Algoritmi e Strutture di Dati

Quick-sort

m.patrignani

1

Nota di copyright

- queste slides sono protette dalle leggi sul copyright
- il titolo ed il copyright relativi alle slides (inclusi, ma non limitatamente, immagini, foto, animazioni, video, audio, musica e testo) sono di proprietà degli autori indicati sulla prima pagina
- le slides possono essere riprodotte ed utilizzate liberamente, non a fini di lucro, da università e scuole pubbliche e da istituti pubblici di ricerca
- ogni altro uso o riproduzione è vietata, se non esplicitamente autorizzata per iscritto, a priori, da parte degli autori
- gli autori non si assumono nessuna responsabilità per il contenuto delle slides, che sono comunque soggette a cambiamento
- questa nota di copyright non deve essere mai rimossa e deve essere riportata anche in casi di uso parziale

2

Quick-sort

- Algoritmo di ordinamento in loco ma non stabile
- Tempo di esecuzione
  - nel caso peggiore $\Theta(n^2)$
  - nel caso migliore e medio $\Theta(n \log n)$
    - i fattori costanti nascosti nella notazione Θ sono abbastanza piccoli
- Introdotto da Hoare nel 1962
  - la versione che vedremo è una variante dovuta a Lomuto
- Basato sul paradigma divide et impera

3

Divide et impera nel quick-sort

Per ordinare un sottoarray A[p...r]

- Divide
  - A[p...r] viene ripartito (e risistemato) in due sottoarray non vuoti A[p...q–1] e A[q+1...r], in modo che ogni elemento del primo sia minore o uguale ad A[q] e ogni elemento del secondo sia maggiore ad A[q]
  - l'indice q viene calcolato dalla procedura di partizionamento
- Impera
  - i due sottoarray A[p...q–1] e A[q+1...r] sono ordinati, ricorsivamente
- Combina
  - non c'è niente da fare: A[p...r] è ordinato

4

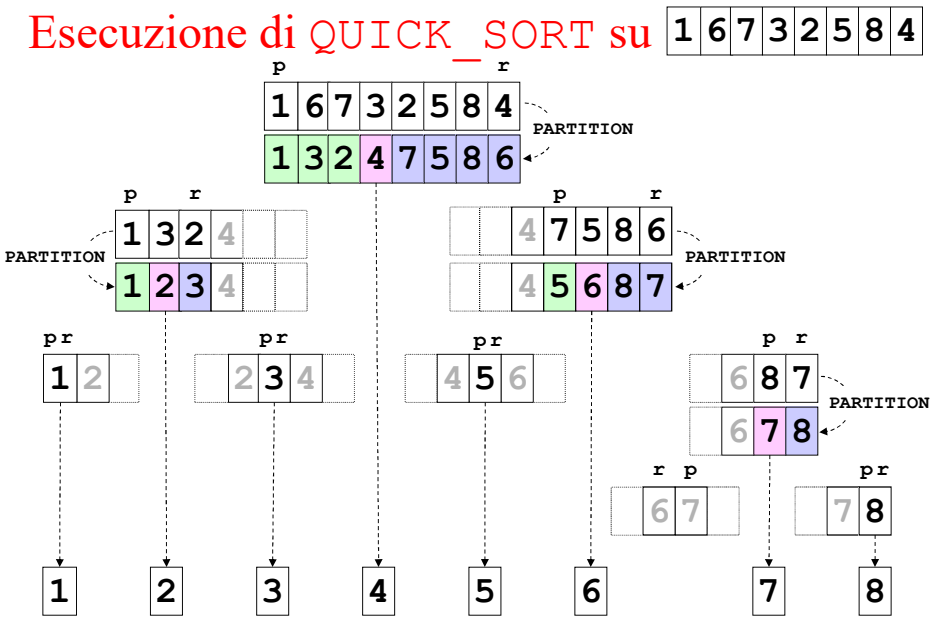
Procedura QUICK_SORT

```
QUICK_SORT(A,p,r)
1.  if p < r
2.      q = PARTITION(A,p,r)
3.      QUICK_SORT(A,p,q-1)
4.      QUICK_SORT(A,q+1,r)
```

