Corso di laurea in Informatica
Introduzione agli Algoritmi
A.A. 2024/25

Il problema dell'ordinamento: Il Quicksort

Tiziana Calamoneri

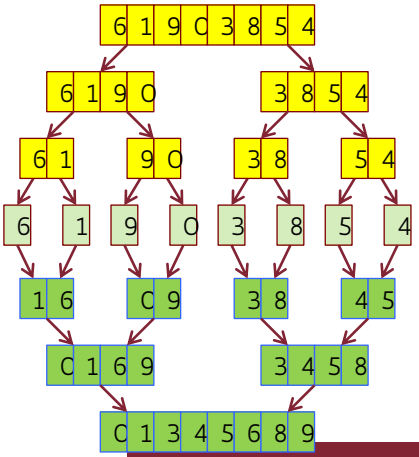
SAPIENZA
UNIVERSITÀ DI ROMA

Slides realizzate sulla base di quelle preparate da T. Calamoneri e G. Bongiovanni per il corso di Informatica Generale tenuto a distanza nell'A.A. 2019/20

Esercizio Risolto - 1

Es. Scrivere la versione iterativa del Merge-Sort.

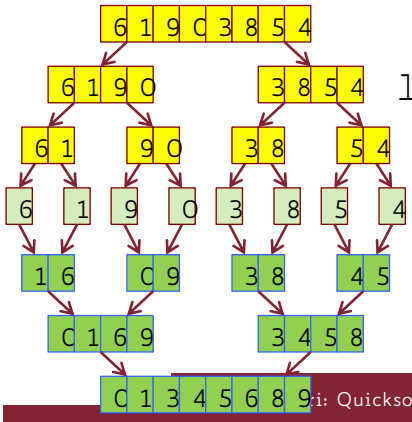
Sol. Sappiamo che è sempre possibile trasformare qualunque algoritmo ricorsivo in un algoritmo iterativo, anche se a volte è meno naturale.



Idea. Possiamo continuare ad usare la funzione Fondi() e basta simulare la parte inferiore nel grafico delle computazioni, visto che la parte superiore mostra semplicemente l'apertura delle chiamate ricorsive.

Esercizio Risolto - 2

```
def mergeSortIter(A)
  lung=1
  while lung<=len(A)//2:
     i=0
     while i<=len(A)-2*lung:
          Fondi(A,i,i+lung,i+2*lung-1)
          i+=2*lung
     lung*=2
```



Quando `lung`=1 vengono fuse le singole celle
Quando `lung`=2 gli array da 2 elementi
Quando `lung`=4 …

Esercizio Risolto - 3

Costo computazionale:

```
def mergeSortIter(A)
  lung=1                                  Θ(1)
  while lung<=len(A)//2:                  log n volte
      i=0                                      Θ(1)
      while i<=len(A)-2*lung:                  n/(2 lung) volte
            Fondi(A,i,i+lung,i+2*lung-1)       Θ(lung)
            i+=2*lung                          Θ(1)
      lung*=2                             Θ(1)
```

$T(n)=\Theta(1)+\log n\ (n/(2\ lung)\ \Theta(lung)) = \log n\ \Theta(n)=$
$=\Theta(n \log n)$

Quicksort - 1

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".

Quicksort - 2

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.

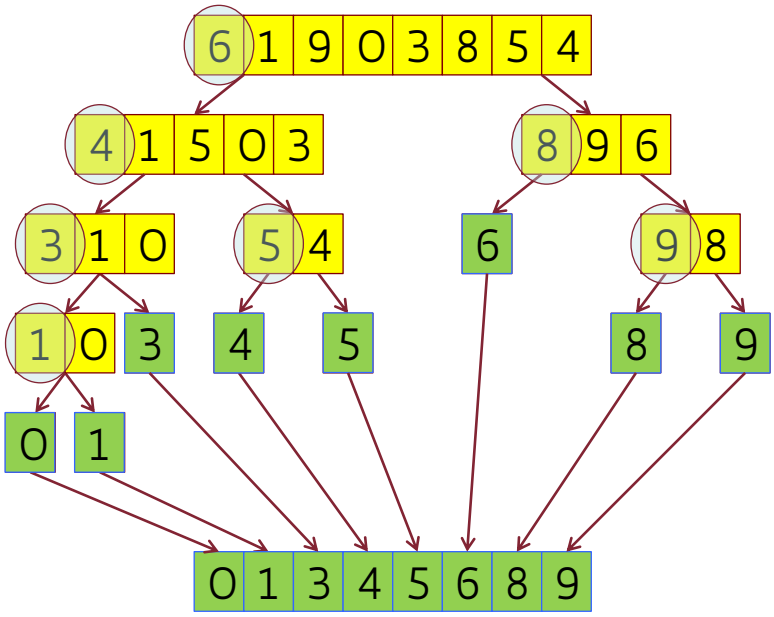
Anche l'algoritmo **Quicksort** è un algoritmo ricorsivo che adotta una tecnica algoritmica detta **divide et impera**:

