Corso di laurea in Informatica
Introduzione agli Algoritmi
Didattica blended

Il problema dell'ordinamento: algoritmo Quicksort

Angelo Monti

SAPIENZA
UNIVERSITÀ DI ROMA

Sulla base delle slides a cura di T. Calamoneri e G. Bongiovanni per il corso di informatica generale AA 2019/2020

Quicksort

L'algoritmo ***quicksort*** (***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.

Quicksort

Anche l'algoritmo ***quicksort*** è un algoritmo ricorsivo che adotta una tecnica algoritmica detta ***divide et impera***:

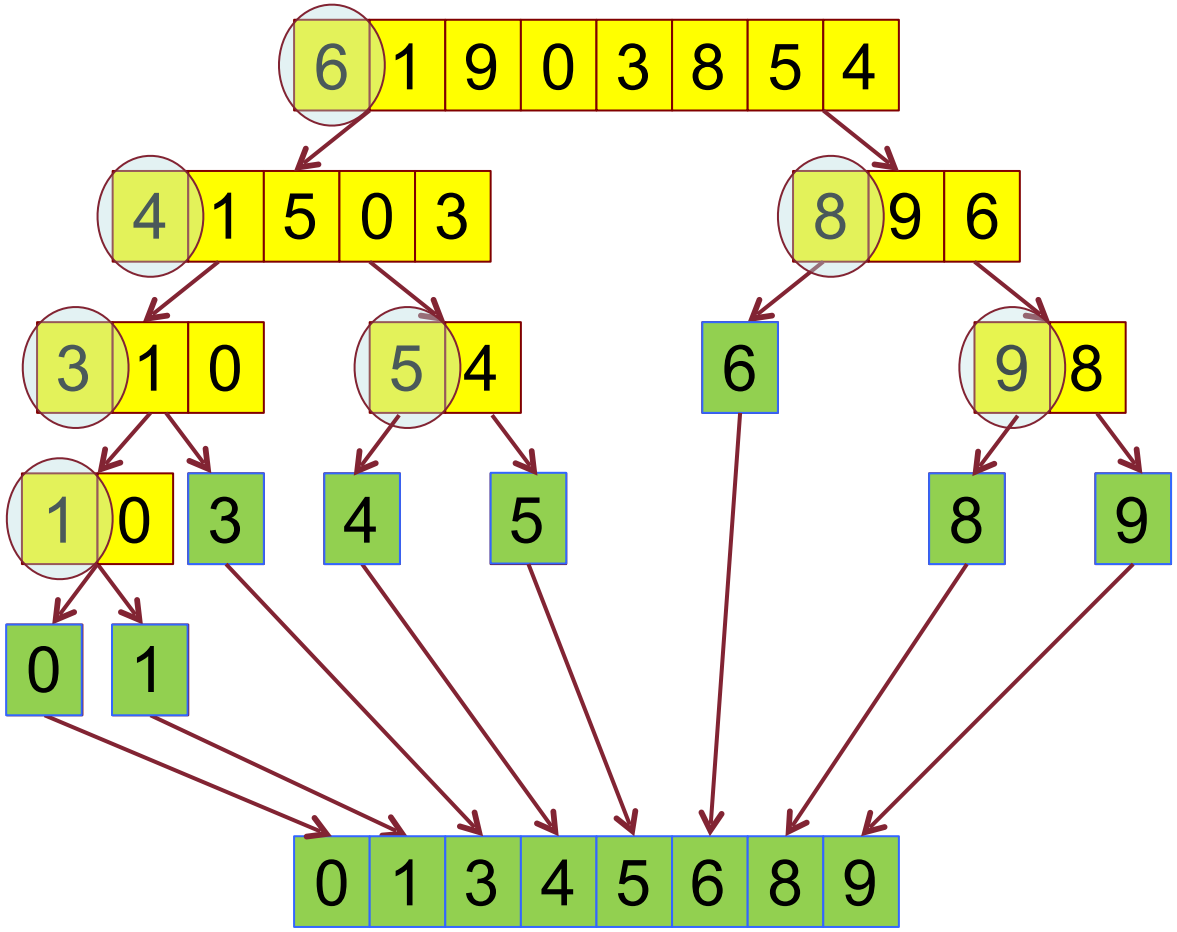
- ***divide***: nella sequenza di *n* elementi si seleziona un ***pivot***. La sequenza viene quindi divisa in due sottosequenze: quella degli elementi minori o uguali del pivot, e quella degli elementi maggiori o uguali del pivot

