Università di Napoli Federico II – Scuola Politecnica e delle Scienze di Base
Corso di Laurea Magistrale in Ingegneria Informatica

Corso di Algoritmi e Strutture Dati

Quicksort

Quicksort

- **Quick Sort** ha un tempo di esecuzione
  - $\Theta(n^2)$ nel caso peggiore
  - $\Theta(n \lg n)$ nel caso “medio”
  - Ordina sul posto

- Nella pratica, risulta essere tra i più efficienti, grazie al fatto che i fattori “nascosti” in $\Theta(n \lg n)$ sono piuttosto piccoli

Quicksort

- Utilizza un approccio divide et impera
- **Divide**
  - Scelto un elemento x, detto *pivot*, partiziona l'array A[p..r] in due sottoarray tali che A[p..q-1] contiene gli elementi <= di x e A[q+1..r] contiene gli elementi >x; il pivot va in A[q]
- **Conquer**
  - Ordina i due sottoarray mediante chiamate ricorsive
- **Combine**
  - I sottoarray sono ordinati sul posto, quindi non occorre fare nulla

Quicksort

```
Quicksort (A,p,r)
if p<r
    q ← Partition (A,p,r)
    Quicksort (A,p,q-1)
    Quicksort (A,q+1,r)
```

q-1 e q+1 perchè l’elemento alla posizione q già è ordinato!

Per ordinare l'intero array si invoca Quicksort (A,1,length[A])

Partition

- La funzione `Partition` utilizza come pivot l'ultimo elemento del sottoarray (x=A[r])
- Analizza, uno ad uno e a partire dal primo, tutti gli elementi compresi tra p e r-1
- …suddividendo il vettore in quattro aree:
  - Gli elementi <= del pivot [nelle posizioni p..i]
  - Gli elementi > del pivot [nelle posizioni i+1..j-1]
  - Gli elementi non ancora analizzati [j..r-1]
  - Il pivot [r]

p	i			j			r
2	1	7	8	3	5	6	4

Partition

p	i			j			r
2	1	7	8	3	5	6	4

- L'elemento di posto j (il primo non analizzato) viene confrontato con il pivot
  - Se è maggiore, è sufficiente incrementare la barriera tra 2ª e 3ª area, ovvero incrementare j
  - Se è minore o uguale, deve essere spostato nella prima area
    - Si incrementa i (si sposta barriera tra 1ª e 2ª area)
    - Si scambia A[i] con A[j]
    - Si incrementa j (si sposta barriera tra 2ª e 3ª area)

Partition

i p,j r

| 2 8 7 1 3 5 6 | 4

p,j,i r

| 2 | 8 7 1 3 5 6 | 4

- Inizialmente, due barriere (quella tra 1ª e 2ª area e tra 2ª e 3ª area) coincidono
- Si confronta il primo elemento con il pivot
- E' minore o uguale
  - Si incrementa i (si sposta barriera tra 1ª e 2ª area)
  - Si scambia A[i] con A[j]
  - Si incrementa j (si sposta barriera tra 2ª e 3ª area)

Partition

p,i j r
2 | 8 7 1 3 5 6 | 4

p,i j r
2 | 8 | 7 1 3 5 6 | 4

p,i j r
2 | 8 7 | 1 3 5 6 | 4

- Il secondo elemento è maggiore del pivot
  - Si incrementa j (si sposta barriera tra 2ª e 3ª area)

- Il terzo elemento è maggiore del pivot
  - Si incrementa j (si sposta barriera tra 2ª e 3ª area)