- La procedura QUICK_SORT ordina in loco l'intervallo A[*p*..*r*]
  - se $p = r$, allora l'intervallo contiene una sola casella ed è già ordinato: l'invocazione di QUICK_SORT non ha effetto
  - se $p > r$, allora l'intervallo è un intervallo degenere e l'invocazione di QUICK_SORT non ha effetto
- Il valore *q* ritornato da PARTITION è tale che $p \le q \le r$
- Per ordinare l'intero array viene invocata la procedura:
  QUICK_SORT(A,0,A.length-1)

5

Esecuzione di QUICK_SORT su



6

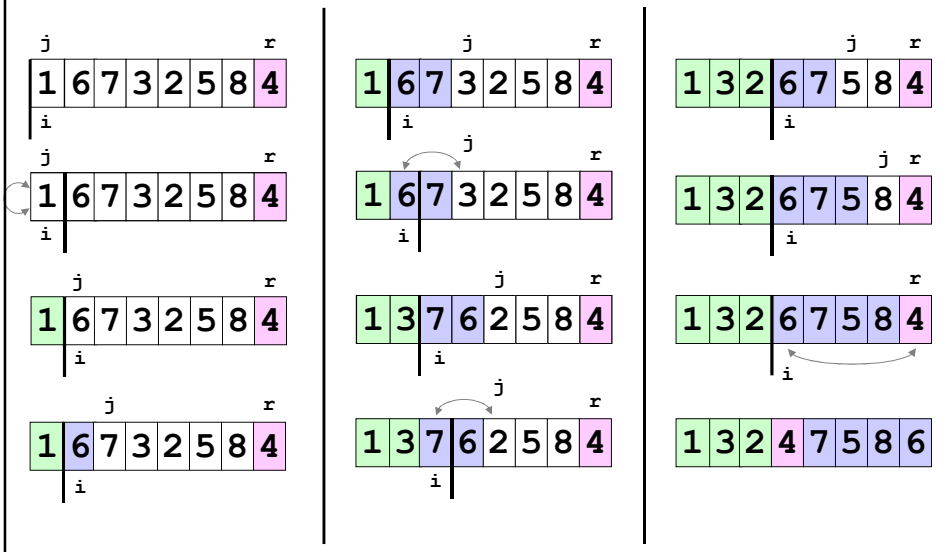
Procedura PARTITION

```
PARTITION(A,p,r)     /* si assume p < r */
1.  i = p      /* i è il primo elemento > A[r] = pivot */
2.    for j = p to r - 1  /* scorro l'array (non il pivot)*/
3.        if A[j] ≤ A[r]   /* A[r] è il pivot */
4.            SCAMBIA(A,i,j)
5.            i = i + 1
6.  SCAMBIA(A,i,r)      /* metto il pivot al centro */
7.  return i      /* ritorno la posizione del pivot */
```

- La procedura PARTITION viene invocata su un intervallo di almeno due elementi ($p < r$)
  - due casi base
    - i due elementi sono ordinati
    - i due elementi non sono ordinati

7

Esecuzione di PARTITION su 1 6 7 3 2 5 8 4

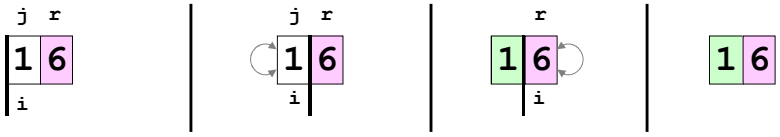


8

Esecuzione di PARTITION su 1 6

- Caso base 1: PARTITION su una coppia ordinata

```
PARTITION(A,p,r)     /* si assume p < r */
1.  i = p      /* i è il primo elemento > A[r] = pivot */
2.    for j = p to r - 1  /* scorro l'array (non il pivot)*/
3.        if A[j] ≤ A[r]    /* A[r] è il pivot */
4.            SCAMBIA(A,i,j)
5.            i = i + 1
6.  SCAMBIA(A,i,r)      /* metto il pivot al centro */
7.  return i      /* ritorno la posizione del pivot */
```