- ***impera***: le due sottosequenze vengono ordinate ricorsivamente

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

- ***combina***: non occorre.

Quicksort



- ***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;
- ***impera***: le due sotto-sequenze vengono ordinate ricorsivamente;
- ***passo base*:** la ricorsione procede fino a quando le sottosequenze sono costituite da un solo elemento;
- ***combina***: non occorre

Quicksort

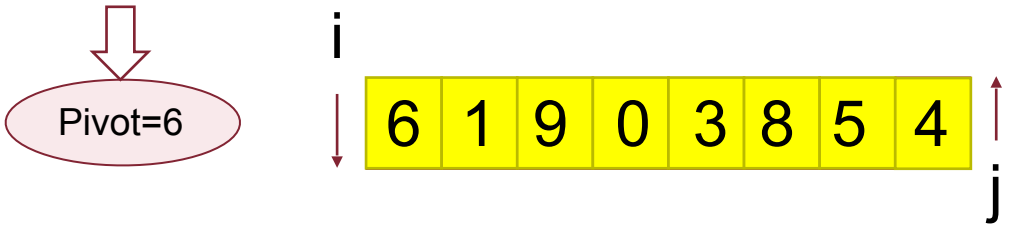
Lo pseudocodice del Quicksort è il seguente:

```
def Quick_sort(lista, ind_primo, ind_ultimo):
   if ind_primo < ind_ultimo:
      ind_medio = Partiziona(lista, ind_primo, ind_ultimo)
      Quick _sort(lista, ind_primo, ind_medio)
      Quick _sort(lista, ind_medio+1, ind_ultimo)
```

In questa implementazione `ind_medio` è l'indice dell'estremo superiore della porzione di sinistra (quella contenente elementi minori o uguali del pivot). Il suo valore quando ind_primo<ind_ultimo è ***sempre*** compreso fra `ind_primo` e `ind_ultimo-1`