...

Quicksort - 3

- *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;
- *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.

Quicksort - 4



•divide: seleziona un pivot. La sequenza è divisa in due sottosequenze: elementi minori o uguali del pivot, e elementi maggiori o uguali del pivot
•passo base: la ricorsione procede fino a quando le sottosequenze sono di un solo elemento
•impera: le due sotto-sequenze vengono ordinate ricorsivamente
•combina: non occorre

Quicksort - 5

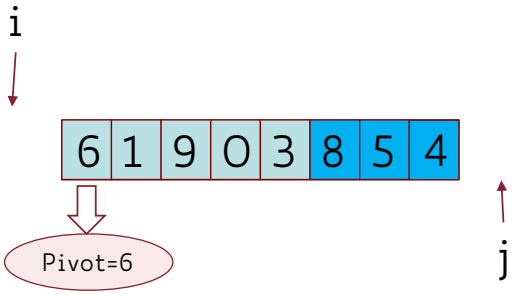
Lo pseudocodice del Quicksort è il seguente:

```
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)
```

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 è *sempre* compreso fra 1 ed (n-1).

Quicksort - 6

Prima di vedere il codice della funzione Partiziona vediamone il funzionamento su un esempio.



Quicksort - 7

```
Funzione Partiziona (A: array; ind_primo, ind_ultimo: intero)
   pivot ← A[indice_primo]          //scelta arbitraria
   i ← indice_primo - 1;
   j ← indice_ultimo + 1
   while true
      repeat
         j ← j - 1
      until A[j] ≤ pivot
      repeat
         i ← i + 1
      until A[i] ≥ pivot
      if (i < j) scambia A[i] e A[j]
      else return j
```

(in effetti, ad esempio, nelle dispense, si procede in modo diverso, prendendo l'ultimo elemento come pivot… Anche questo codice è un po' diverso, ma la filosofia è la stessa…)

NOTA: In ogni caso, il valore j restituito da `Partiziona()` deve valere 1 se il pivot è più piccolo degli altri elementi, ed (n-1) se il pivot è più grande degli altri elementi.

Quicksort - 8

```
Funzione Partiziona (A: array; ind_primo, ind_ultimo: intero)
   pivot ← A[indice_primo]
   //scelta arbitraria
   i ← indice_primo - 1;
   j ← indice_ultimo + 1
   while true
      repeat
         j ← j - 1
      until A[j] ≤ pivot
      repeat
         i ← i + 1
      until A[i] ≥ pivot
      if (i < j) scambia A[i] e A[j]
      else return j
```

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…

Quicksort - 9

Il costo del `while` è Θ(n) poiché:

- ciascuna iterazione di ognuno dei due `repeat` costa Θ(1) e avvicina di una posizione un indice all'altro;
- quindi complessivamente si effettuano Θ(n) iterazioni dei due `repeat`;
- terminati i due `repeat` si trova un `if` ed un possibile scambio di elementi, Θ(1)

```
Funzione Partiziona (A: array;
ind_primo, ind_ultimo: intero)
   pivot ← A[indice_primo]
   //scelta arbitraria
   i ← indice_primo - 1;
   j ← indice_ultimo + 1
   while true
      repeat
         j ← j - 1
      until A[j] ≤ pivot
      repeat
         i ← i + 1
      until A[i] ≥ pivot
      if (i < j) scambia A[i] e A[j]
      else return j
```

Dunque il costo di `Partiziona()` è Θ(n).