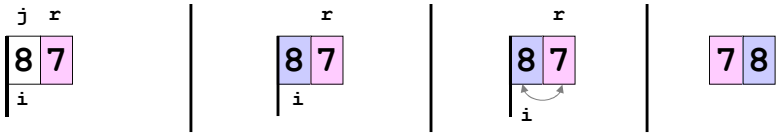


9

Esecuzione di PARTITION su 8 7

- Caso base 2: PARTITION su una coppia non ordinata

```
PARTITION(A,p,r)     /* si assume p < r */
1.  i = p      /* i è il primo elemento > A[r] = pivot */
2.    for j = p to r - 1  /* scorro l'array (non il pivot)*/
3.        if A[j] ≤ A[r]    /* A[r] è il pivot */
4.            SCAMBIA(A,i,j)
5.            i = i + 1
6.  SCAMBIA(A,i,r)      /* metto il pivot al centro */
7.  return i      /* ritorno la posizione del pivot */
```



10

Tempo di esecuzione di PARTITION

```
PARTITION(A,p,r)     /* si assume p < r */
1.  i = p      /* i è il primo elemento > A[r] = pivot */
2.    for j = p to r - 1   /* scorro l'array (non il pivot)*/
3.        if A[j] ≤ A[r]    /* A[r] è il pivot */
4.            SCAMBIA(A,i,j)
5.            i = i + 1
6.  SCAMBIA(A,i,r)      /* metto il pivot al centro */
7.  return i      /* ritorno la posizione del pivot */
```

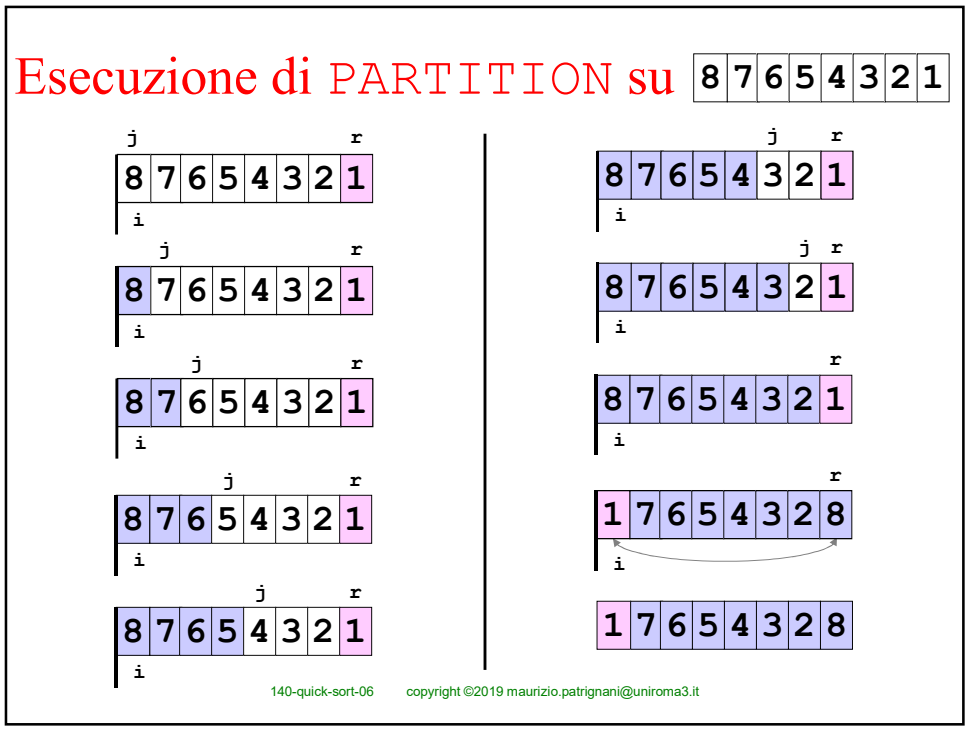
- Le assegnazioni iniziali e finali richiedono tempo costante
- Nel caso peggiore, come nel caso migliore, il sottoarray A[p...r] viene scorso per intero da sinistra verso destra
- Il tempo di esecuzione $T_{\text{PARTITION}}(n) \in \Theta(n)$

11

Esercizi

1. Che cosa succederebbe nel QUICK_SORT se PARTITION(A,*p*,*r*) restituisse un valore *q* uguale a *r*?
2. Illustrare le operazioni di PARTITION sull'array A = <13, 19, 9, 5, 12, 8, 7, 4, 11, 2, 6, 21>
3. Illustrare le operazioni di PARTITION su un array
   - già ordinato in senso decrescente
   - già ordinato in senso crescente
4. Quale valore restituisce PARTITION se tutti gli elementi dell'array A[*p*...*r*] hanno lo stesso valore?

