sviluppo

$T(n) = T(n-1) + cn$ se $n > 1$
$T(n) = d$ altrimenti

Sviluppando la ricorrenza:
$T(n) = T(n-1) + cn =$
$T(n-2) + c(n-1) + cn =$
$T(n-3) + c(n-2) + c(n-1) + cn =$
…
$T(n-i) + c(n-(i-1)) + \ldots + c(n-2) + c(n-1) + cn$
…
$= T(1) + c(n-(n-2)) + \ldots + c(n-2) + c(n-1) + cn$
$= d + c \sum_{i=2 \ldots n} i = O(n^2)$.



La ricorrenza nel caso peggiore: passo induttivo

$T(n) = T(n-1) + cn$ se $n > 1$

$T(n) = d$ altrimenti

Dimostriamo per induzione che $T(n) = O(n^2)$, cioè che esistono due costanti positive k e n_0 tali che $T(n) \leq kn^2$ per ogni $n \geq n_0$.

Passo induttivo.

Ipotesi: per $n>1$, $T(n-1) \leq k(n-1)^2$

$T(n) \leq k(n-1)^2 + cn = kn^2 + k - 2kn + cn$

Cerchiamo il valore di k tale che

$kn^2 + k - 2kn + cn \leq kn^2$ cioè

$k - 2kn + cn \leq 0$, quindi

$cn \leq 2kn - k = k(2n - 1)$

questo è vero per $k \geq c+1$ e $n \geq 1$

La ricorrenza nel caso peggiore: passo base

$T(n) = T(n-1) + cn$ se $n > 1$
$T(n) = d$ altrimenti

Dimostriamo per induzione che $T(n) = O(n^2)$, cioè che esistono due costanti positive k e n_0 tali che $T(n) \leq kn^2$ per ogni $n \geq n_0$.

Passo base

$T(1) = T(0) = d$

$T(2) = T(1) + 2c = d+2c$

Quindi prendiamo $k = c + d$ e possiamo concludere che per $k=d+c$ e $n_0 = 2$ vale $T(n) \leq kn^2$ per ogni $n \geq n_0$.
Quindi che $T(n) = O(n^2)$

QuickSort: analisi

```
QuickSort (A,p,q)
  if p < q then
        i = Partition(A,p,q)
       QuickSort(A,p,i-1)        n/2
       QuickSort(A,i+1,r)        n/2
```

Il caso migliore:
$T(n) = 2T(n/2) + \Theta(n)$
$T(n) = \Theta(n \log n)$

Si può dimostrare che nel caso medio:
$T(n) = \Theta(n \log n)$

QuickSort: un caso sfavorevole ?

```
QuickSort (A,p,q)
  if p < q then
        i = Partition(A,p,q)
        QuickSort(A,p,i-1)        n/10
        QuickSort(A,i+1,r)        9n/10
```