Quicksort - 10

Valutiamo ora il costo computazionale del Quicksort:

```
Funzione Quick_sort (A, ind_pr, ind_ult)
  if (ind_pr < ind_ult):                          Θ(1)
        ind_med=Partiziona(A,ind_pr,ind_ult)      Θ(n)
        Quick _sort (A,ind_pr,ind_med)            T(k)
        Quick _sort (A, ind_med+1,ind_ult)        T(n-k)
```

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

non sappiamo risolverla con i metodi iterativo, dell'albero e principale…

Quicksort - 11

L'unica speranza è il metodo di sostituzione, per il quale dobbiamo però ipotizzare una soluzione.

Per farci un'idea, deriviamo la soluzione per dei casi speciali di k che possiamo pensare come migliore e peggiore - caso base fissato a $T(1)=\Theta(1)$.

Quicksort - 12

Caso che potrebbe sembrare favorevole:
ad ogni passo, la dimensione dei due sotto-problemi è identica. L'equazione di ricorrenza diventa:

$T(n)=2T(n/2)+\Theta(n)$ che ha soluzione $T(n)=\Theta(n \log n)$

Caso che potrebbe sembrare sfavorevole:
ad ogni passo, la dimensione di uno dei due sotto-problemi da risolvere è 1. L'equazione di ricorrenza diventa:

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

Poiché le soluzioni di due casi diversi non coincidono...

Quicksort - 13

...valutiamo il costo computazionale nel caso medio, nell'ipotesi che il pivot suddivida con uguale probabilità 1/(n-1) la sequenza in due sotto-sequenze di dim. k ed n-k, per tutti i valori di k tra 1 ed n-1.

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

Per ogni valore di k = 1, 2, ..., n - 1 il termine T(k) compare due volte nella sommatoria, la prima quando k = i e la seconda quando k = n - i. Valutiamo dunque il valore di:

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

Quicksort - 14

Utilizziamo il metodo di sostituzione, e quindi eliminiamo innanzi tutto la notazione asintotica:

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

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

Per ragioni che saranno chiare tra breve, calcoliamo:

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

Ipotizziamo ora la soluzione:

$$T(n) \leq an \log n$$

Quicksort - 15

Sostituiamo la soluzione innanzi tutto nel caso base.

Visto che log 1 = 0, non possiamo utilizzare T(1) e sostituiamo quindi in T(2), ottenendo:

$$T(2) = 2k + 2h \leq 2a \log 2 = 2a$$

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

Quicksort - 16

Per il passo induttivo possiamo scrivere:

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

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

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

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

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

Quicksort - 17

Dunque possiamo scrivere:

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

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

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

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

Quicksort - 18

Ora:

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

$$= \log n \frac{(n-1)n}{2} - \frac{1}{2}\left(\frac{n}{2} - 1\right)\frac{n}{2} = \text{(perché } \left\lceil \frac{n}{2} \right\rceil \geq \frac{n}{2}\text{)}$$

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

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

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

Ricapitolando:

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

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

Quicksort - 19

Sapendo quindi che:

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

e che:

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

possiamo scrivere:

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

$$= an\ \log\ n - \frac{an}{4} + \frac{a}{2} + hn$$

Quicksort - 20

Scegliendo *a* sufficientemente grande si ha che:

$hn - \frac{an}{4} + \frac{a}{2} \leq 0$ e per tale a si avrà:

$$T(n) \leq an\ \log\ n - \frac{an}{4} + \frac{a}{2} + hn \leq an\ \log\ n$$

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

Analogamente possiamo dimostrare che
$T(n) = \Omega(n\ \log\ n)$, per cui abbiamo che nel caso medio il Quicksort ha un costo computazionale:

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

Quicksort - 21

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.

...

Quicksort - 22

...

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.

...

Quicksort - 23

…

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
A.A. 2024/25

Esercizi per casa

SAPIENZA
UNIVERSITÀ DI ROMA

Esercizi - 1

- Sia dato un array di lunghezza *n* contenente solo valori 0 e 2. Si progetti un algoritmo con costo computazionale lineare che modifichi l'array 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 QuickSort.

Esercizi - 2

- Calcolare il costo computazionale del QuickSort nel caso in cui l'array contenga tutti elementi uguali ed in cui sia già ordinato da destra a sinistra.

- Si progetti un algoritmo il più efficiente possibile per il seguente problema:

  Data una matrice m x n, si vogliono rimescolare i suoi elementi in modo che tutte le righe e tutte le colonne siano ordinate in modo non decrescente.