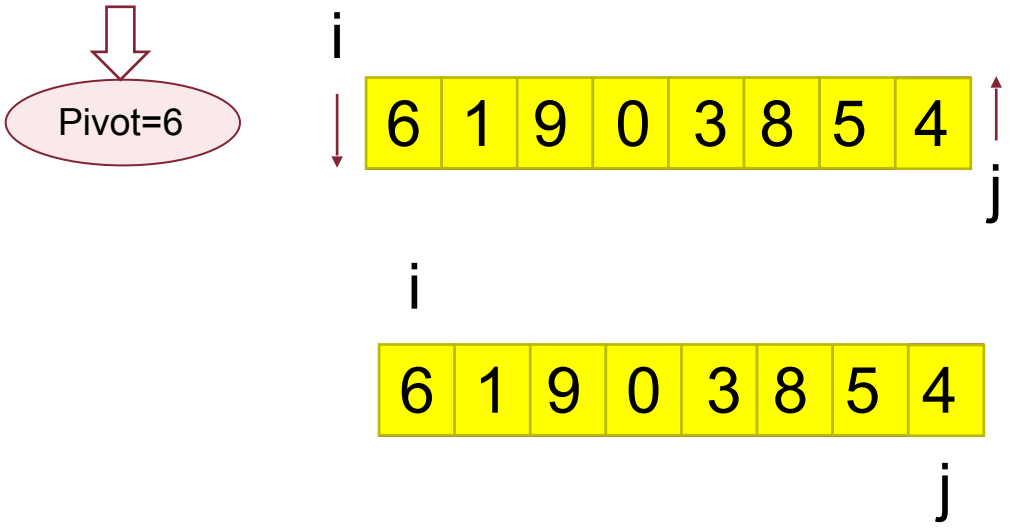
Quicksort

funzionamento di partizione su un esempio



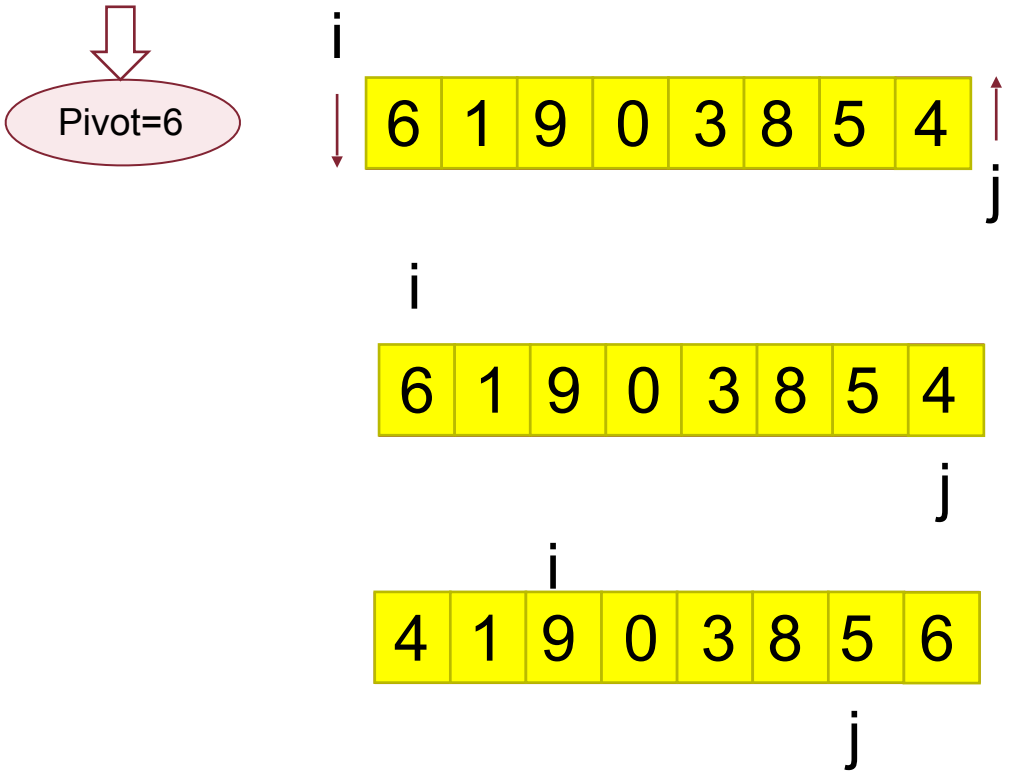
Quicksort

funzionamento di partizione su un esempio



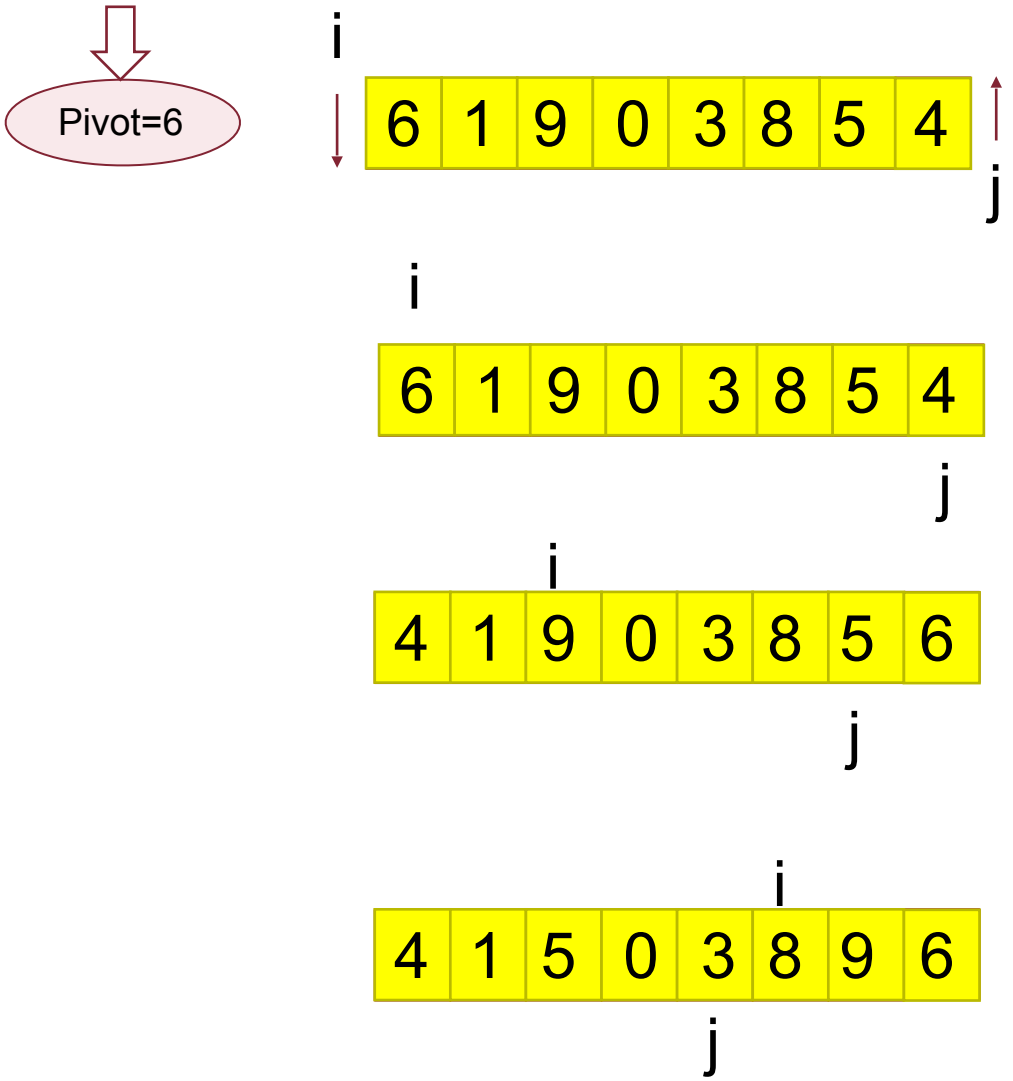
Quicksort

funzionamento di partizione su un esempio



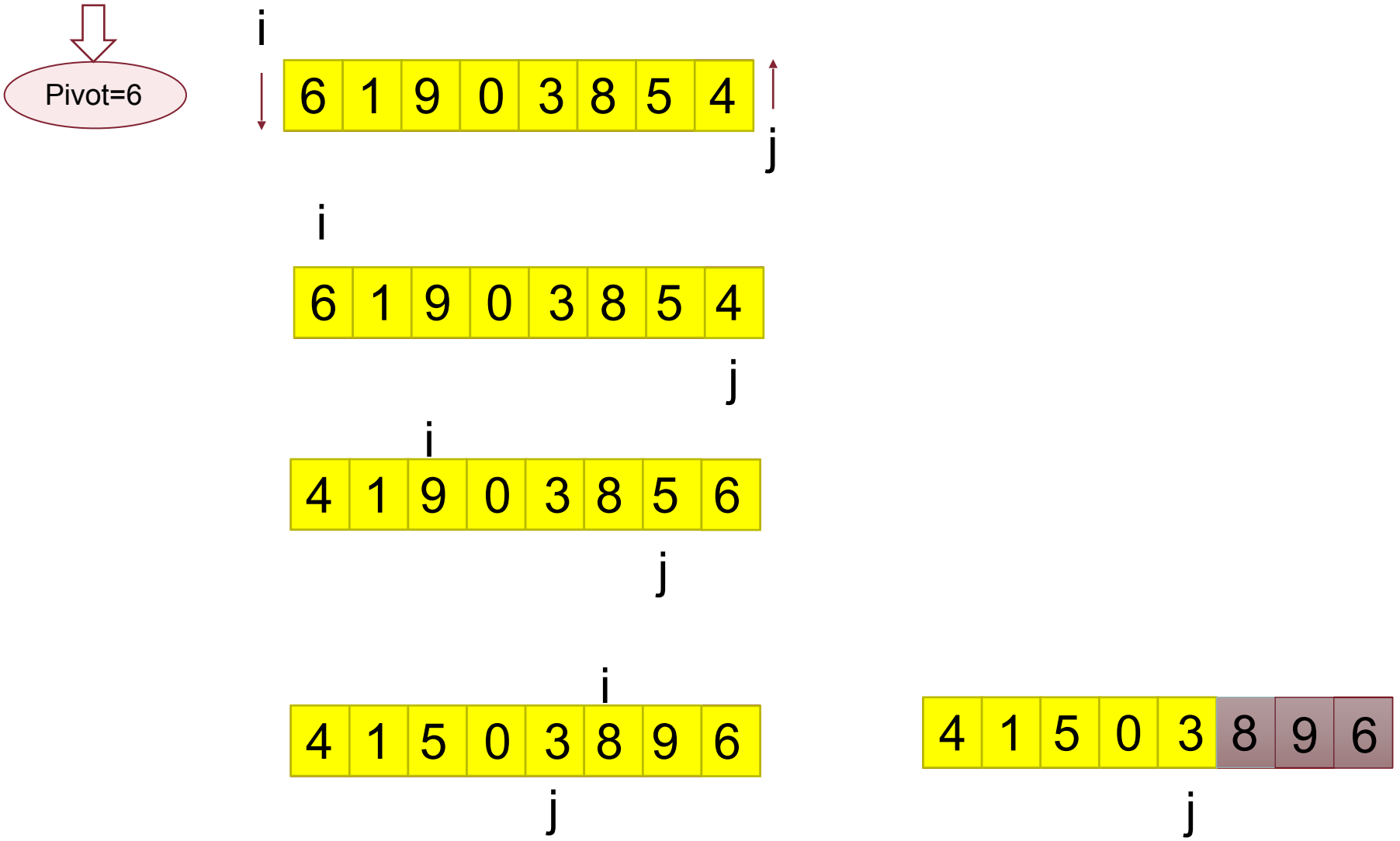
Quicksort

funzionamento di partizione su un esempio



Quicksort

funzionamento di partizione su un esempio



Quicksort

```
def Partiziona (lista, indice_primo, indice_ultimo):
   pivot = lista[indice_primo]          #scelta arbitraria
   i = indice_primo - 1
   j = indice_ultimo + 1
   while True:
         while True:
            j -= 1
             if lista[j] <= pivot :break
         while True:
            i += 1
             if lista[i] >= pivot : break
         if i < j:
             lista[i], lista[j]=lista[j], lista[i]
         else: return j
```

Quicksort

Analizziamo il costo computazionale della funzione `Partiziona()`:

- Le prime tre istruzioni costano $\Theta(1)$.
- Il `while` ha un costo *O(n)* più un costo pari alla somma dei costi di ciò che avviene al suo interno, che è $\Theta(n)$ poiché:
  - ciascuna iterazione di ognuno dei due `while` costa *Θ(1)* e avvicina di una posizione un indice all'altro;
  - quindi complessivamente si effettuano $\Theta(n)$ iterazioni dei due `while`;
  - terminati i due `while` si trova un `if` ed un possibile scambio di elementi, $\Theta(1)$

Dunque il costo di `Partiziona()` è $\Theta(n)$.