12


Esecuzione di PARTITION su 8 7 6 5 4 3 2 1
j r
8 7 6 5 4 3 2 1
i
8 7 6 5 4 3 2 1
8 7 6 5 4 3 2 1
8 7 6 5 4 3 2 1
8 7 6 5 4 3 2 1
8 7 6 5 4 3 2 1
1 7 6 5 4 3 2 8
1 7 6 5 4 3 2 8
140-quick-sort-06
copyright ©2019 maurizio.patrignani@uniroma3.it

13


Esecuzione di PARTITION su 1 2 3 4 5 6 7 8
j r
1 2 3 4 5 6 7 8
i
140-quick-sort-06
copyright ©2019 maurizio.patrignani@uniroma3.it

14

Caso peggiore e migliore per QUICK_SORT

- Il caso peggiore per QUICK_SORT è quando PARTITION elegge a *pivot* il valore massimo o minimo dell'array
  - in questo caso QUICK_SORT non ricorre su due sottoarray bilanciati, ma ricorre su un sottoarray più corto di una casella ed un sottoarray degenere
- Il caso migliore per QUICK_SORT è invece quando PARTITION elegge a *pivot* il valore mediano dell'array
  - in questo caso QUICK_SORT ricorre su due sottoarray bilanciati

15

Analisi del caso migliore per QUICK_SORT

- Nel caso migliore il tempo di calcolo di QUICK_SORT su un array con n posizioni è

$$T(n) = 2 \cdot T(n/2) + \Theta(n)$$

- Questa equazione di ricorrenza può essere risolta con il teorema dell'esperto

$$T(n) = a \cdot T(n/b) + p(n^k)$$

- Nello speciale caso in cui

$$a=2 \qquad b=2 \qquad k=1$$

- Che per $a = b^k$ si risolve in

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

16

Analisi del caso peggiore per QUICK_SORT

- Si ha

  $T(0) = a$

  $T(n) = T(n\text{-}1) + \Theta(n)$

- Sappiamo che la soluzione di questa equazione di riccorrenza è

  $$T(n) = a + \sum_{k=1}^{n} g(k)$$

- E dunque

  $$T(n) = \sum_{k=1}^{n} \Theta(k) = \Theta\left(\sum_{k=1}^{n} k\right) = \Theta(n^2)$$

17

Analisi del caso medio per QUICK_SORT

- Si può dimostrare formalmente che nel caso medio QUICK_SORT ha una complessità $\Theta(n \log n)$
  - l'analisi, però, è molto più complessa del caso migliore e del caso peggiore
- Nel seguito vedremo solamente due considerazioni intuitive che ci aiutano a giustificare questo risultato
  1. qual è la complessità nel caso in cui lo sbilanciamento della ricorsione non supera mai una determinata soglia
  2. qual è la complessità nel caso in cui ricorsioni sbilanciate si alternano a ricorsioni più bilanciate

18

Caso bilanciato 9-a-1

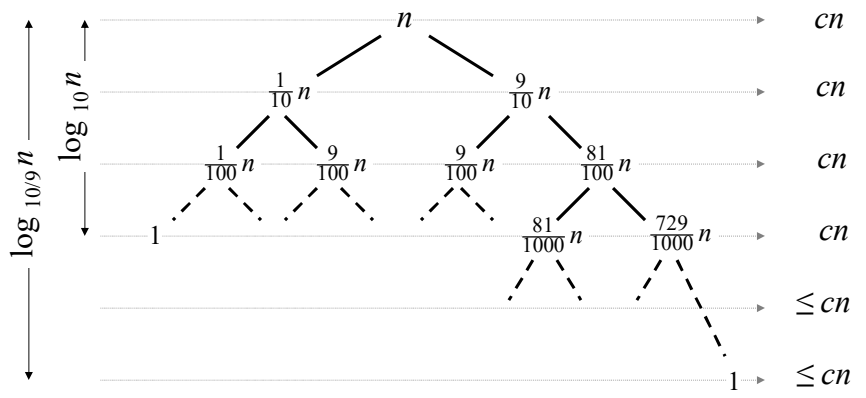
- Supponiamo che `PARTITION` divida il sottoarray in due parti che hanno una proporzione fissa
  - supponiamo che la proporzione sia 9-a-1
- Abbiamo