Partition

p,i   j   r
2 | 8 7 | 1 3 5 6 | 4

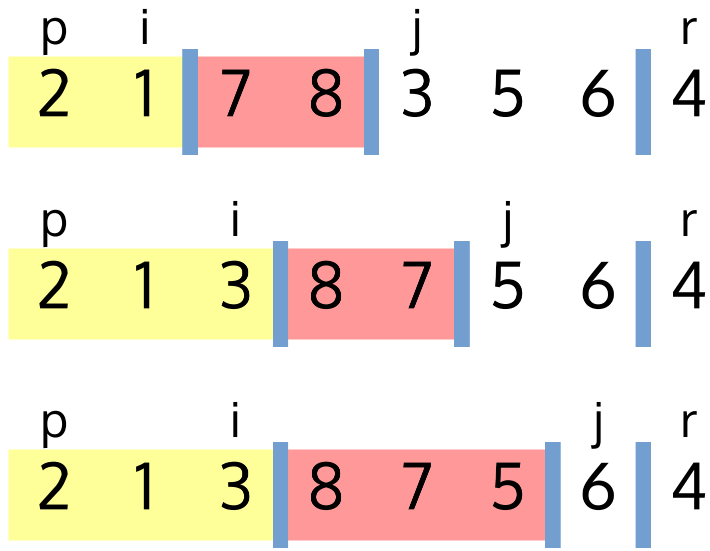
p i   j   r
2 8 | 7 | 1 3 5 6 | 4

p i   j   r
2 1 | 7 | 8 3 5 6 | 4

p i   j   r
2 1 | 7 8 | 3 5 6 | 4

- Il quarto elemento è minore o uguale del pivot
  - Si incrementa i (si sposta barriera tra 1ª e 2ª area)
  - Si scambia A[i] con A[j]
  - Si incrementa j (si sposta barriera tra 2ª e 3ª area)

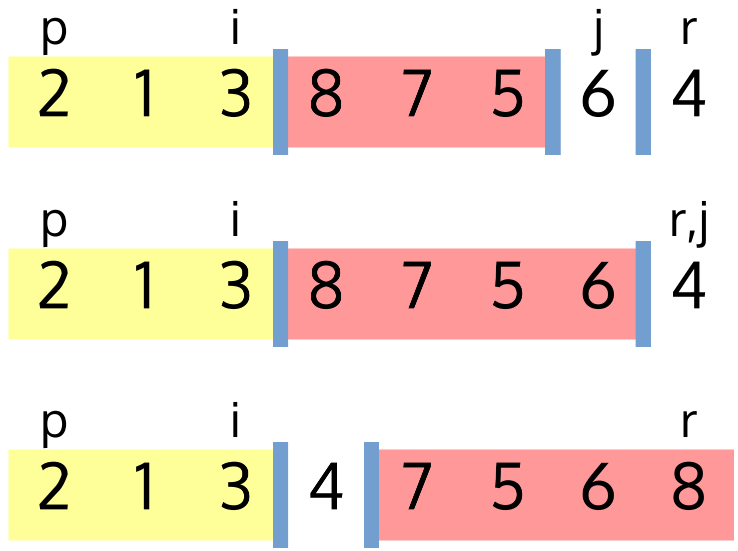
Partition



- Il quinto elemento è minore o uguale del pivot
  - Si incrementa i (si sposta barriera tra 1ª e 2ª area)
  - Si scambia A[i] con A[j]
  - Si incrementa j (si sposta barriera tra 2ª e 3ª area)

- Il sesto elemento è maggiore del pivot
  - Si incrementa j (si sposta barriera tra 2ª e 3ª area)

Partition



- Il settimo elemento è maggiore del pivot
  - Si incrementa j (si sposta barriera tra 2ª e 3ª area)
- Il ciclo è finito
  - Si scambia A[i+1] con A[r]

Partition

p		i				j	r
2	1	3	8	7	5	6	4

```
Partition (A,p,r)
x ← A[r]
i ← p-1
for j ← p to r-1
    if A[j]≤x
        i ← i+1
        exchange A[i]↔A[j]
exchange A[i+1] ↔ A[r]
return i+1
```

- Il tempo di esecuzione è Θ(n), con n=r-p+1
  - n iterazioni, ciascuna delle quali impiega un tempo costante

Partition

p		i				j	r
2	1	3	8	7	5	6	4

- Invariante
- All'inizio di ogni iterazione del ciclo for
  - $A[k] \leq x$ per $p \leq k \leq i$
  - $A[k]>x$ per $i+1 \leq k \leq j-1$
  - $A[k]=x$ se $k=r$
- Vero prima della prima iterazione ($i=p-1$ e $j=p$) perché la 1ª e la 2ª area sono entrambe vuote
- Il ciclo conserva l'invariante
- Al termine ($j=r$), l'array è suddiviso in tre insiemi
- Elementi minori o uguali al pivot, pivot, elementi maggiori del pivot

Quicksort: worst-case partitioning

- Il caso peggiore si presenta quando il partizionamento produce un sottoproblema con n-1 elementi e uno con 0 elementi
  - Dimostrazione in seguito
- Supponendo che un tale partizionamento sbilanciato si verifichi in ogni iterazione, il tempo di esecuzione è
  - $T(n) = T(n-1) + T(0) + \Theta(n) = T(n-1) + \Theta(n)$
- Intuitivamente, la soluzione è $T(n)=\Theta(n^2)$
  - Si ottiene una serie aritmetica
  - Verificabile con il metodo di sostituzione
- Il tempo di esecuzione di Quicksort nel caso peggiore non è migliore di quello di Insertion Sort
  - Il caso peggiore di Quicksort si ha quando il vettore è ordinato
  - ... caso in cui Insertion Sort richiede un tempo $O(n)$

