Due algoritmi di ordinamento

basati sulla tecnica Divide et Impera:

Mergesort e Quicksort

(13 ottobre 2009, 2 novembre 2010)

Ordinamento

INPUT: un insieme di n oggetti $a_1, a_2, \ldots, a_n$
presi da un dominio totalmente ordinato secondo ≤

OUTPUT: una permutazione degli oggetti $a'_1, a'_2, \ldots, a'_n$ tale che
$a'_1 \leq a'_2 \leq \ldots \leq a'_n$

Applicazioni:

- Ordinare alfabeticamente lista di nomi, o insieme di numeri, o insieme di compiti d'esame in base a cognome studente

- Velocizzare altre operazioni (per es. è possibile effettuare ricerche in array ordinati in tempo O(log n))

- Subroutine di molti algoritmi (per es. *greedy*)

- ….

Algoritmi per l'ordinamento

Data l'importanza, esistono svariati algoritmi di ordinamento, basati su tecniche diverse:

Insertionsort
Selectionsort
Heapsort
Mergesort
Quicksort
Bubblesort
Countingsort
…..

Ognuno con i suoi lati positivi e negativi.

Il Mergesort e il Quicksort sono entrambi basati sulla tecnica Divide et Impera, ma risultano avere differenti prestazioni

Mergesort

Dato un array di n elementi

I) Divide: trova l'indice della posizione centrale e divide l'array in due parti ciascuna con n/2 elementi (più precisamente $\lfloor n/2 \rfloor$ e $\lceil n/2 \rceil$)

II) Risolve i due sottoproblemi ricorsivamente

III) Impera: fonde i due sotto-array ordinati usando la procedura Merge

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

Vedremo che la soluzione è $T(n) = \Theta(n \log n)$

Nota:
Il tempo di esecuzione di Merge è $\Theta(n)$ (e non solo O(n)).

Infatti:
nel caso peggiore faremo O(n) confronti:

nel caso migliore faremo $\Omega(n)$ confronti

Ricorda: Il tempo di esecuzione di un algoritmo è $\Theta(f(n))$ se nel caso peggiore è O(f(n)) e nel caso migliore è $\Omega(f(n))$

Il tempo di esecuzione di Mergesort è $\Theta(n \log n)$

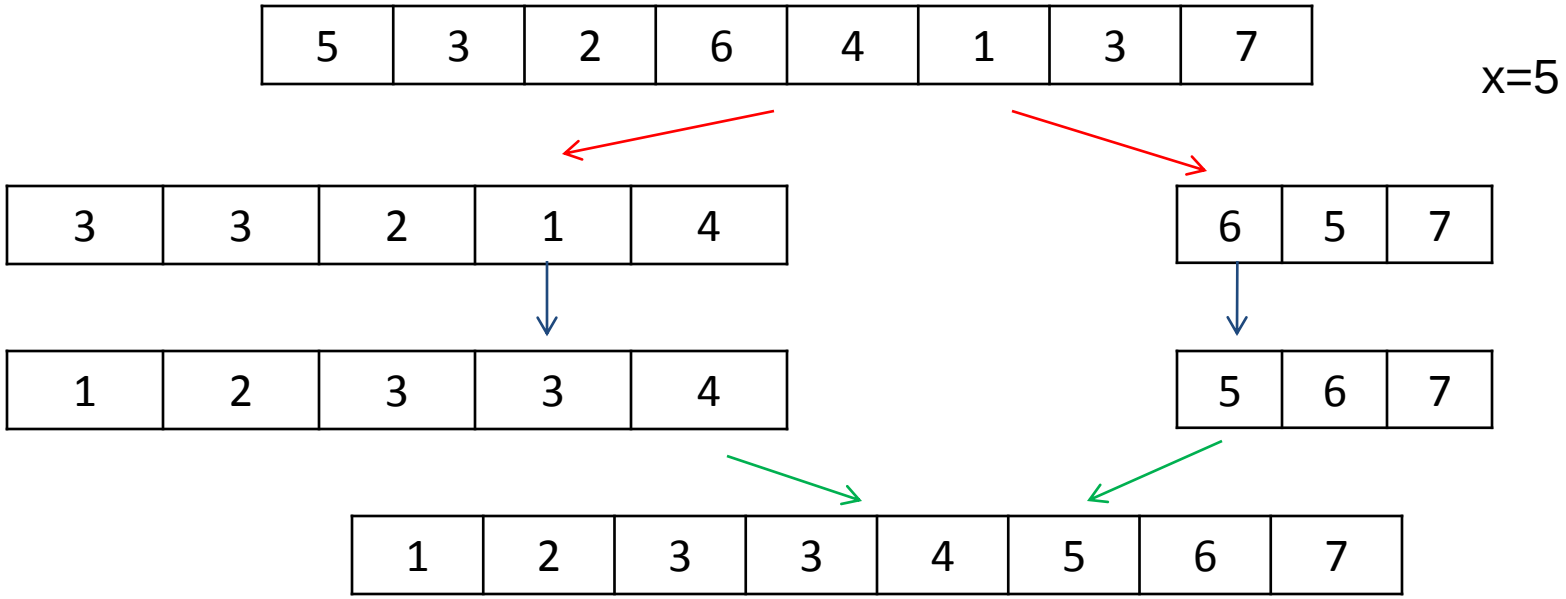
Quicksort

Dato un array di n elementi

I) Divide: scegli un elemento x dell'array (detto "pivot" o perno) e partiziona la sequenza in elementi ≤ x ed elementi ≥ x