$$T(n) \le T(9n/10) + T(n/10) + cn$$

dove cn esplicita $\Theta(n)$

19

Ricorsione con proporzione 9-a-1



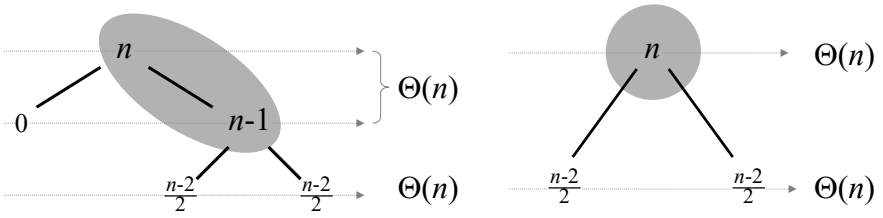
- Ciò fa presumere che il costo nel caso medio sia molto vicino al caso migliore

$O(n \lg n)$

20

Alternanza di ricorsioni bilanciate e sbilanciate

- Supponiamo che nel 20% dei casi `PARTITION` produca una partizione meno bilanciata di 9-a-1
- Supponiamo che nell'albero delle chiamate ricorsive una ripartizione sbilanciata sia sempre seguita da una bilanciata
- Il costo di una ripartizione sbilanciata può essere assorbito dal costo della ripartizione bilanciata



21

Versione randomizzata di `QUICK_SORT`

- E' possibile modificare `QUICK_SORT` in maniera che i casi peggiori non coincidano con disposizioni notevoli degli elementi

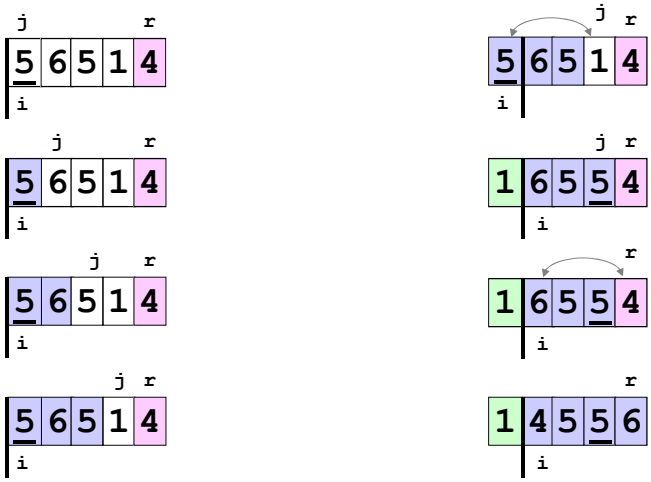
```
RANDOMIZED_PARTITION(A,p,r)
1.  i = RANDOM(p,r)
2.  SCAMBIA(A,r,i)
3.  return PARTITION(A,p,r)
```

```
RANDOMIZED_QUICK_SORT(A,p,r)
1.  if p < r then
2.      q = RANDOMIZED_PARTITION(A,p,r)
3.      RANDOMIZED_QUICK_SORT(A,p,q-1)
4.      RANDOMIZED_QUICK_SORT(A,q+1,r)
```


22

Stabilità di QUICK_SORT

- QUICK_SORT non è stabile:



23

Algoritmi di ordinamento per confronto

	caso migliore	caso medio	caso peggiore	in loco	stabile
SELECTION-SORT	$\Theta(n^2)$			*si*	*si*
INSERTION-SORT	$\Theta(n)$	$\Theta(n^2)$	$\Theta(n^2)$	*si*	*si*
MERGE-SORT	$\Theta(n \log n)$			*no*	*si*
HEAP-SORT	$\Theta(n \log n)$			*si*	*no*
QUICK-SORT	$\Theta(n \log n)$	$\Theta(n \log n)$	$\Theta(n^2)$	*si*	*no*

24