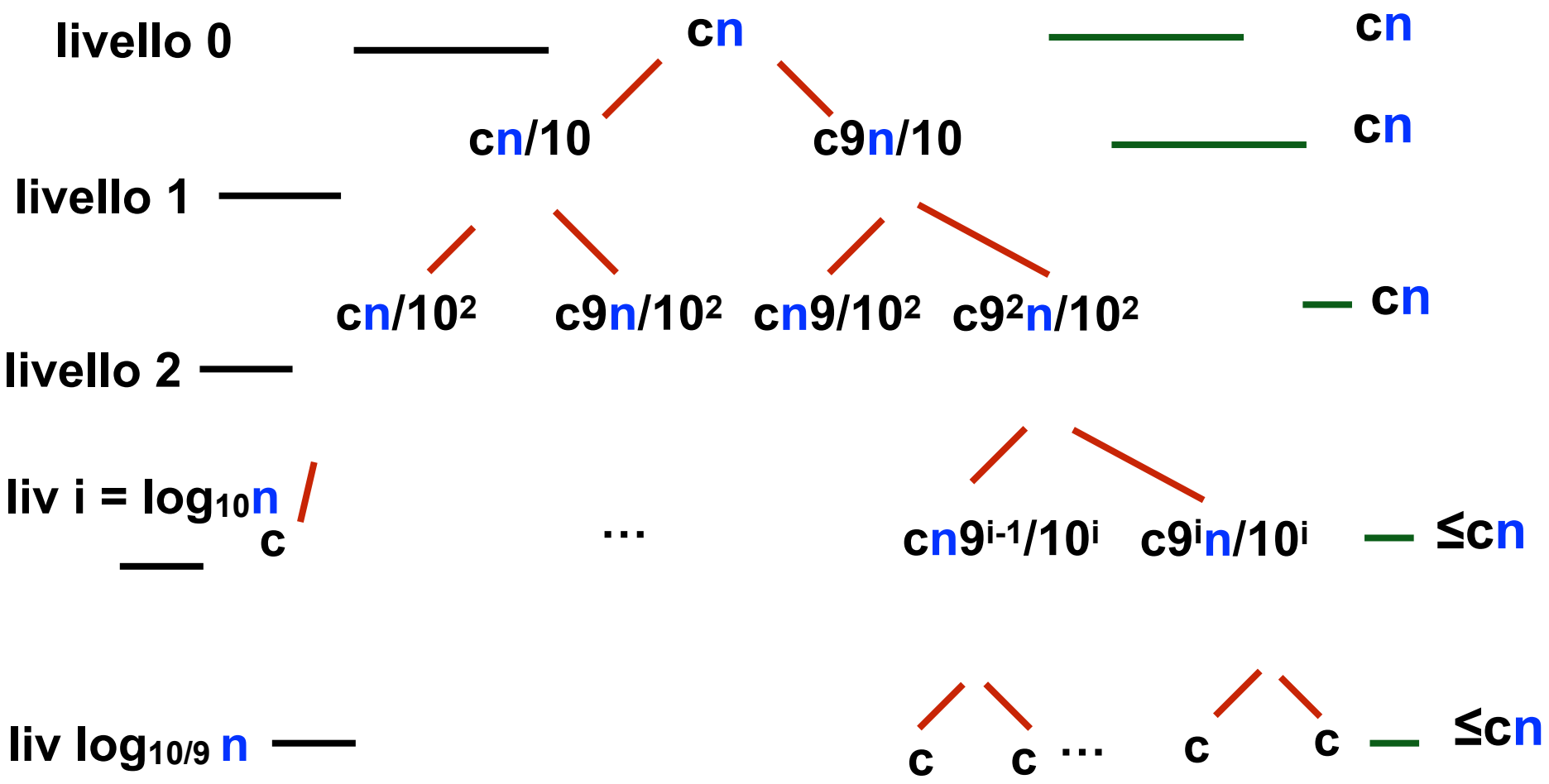
In questo caso:
$T(n) = T(n/10) + T(9n/10) + \Theta(n)$

Si può dimostrare che vale ancora:
$T(n) = \Theta(n \log n)$

Ricorrenza

$T(n) = c$ se $n \leq 1$
$T(n) = T(n/10) + T(9n/10) + cn$

Per quale i $9^i n/10^i = 1$?
$9^i n/10^i = 1$ sse $n = 10^i/9^i$
sse $\log_{10/9} n = i$



QuickSort

$T(n) = d$ se $n \leq 1$
$T(n) = T(9/10n) + T(1/10n) + cn$
ha soluzione $O(n \lg n)$.

Infatti l'albero della ricorsione ha altezza (parte intera di) $\log_{10/9} n$ e ogni livello ha un costo totale $\leq cn$.

Dimostriamo per induzione $T(n) = O(n \lg n)$.