NOTA: il valore j restituito da `Partiziona()` sulla lista di n elementi vale:

- 0 se il pivot è più piccolo degli altri elementi;
- $n-2$ se il pivot è più grande degli altri elementi.

Quicksort

Valutiamo ora il costo computazionale del Quicksort:

```
def Quick_sort (A, ind_primo, ind_ultimo):
   if (ind_primo < ind_ultimo):
         ind_medio = Partiziona(A, ind_primo, ind_ultimo)
         Quick _sort (A, ind_primo, ind_medio)
         Quick _sort (A, ind_medio+1, ind_ultimo)
```

$T(n)$

$\Theta(1)$

$\Theta(n)$

$T(k)$

$T(n-k)$

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

Che non sappiamo risolvere con alcuno dei metodi studiati…

Quicksort

Possiamo facilmente derivare la soluzione per due situazioni, il caso migliore e quello peggiore.

- Caso migliore:
  è quello in cui, ad ogni passo, la dimensione dei due sotto-problemi è identica. L'equazione di ricorrenza diventa:

$$T(n) = 2T\left(\frac{n}{2}\right) + \Theta(n) \text{ che ha soluzione } T(n) = \Theta(n \log n)$$

- Caso peggiore:
  è quello in cui, ad ogni passo, la dimensione di uno dei due sotto-problemi da risolvere è 1. L'equazione di ricorrenza diventa:

$$T(n) = T(n-1) + \Theta(n) \text{ che ha soluzione } T(n) = \Theta(n^2)$$

QuickSort

Valutiamo ora il costo computazionale nel caso medio, nell'ipotesi che il valore del pivot suddivida con uguale probabilità $\frac{1}{n-1}$ la sequenza da ordinare in due sotto-sequenze di dimensioni k ed $n-k$, per tutti i valori di k tra 1 ed $n-1$.

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

Ora, per ogni valore di q e $q = 1,2,\ldots,n-1$ il termine $T(q)$compare due volte nella sommatoria, la prima quando $k = q$ e la seconda quando $k = n-q$.

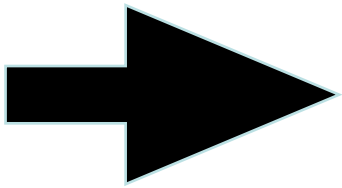
Valutiamo dunque il valore di $T(n) = \frac{2}{n-1}\sum_{q=1}^{n-1} T(q) + \Theta(n)$

Quick Sort

Per risolvere la ricorrenza utilizziamo il metodo di sostituzione, e quindi eliminiamo per prima cosa la notazione asintotica:

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

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



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

$$T(1) = k$$

Ipotizziamo ora la soluzione:

$$T(n) \leq a \cdot n \log n$$

Quicksort

Sostituiamo la soluzione innanzi tutto nel caso base.

Visto che $\log 1 = 0$, non possiamo utilizzare $T(1)$ e sostituiamo quindi in $T(2)$. Vale:

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

deve quindi aversi:

$$\begin{aligned} T(2) &= 2 \cdot k + 2 \cdot h \\ &\leq 2 \cdot a \log 2 = 2 \cdot a \end{aligned}$$

che è vera per a opportunamente grande ($a \geq k + h$).

Quicksort

Per il passo induttivo possiamo scrivere:

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

$$\leq \frac{2}{n-1}\sum_{q=1}^{n-1} a \cdot q \cdot \log\, q + hn$$

$$= \frac{2a}{n-1}\sum_{q=1}^{n-1} q \cdot \log\, q + hn$$

Valutiamo ora la sommatoria $\sum_{q=1}^{n-1} q \log\, q$, che spezziamo in due:

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

$\leq \log \frac{n}{2} = \log n - 1$

$\leq \log n$

Quicksort

Dunque possiamo scrivere:

$$\sum_{q=1}^{n-1} q \cdot \log q \leq \sum_{q=1}^{\lceil \frac{n}{2} \rceil - 1} q \cdot (\log n - 1) + \sum_{q=\lceil \frac{n}{2} \rceil}^{n-1} q \cdot \log n$$