II) Risolvi i due sottoproblemi ricorsivamente

III) Impera: restituisci la concatenazione dei due sotto-array ordinati



Scelta del pivot

L’algoritmo funziona per qualsiasi scelta (primo / ultimo / …),
ma se voglio algoritmo “deterministico” devo fissare la scelta;
nel seguito sceglierò il **primo**.

Altrimenti: scelgo “random” e avrò “algoritmi randomizzati”
(vedi Kleinberg, Tardos)

Partizionamento

Partiziona l'array in elementi ≤ x ed elementi ≥ x

Banalmente:
scorro l'array da 1 ad n e inserisco gli elementi ≤ pivot in un nuovo array e quelli ≥ del pivot in un altro nuovo array

Però:

1) avrei bisogno di array ausiliari

2) di che dimensione? I due sotto-array hanno un numero variabile di elementi

Partizione “in loco”

Partition:

- pivot = A[1]
- Scorri l’array da destra verso sinistra (con un indice j) e da sinistra verso destra (con un indice i) :

  da destra verso sinistra, ci si ferma su un elemento ≤ del pivot
  da sinistra verso destra, ci si ferma su un elemento ≥ del pivot;

- Scambia gli elementi
- Riprendi la scansione finché i e j si incrociano

```
Partition (A, p, r)

x = A[p]
i = p-1
j = r+1
while True
      do repeat j=j-1 until A[j]≤ x
         repeat i=i+1 until A[i]≥ x
         if i < j
              then scambia A[i] ↔ A[j]
              else return j
```

Partizione in loco: un esempio

Correttezza di Partition

Perché funziona?

Ad ogni iterazione (quando raggiungo il while):
la “parte verde” di sinistra (da p ad i) contiene elementi ≤ 5;
la “parte verde” di destra (da j a r) contiene elementi ≥ 5.

Tale affermazione è vera all’inizio e si mantiene vera ad ogni iterazione (per induzione)

Analisi Partition

Il tempo di esecuzione è $\Theta(n)$

```
Quicksort (A, p, r)

   if p < r then
            q = Partition (A,p,r)
            Quicksort(A, p, q)
            Quicksort(A, q+1, r)
```

Correttezza: la concatenazione di due array ordinati in cui l'array di sinistra contiene elementi minori o uguali degli elementi dell'array di destra è un array ordinato

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

Se k sono gli elementi da p a q (e n-k i rimanenti da q+1 a r)
con $1 \leq k \leq n-1$

Analisi Quicksort

Un primo caso: ad ogni passo il pivot scelto è il minimo o il massimo degli elementi nell'array (la partizione è 1 | n-1):