Prova induttiva

T(n) = c se n≤1

T(n) = T(9/10n) + T(n/10) + cn

Vogliamo far vedere che T(n) = O(nlg n) cioè che esistono due costanti positive k e n_0 tali che(n) ≤ knlg n, per ogni n ≥ n_0

Passo induttivo:

Supponiamo che sia vero che T(m) = O(mlgm) per ogni m<n.

Consideriamo T(9/10n) prima e poi T(n/10)

Allora per ipotesi induttiva

T(9/10n) ≤ k(9/10n)lg 9n/10 =

k(9/10n)(lg 9n - lg10) =

k(9/10n)(lg 9 + lg n - lg10) ≤

k(9/10n)(lg n - (lg10 - lg 9) ≤

k9/10nlg n **perchè (lg10 - lg 9) > 0**

Prova induttiva

Consideriamo T(n/10)

sempre per ipotesi induttiva si ha
$T(n/10) \leq k(n/10)\lg n/10$
$= k(n/10)(\lg n - \lg 10) \leq$
$kn/10(\lg n - 3)$ **perchè $\lg 10 > 3$**

Consideriamo T(n):
$T(n) = T(9/10n) + T(n/10) + cn \leq$
$k9/10n\lg n + kn/10\lg n - 3/10kn+cn =$
$k\, n \lg n - 3/10kn + cn$

Passo induttivo: fine

Abbiamo ottenuto $T(n) \leq k\, n \lg n - 3/10kn + cn$

Bisogna controllare se è vero che esiste un k tale che

$k\, n \lg n - 3/10kn + cn \leq k\, n \lg n$ ⇔

$- 3/10kn + cn \leq 0$ ⇔

$cn \leq 3/10kn$ ⇔

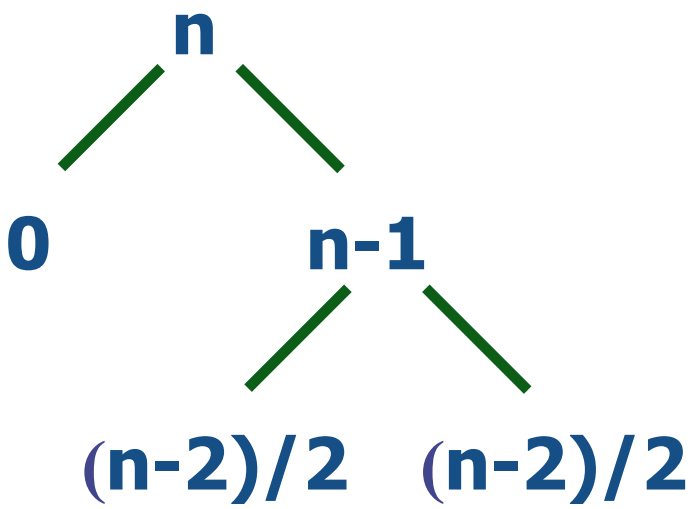
$10/3cn \leq kn$

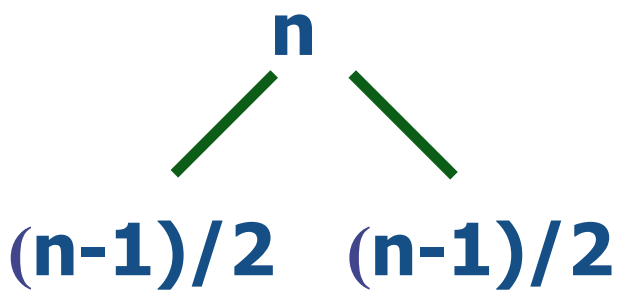
vero per esempio per $k = 4c$ e $n \geq 1$.

Considerate il passo base per esercizio.

Caso medio

In generale è difficile che ogni chiamata della funzione di Partizione divida il sotto array in due sotto array della stessa dimensione.
Potrà capitare qualche divisione "favorevole" e qualche sfavorevole. Se si alternassero come nell'esempio,





si avrebbe la stessa complessità asintotica del caso in cui tutte le divisioni fossero favorevoli.