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

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

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

$$\leq \log n \sum_{q=1}^{n-1} q - \sum_{q=1}^{\frac{n}{2}-1} q$$

Nota: l'ultima disuguaglianza è vera perché $\left\lceil \frac{n}{2} \right\rceil - 1 \geq \frac{n}{2} - 1$

Quicksort

Ora,

$$\log n \sum_{q=1}^{n-1} q - \sum_{q=1}^{\frac{n}{2}-1} q = \log n \frac{(n-1)n}{2} - \frac{\left(\frac{n}{2}-1\right)\frac{n}{2}}{2}$$

$$= \frac{1}{2}(n-1)n \log n - \frac{1}{4}\left(\frac{n}{2}-1\right)n$$

$$\leq \frac{1}{2}(n-1)n \log n - \frac{1}{4}\left(\frac{n}{2}-1\right)(n-1)$$

$$= (n-1)\left(\frac{1}{2}n \log n - \frac{n}{8} + \frac{1}{4}\right)$$

Ricapitolando:

$$\sum_{q=1}^{n-1} q \cdot \log q \leq \log n \sum_{q=1}^{n-1} q - \sum_{q=1}^{\frac{n}{2}-1} q \leq (n-1)\left(\frac{1}{2}n \log n - \frac{n}{8} + \frac{1}{4}\right)$$

Quicksort

Sapendo quindi che: $$T(n) \leq \frac{2a}{n-1} \sum_{q=1}^{n-1} q \cdot \log\, q + h \cdot n$$

e che: $$\sum_{q=1}^{n-1} q \cdot \log\, q \leq (n-1)\left(\frac{1}{2} n \log\, n - \frac{n}{8} + \frac{1}{4}\right)$$

possiamo scrivere:

$$\begin{aligned} T(n) &\leq \frac{2a}{n-1}(n-1)\left(\frac{1}{2} n \log\, n - \frac{n}{8} + \frac{1}{4}\right) + h \cdot n \\ &= a \cdot n \cdot \log\, n - \frac{an}{4} + \frac{a}{2} + h \cdot n \\ &\leq a \cdot n \cdot \log\, n \end{aligned}$$

dove l'ultima diseguaglianza segue se prendiamo a sufficientemente grande in modo da avere $h \cdot n - \frac{an}{4} + \frac{a}{2} \leq 0$

Il che ci permette di dimostrare che $T(n) = O(n \log\, n)$.

Quicksort

Abbiamo appena dimostrato che nel caso medio si ha $T(n) = O(n \log n)$.

Inoltre in media non si può avere meglio dell'ottimo che sappiamo essere $\Theta(n \log n)$ da questo deduciamo che per il caso medio vale anche $T(n) = \Omega(n \log n)$.

Possiamo dunque concludere che nel caso medio il Quicksort ha un costo computazionale:

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

Osservazioni

L'analisi ora fatta è valida nell'ipotesi che il valore del pivot sia equiprobabile e, quando questo è il caso, **il quicksort 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. 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.

Corso di laurea in Informatica
Introduzione agli Algoritmi
Didattica blended

Esercizi per casa

SAPIENZA
UNIVERSITÀ DI ROMA

Esercizi - 1

- 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 e poi nel caso in cui sia già ordinato da destra a sinistra.

Esercizi - 2

- Si progetti un algoritmo il più efficiente possibile per i seguenti problemi:

  - Data una matrice *m x n,* si vogliono rimescolare i suoi elementi in modo che tutti i vettori riga e tutti i vettori colonna siano ordinati in senso non decrescente.

  - Data una matrice *n x n,* si vogliono rimescolare i suoi elementi in modo che tutti gli elementi posizionati al di sopra della diagonale principale siano minori o uguali di tutti gli elementi che giacciono sulla diagonale principale che, a loro volta, siano minori o uguali di tutti gli elementi posizionati al di sotto della diagonale principale.