UNITELMA SAPIENZA

SAPIENZA
UNIVERSITÀ DI ROMA
DIPARTIMENTO DI INFORMATICA

Corso di Introduzione agli algoritmi
Prof.ssa Tiziana Calamoneri

Il problema dell'Ordinamento: il Quick sort

Quick sort (1)

L'algoritmo ***quick sort*** (***ordinamento veloce***) ha costo $O(n^2)$ nel caso peggiore ma, nella pratica è spesso la soluzione migliore per grandi valori di *n* perché:

- il suo tempo di esecuzione atteso è $\Theta(n \log n)$;
- i fattori costanti nascosti sono molto piccoli;
- permette l'ordinamento "in loco".

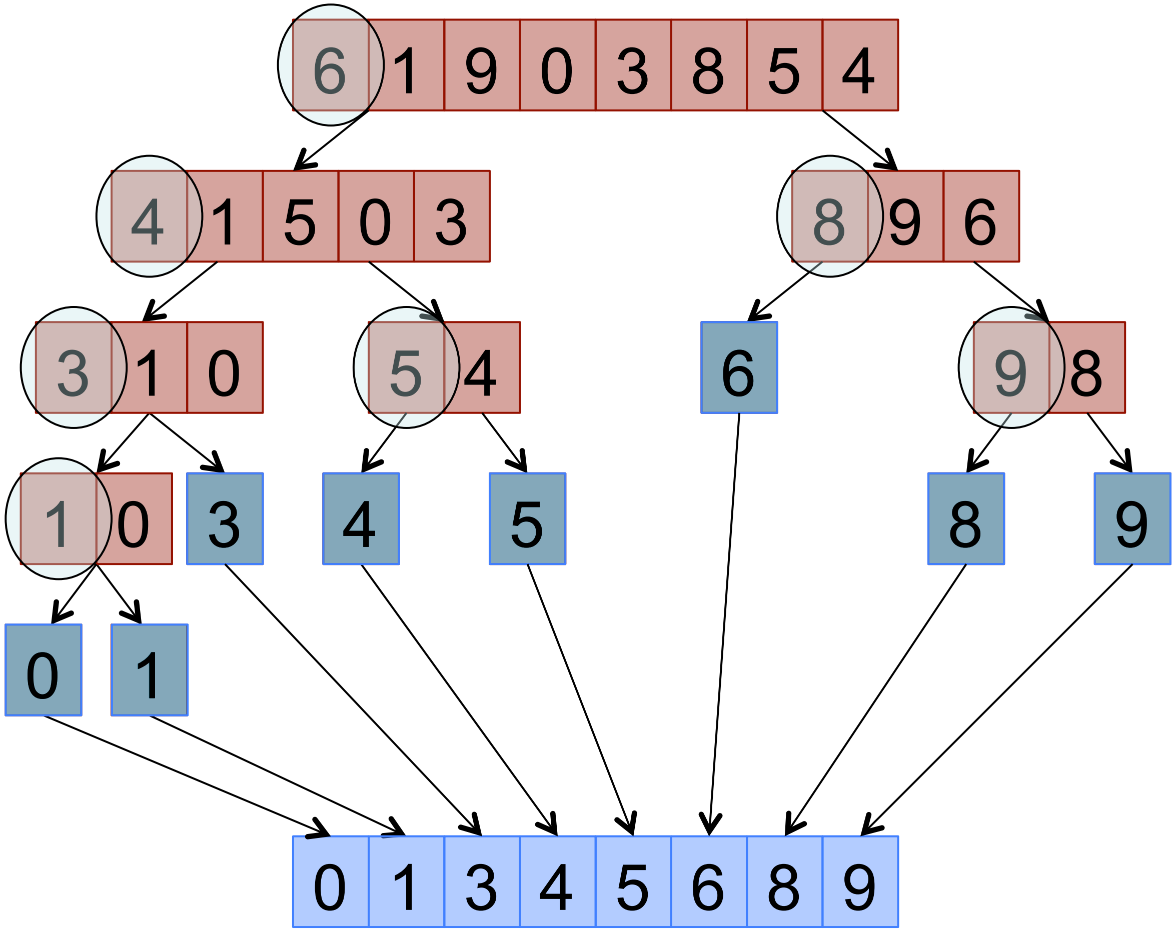
Riunisce i vantaggi del selection sort (ordinamento in loco) e del merge sort (ridotto tempo di esecuzione). Ha però lo svantaggio dell'elevato costo computazionale nel caso peggiore.