Quicksort: best-case partitioning

- Il caso migliore è quando i due sottoproblemi sono perfettamente bilanciati ad ogni iterazione
  - Dimensioni $\lfloor n/2 \rfloor$ e $\lceil n/2 \rceil-1$

- Tempo di esecuzione $T(n) \leq 2T(n/2) + \Theta(n)$

- Soluzione: $T(n)=O(n\lg n)$
  - Dal caso 2 del teorema dell'esperto

Quicksort: balanced partitioning

- Il tempo medio di esecuzione di Quicksort è più vicino al caso migliore che non al caso peggiore

- Supponiamo che il partizionamento produce due sottoproblemi la cui dimensione è sempre nel rapporto 9:1

- $T(n) \leq T(9n/10)+T(n/10)+cn$

- Utilizziamo il metodo dell'albero di ricorrenza

Quicksort: balanced partitioning

n — cn

$\frac{1}{10}n$ $\quad \frac{9}{10}n$ — cn

$\log_{10} n$

$\frac{1}{100}n$ $\quad \frac{9}{100}n$ $\quad \frac{9}{100}n$ $\quad \frac{81}{100}n$ — cn

$\log_{10/9} n$

1 $\quad \frac{81}{1000}n$ $\quad \frac{729}{1000}n$ — cn

$\leq cn$

1 — $\leq cn$

- Il costo è O(nlg n)
  - Θ(lg n) livelli, ciascuno di costo al più cn
- Qualunque partizionamento con rapporto *costante* (anche 99:1) produce un costo O(nlg n) perché la profondità è sempre Θ(lg n)

Quicksort: average case

- Nel caso medio, tipicamente partizioni “buone” e “cattive” si alternano
- Intuitivamente, se si alternano best case e worst case, il tempo di esecuzione complessivo è quello del best case

n

$\Theta(n)$

n

0 $n-1$

$\Theta(n)$

$\Theta(n-1)$

$\frac{n-1}{2}$ $\frac{n-1}{2}$

$\frac{n-1}{2} - 1$ $\frac{n-1}{2}$

- Il costo $\Theta(n-1)$ della partizione cattiva viene assorbito nel costo $\Theta(n)$ della partizione buona, e la partizione risultante è buona

Randomizzazione

Randomizzazione

- *«Randomization is a powerful tool to improve algorithms with bad worst-case but good average-case complexity*
  - Random sampling
  - Randomized hashing
  - Random search

Quicksort randomizzato

- Per aumentare la probabilità che in media il partizionamento effettuato da Quicksort sia ben bilanciato, si può scegliere in maniera aleatoria il pivot
- La versione randomizzata di Quicksort è ritenuta la scelta migliore per ordinare array di grosse dimensioni

```
Randomized-Partition (A,p,r)
i ← Random (p,r)
exchange A[r] ↔ A[i]
return Partition (A,p,r)
```

```
Randomized-Quicksort (A,p,r)
if p<r
     q ← Randomized-Partition (A,p,r)
    Randomized-Quicksort (A,p,q-1)
    Randomized-Quicksort (A,q+1,r)
```

Analisi del worst case

- Dimostriamo che il tempo di esecuzione è $\Theta(n^2)$ nel caso peggiore
  - Dimostrando prima con il metodo di sostituzione che $T(n)=O(n^2)$ nel caso peggiore
    - $T(n)=\max_{0\leq q\leq n-1} (T(q)+T(n-q-1))+\Theta(n)$
    - $\leq \max_{0\leq q\leq n-1} (cq^2 + c(n-q-1)^2)+\Theta(n)$
    - $= c\cdot \max_{0\leq q\leq n-1} (q^2 + (n-q-1)^2)+\Theta(n)$
    - $\leq c(n-1)^2+\Theta(n)$
    - $= cn^2 – c(2n-1)+\Theta(n)$
    - $\leq cn^2$

La funzione ha la concavità verso l'alto
(la derivata seconda rispetto a q è positiva),
quindi il massimo è assunto agli estremi

Analisi del worst case

- In precedenza abbiamo visto un caso (il caso sbilanciato) in cui il tempo di esecuzione è $\Theta(n^2)$
- Quindi il tempo di esecuzione nel caso peggiore è $\Omega(n^2)$
- Combinando questo risultato con quello della slide precedente, si ha che il tempo di esecuzione nel caso peggiore è $\Theta(n^2)$

Tempo di esecuzione atteso

- Nella versione randomizzata, è di interesse il tempo di esecuzione atteso.
- Si dimostra che è O($n$log$n$)

Quick-sort random tempo atteso:La complessità dipende dal numero di confronti effettuati dal pivot