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

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

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

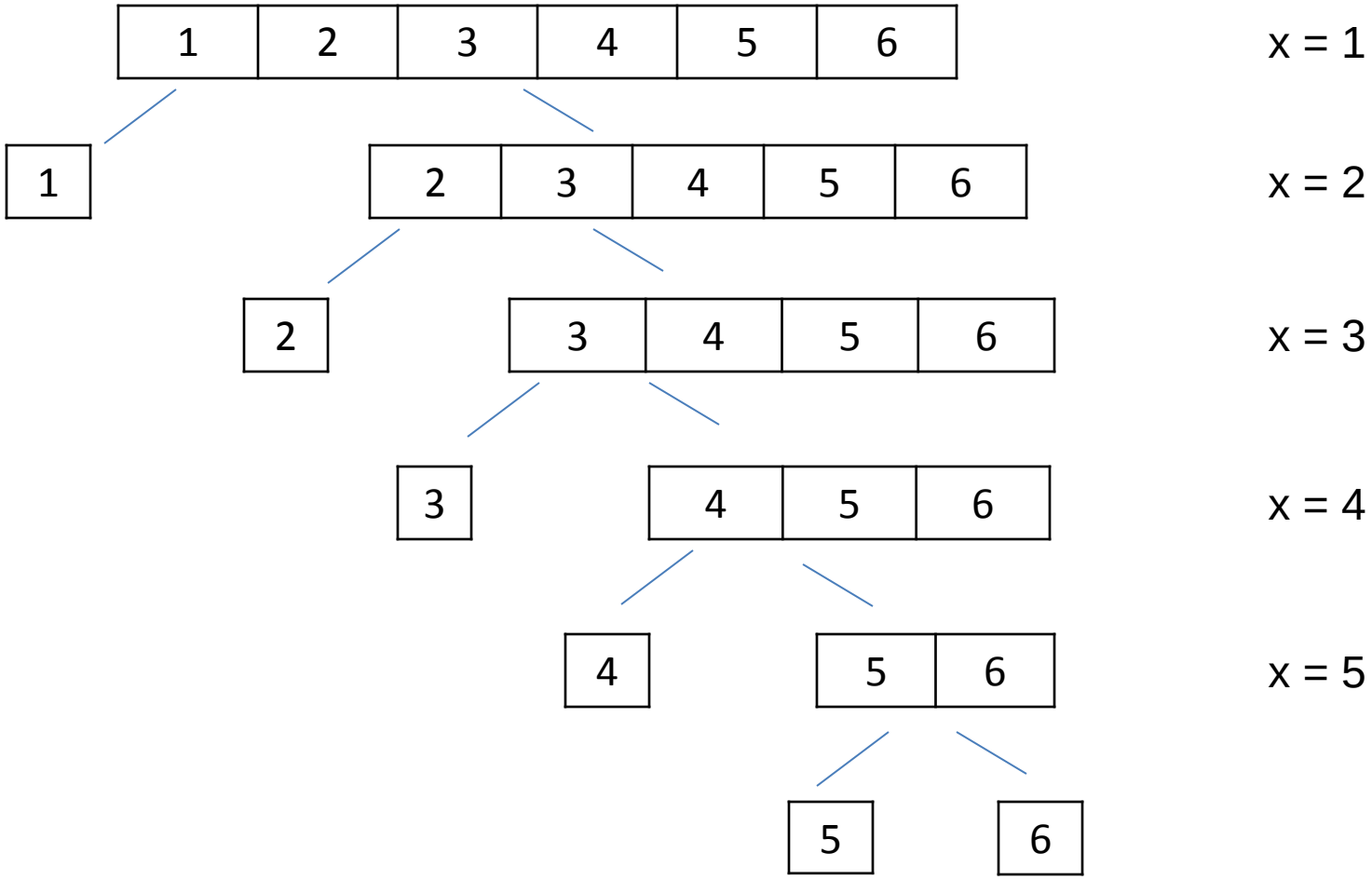
La cui soluzione (vedremo) è $T(n) = \Theta(n^2)$

Si può dimostrare che questo è il **caso peggiore**; quindi per il Quicksort:

$$\mathbf{T(n) = O(n^2)}$$

Un esempio del caso peggiore del Quicksort

Un array ordinato



Analisi Quicksort

Un altro caso: ad ogni passo il pivot scelto è la “mediana” degli elementi nell’array (la partizione è n/2 | n/2):

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

La cui soluzione è $T(n) = \Theta(n \log n)$

(è la stessa relazione di ricorrenza del Mergesort)

Si può dimostrare che questo è il **caso migliore**; quindi per il Quicksort:

$$\mathbf{T(n) = \Omega(n \log n)}$$

Riassumendo: $T(n) = O(n^2)$ e $T(n) = \Omega(n \log n)$

Il caso migliore è diverso dal caso peggiore quindi
T(n) **non** è Θ di nessuna funzione

Quicksort vs Mergesort

Fase		MergeSort	Tempi	QuickSort	Tempi
I	Divide	$q = \lfloor (p+r)/2 \rfloor$	$\Theta(1)$	PARTITION	$\Theta(n)$
II	Ricorsione	$\lfloor n/2 \rfloor \mid \lceil n/2 \rceil$	$2T(n/2)$	$k \mid n-k$	T(k) + T(n-k)
III	Combina	MERGE	$\Theta(n)$	niente	$\Theta(1)$
			$T(n) = 2T(n/2) + \Theta(n)$		$T(n) = T(k) + T(n-k) + \Theta(n)$
			$T(n) = \Theta(n \log n)$		$T(n) = O(n^2), T(n) = \Omega(n \log n)$

Da ricordare

Esistono algoritmi di ordinamento con tempo nel caso peggiore

$\Theta(n^2)$

$\Theta(n \log n)$

Esistono anche algoritmi di ordinamento con tempo nel caso peggiore

$\Theta(n)$

ma **non** sono basati sui confronti e funzionano **solo** sotto certe ipotesi

Inoltre si può dimostrare che **tutti** gli algoritmi di ordinamento basati sui confronti richiedono $\Omega(n \log n)$ confronti nel caso peggiore!

Si dice che $\Omega(n \log n)$ è una delimitazione inferiore (lower bound) al numero di confronti richiesti per ordinare n oggetti.

Delimitazione inferiore = quantità di risorsa **necessaria** per risolvere un determinato problema

Indica la difficoltà intrinseca del problema.