Quick sort (2)

L'algoritmo ***quick sort*** è un algoritmo ricorsivo che adotta una tecnica algoritmica detta ***divide et impera***:

- *divide*: nella sequenza di *n* elementi seleziona un ***pivot***. La sequenza viene quindi divisa in due sottosequenze: quella degli elementi minori o uguali del pivot, e quella degli elementi maggiori o uguali del pivot;
- *passo base*: la ricorsione procede fino a quando le sottosequenze sono costituite da un solo elemento;
- *impera*: le due sottosequenze vengono ordinate ricorsivamente;
- *combina*: non occorre.

Quick sort (3)



•***divide***: nella sequenza di *n* elementi seleziona un ***pivot***. La sequenza viene quindi divisa in due sottosequenze: quella degli elementi minori o uguali del pivot, e quella degli elementi maggiori o uguali del pivot;

•***passo base***: la ricorsione procede fino a quando le sottosequenze sono costituite da un solo elemento;

•***impera***: le due sotto-sequenze vengono ordinate ricorsivamente;

•***combina***: non occorre

Quick sort (4)

```
Funzione Quick_sort (A: vettore; ind_primo, ind_ultimo: intero)
   if (ind_primo < ind_ultimo)
        then
                ind_medio ← Partiziona (A, ind_primo, ind_ultimo)
                Quick _sort (A, indice_primo, indice_medio)
                Quick _sort (A, indice_medio + 1, indice_ultimo)
   return
```

Quick sort (5)

```
Funzione Partiziona (A: vettore; ind_primo, ind_ultimo: intero)
pivot ← A[indice_primo]                            //scelta arbitraria
i ← indice_primo - 1;               Θ(1)
j ← indice_ultimo + 1               Θ(1)
while true                          #di volte tale da scorrere tutto A
    repeat
        j ← j – 1                   Θ(1)
    until A[j] ≤ pivot
    repeat
        i ← i + 1                   Θ(1)
    until A[i] ≥ pivot
    if (i < j) scambia A[i] e A[j]  Θ(1)
    else return j
```

costo di Partiziona: $\Theta(n)$

Quick sort (6)

Costo computazionale del Quick Sort:

Funzione Quick_sort (A: vettore; ind_primo, ind_ultimo)	$T(n)=$
if (ind_primo < ind_ultimo)	$\Theta(1)+$
then	$\Theta(1)+$
ind_medio ← Partiziona (A, ind_primo, ind_ultimo)	$\Theta(n)+$
Quick _sort (A, indice_primo, indice_medio)	$T(k)+$
Quick _sort (A, indice_medio + 1, indice_ultimo)	$T(n-k)+$
return	$\Theta(1)$

$T(n)=T(k)+T(n-k)+\Theta(n)$

$T(1)=\Theta(1)$

→ *Non sappiamo risolvere con alcuno dei metodi studiati...*

Quick sort (7)

Costo computazionale del Quick Sort (segue):

- **Caso peggiore**:
  quello in cui, ad ogni passo, la dimensione di uno dei due sotto-problemi da risolvere è 1. L'equazione di ricorrenza diventa: $T(n) = T(n - 1) + \Theta(n)$ $SOL: \Theta(n^2)$

- **Caso migliore:**
  quello in cui, ad ogni passo, la dimensione dei due sotto-problemi è identica. L'equazione di ricorrenza diventa:
  $$T(n) = 2T(n/2) + \Theta(n) \quad SOL: \Theta(n \log n)$$

Quick sort (8)

Costo computazionale del Quick Sort (segue):

- Caso medio:
  Valutiamo il costo computazionale del caso medio, nell'ipotesi che il valore del pivot suddivida con uguale probabilità *1/n* la sequenza da ordinare in due sotto-sequenze di dimensioni *k* ed *n – k*, per tutti i valori di *k*.

$$T(n) = \frac{1}{n}\left[\sum_{k=1}^{n-1}\big(T(k)+T(n-k)\big)\right] + \Theta(n) =$$

$$= \frac{2}{n}\sum_{q=1}^{n-1} T(q) + \Theta(n)$$

Costo computazionale del Quick Sort (segue):

- Caso medio (segue):
  Usiamo il metodo di sostituzione ed ipotizziamo la soluzione
  $$Tn \leq a\, n \log n + b$$

  Sostituiamo nel caso base:
  $$T(1) \leq a\, 1 \log 1 + b = b$$
  vera per b opportuno.

  Sostituiamo nell'equazione generica:

$$T(n) = \frac{2}{n}\sum_{q=1}^{n-1} T(q) + \Theta(n) \leq \frac{2}{n}\sum_{q=1}^{n-1}(aq\log q + b) + \Theta(n) = \frac{2a}{n}\boxed{\sum_{q=1}^{n-1} q\log q} + \frac{2b}{n}(n-1) + \Theta(n)$$

Quick sort (10)

Costo computazionale del Quick Sort (segue):

- Caso medio (segue):

≤ log(n/2)=log n -1

≤ log n

$$\sum_{q=1}^{n-1} q\log q = \sum_{q=1}^{\lceil n/2\rceil -1} q\log q + \sum_{q=\lceil n/2\rceil}^{n-1} q\log q$$

$$\sum_{q=1}^{n-1} q\log q \le \sum_{q=1}^{\lceil n/2\rceil -1} q(\log n - 1) + \sum_{q=\lceil n/2\rceil}^{n-1} q\log n = (\log n - 1)\sum_{q=1}^{\lceil n/2\rceil -1} q + \log n \sum_{q=\lceil n/2\rceil}^{n-1} q =$$

$$= \log n \sum_{q=1}^{\lceil n/2\rceil -1} q - \sum_{q=1}^{\lceil n/2\rceil -1} q + \log n \sum_{q=\lceil n/2\rceil}^{n-1} q = \log n \sum_{q=1}^{n-1} q - \sum_{q=1}^{n/2-1} q =$$

$$= \log n \frac{(n-1)n}{2} - \frac{1}{2}\left(\frac{n}{2} - 1\right)\frac{n}{2} = \frac{1}{2}n^2\log n - \frac{1}{8}n^2 - \frac{n}{2}\log n + \frac{1}{8} \le \frac{1}{2}n^2\log n - \frac{1}{8}n^2$$

Quick sort (11)

Costo computazionale del Quick Sort (segue):

- Caso medio (segue):
  Sostituiamo nella disuguaglianza

$$T(n) \le \frac{2a}{n}\sum_{q=1}^{n-1} q\log q + \frac{2b}{n}(n-1) + \Theta(n)$$

$$T(n) \le \frac{2a}{n}\left(\frac{1}{2}n^2\log n - \frac{1}{8}n^2\right) + \frac{2b}{n}(n-1) + \Theta(n) \le an\log n + b + \left[\Theta(n) + b - \frac{a}{4}n\right]$$

Per *a* sufficientemente grande la disuguaglianza è verificata, quindi, nel caso medio:

$$T(n)=O(n \log n)$$

Costo computazionale del Quick Sort (segue):

- Caso medio (segue):
  OSS. L'analisi ora fatta è valida nell'ipotesi che il valore del pivot sia equiprobabile e, quando questo è il caso, **il quick sort è considerato l'algoritmo ideale per input di grandi dimensioni**.
  A volte però l'ipotesi di equiprobabilità non è soddisfatta (ad esempio quando i valori in input sono "poco disordinati") e le prestazioni dell'algoritmo degradano.
  Per ovviare a tale inconveniente si possono adottare delle tecniche volte a ***randomizzare*** la sequenza da ordinare, cioè volte a disgregarne l'eventuale regolarità interna. Tali tecniche mirano a rendere l'algoritmo indipendente dall'input, e quindi consentono di ricadere nel caso medio…

Costo computazionale del Quick Sort (segue):

- Caso medio (segue):
  Alcune di tali tecniche sono:
  - prima di avviare l'algoritmo, alla sequenza da ordinare viene applicata una permutazione degli elementi generata casualmente;
  - l'operazione di partizionamento sceglie casualmente come pivot il valore di uno qualunque degli elementi della sequenza anziché sistematicamente il valore di quello più a sinistra.

Quick sort (13)

Ecco una visualizzazione inusuale del Quicksort:

https://www.youtube.com/watch?feature=player_embedded&v=kDgvnbUIqT4#!

- Sia dato un vettore di lunghezza *n* contenente solo valori 0 e 2. Si progetti un algoritmo con costo computazionale lineare che modifichi il vettore in modo che tutte le occorrenze di 0 si trovino più a sinistra di tutte le occorrenze di 2.

- Si considerino i valori 0 1 2 3 4 5 6 7. Si determini una permutazione di questi valori che generi il caso peggiore per l'algortimo Quick sort.

- Calcolare il costo computazionale del Quick sort nel caso in cui il vettore contenga tutti elementi uguali ed in cui sia già ordinato da destra a sinistra.