

Corso di laurea in Matematica
Insegnamento di Informatica generale
Lezioni in modalità mista o a distanza

Il problema dell'ordinamento: algoritmo Merge Sort

Giancarlo Bongiovanni
Ivano Salvo



SAPIENZA
UNIVERSITÀ DI ROMA

Queste dispense sono state realizzate sulla base delle slide
preparate da T. Calamoneri e G. Bongiovanni
per il corso di Informatica Generale tenuto a distanza nell'A.A. 2019/20

La vignetta della settimana



Prima di cominciare... un esercizio - 1

Nell'algoritmo di Insertion sort è possibile ricercare la posizione in cui inserire l'elemento i -esimo tramite la ricerca binaria. Come cambia il calcolo costo computazionale dell'algoritmo?

Soluzione.

Per ogni $j = 2$ to n , l'algoritmo di Insertion Sort in versione standard:

- scorre la parte ordinata di vettore da destra a sinistra
- confronta ciascun elemento con x
- trova la posizione corretta di x
- sposta tutti gli elementi che sono a destra per fare posto ad x

Prima di cominciare... un esercizio - 2

Consideriamo lo pseudocodice modificato dell'Insertion Sort

```
Funzione Insertion_Sort(A[1..n])
  for j = 2 to n do
    x ← A[j]
    k ← RicercaBin(A,1, j, x)
    /* x dovrebbe stare in pos.k
    i ← j - 1
    while ((i>0) and (i>=k)
      A[i+1] ← A[i]
      i ← i - 1
    A[k] ← x
```

$(n-1) \Theta(1) + \Theta(1)$
 $\Theta(1)$
 $O(\log j)$
 $\Theta(1)$
 $t_j \Theta(1) + \Theta(1)$
 $\Theta(1)$
 $\Theta(1)$

Prima di cominciare... un esercizio - 3

- Il contributo della ricerca binaria non si sostituisce a t_j , ma **si aggiunge** ad esso.
- Anche se troviamo la posizione di x più velocemente, dobbiamo comunque spostare gli elementi a destra...
- pertanto il costo computazionale non migliora!

Nella precedente lezione...

...abbiamo dimostrato il seguente:

Teorema. Il costo computazionale di qualunque algoritmo di ordinamento basato su confronti è $\Omega(n \log n)$.

Riusciamo a progettare degli algoritmi che richiedono costo computazionale uguale proprio a $\Theta(n \log n)$ e sono, quindi, **ottimi**?

Merge Sort - 1

L'algoritmo **merge sort** (**ordinamento per fusione**) è un algoritmo ricorsivo che adotta una tecnica algoritmica detta **divide et impera**. Essa può essere descritta come segue:

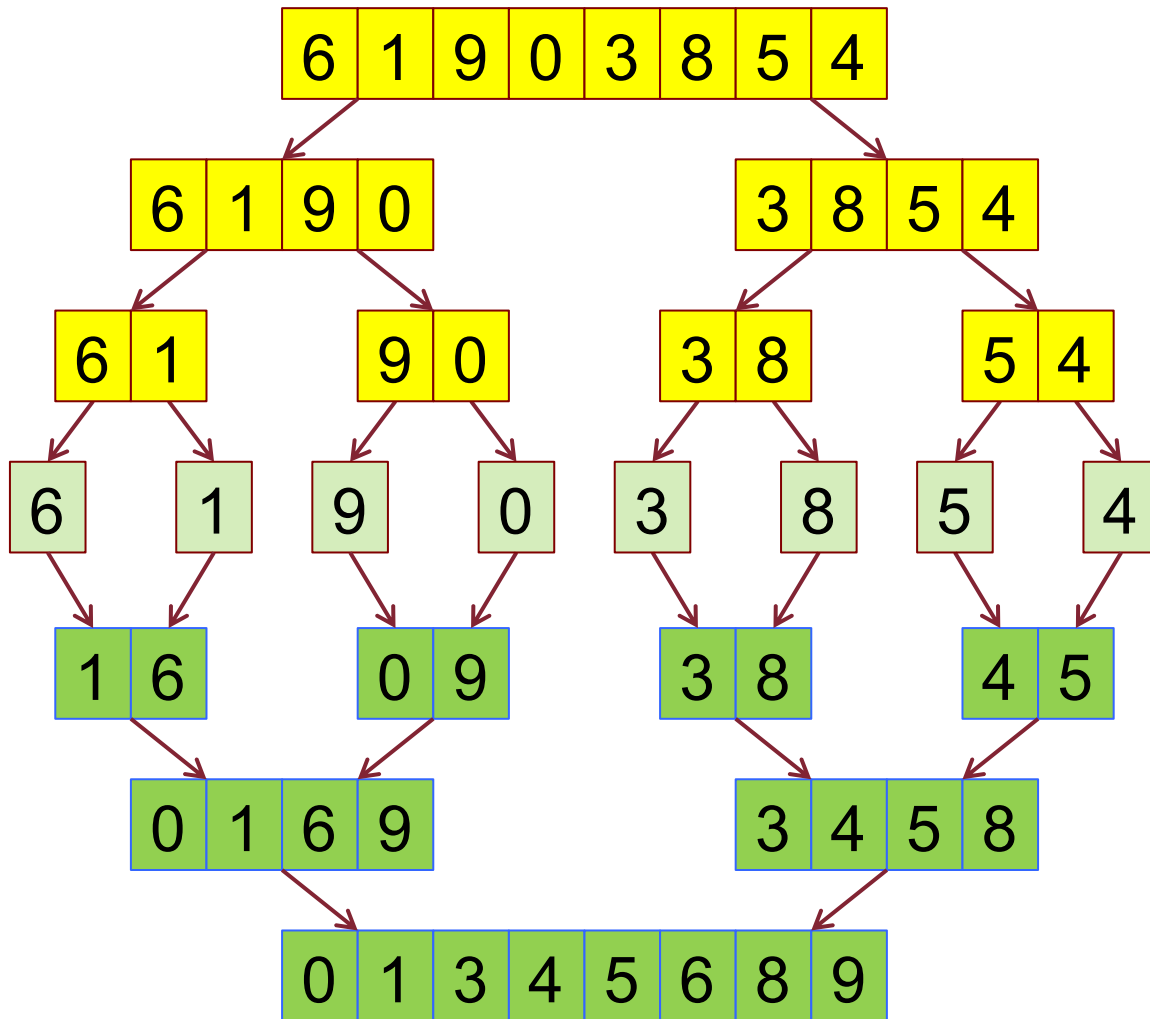
- il problema complessivo si suddivide in sottoproblemi di dimensione inferiore (**divide**);
- i sottoproblemi si risolvono ricorsivamente (**impera**);
- le soluzioni dei sottoproblemi si compongono per ottenere la soluzione al problema complessivo (**combina**).

Merge Sort - 2

L'approccio dell'algoritmo Merge Sort è il seguente:

- **divide**: la sequenza di n elementi viene divisa in due sottosequenze di $n/2$ elementi ciascuna;
- **impera**: le due sottosequenze di $n/2$ elementi vengono ordinate ricorsivamente;
- **passo base**: la ricorsione termina quando la sottosequenza è costituita di un solo elemento, per cui è già ordinata;
- **combina**: le due sottosequenze – ormai ordinate – di $n/2$ elementi ciascuna vengono “fuse” in un'unica sequenza ordinata di n elementi.

Merge Sort - 3



• **divide**: la sequenza di n elementi viene divisa in due sotto-sequenze di $n/2$ elementi ciascuna;

• **impera**: le due sotto-sequenze di $n/2$ elementi vengono ordinate ricorsivamente;

• **passo base**: la ricorsione termina quando la sotto-sequenza è costituita di un solo elemento, per cui è già ordinata;

• **combina**: le due sotto-sequenze – ormai ordinate – di $n/2$ elementi ciascuna vengono “fuse” in un’unica sequenza ordinata di n elementi.

Merge Sort - 4

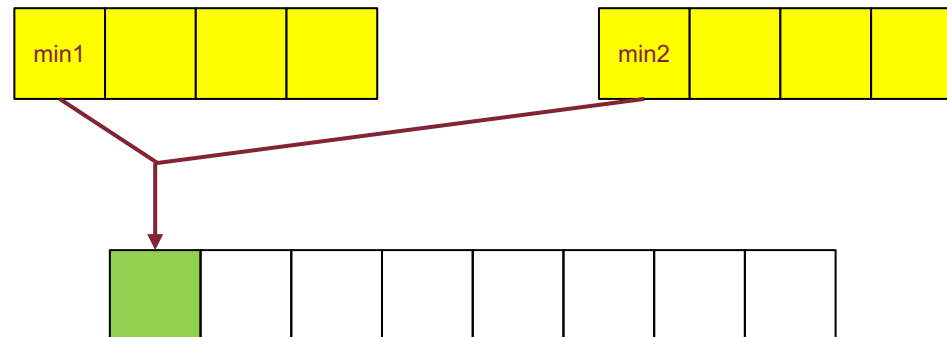
```
Funzione Merge_sort (A: vettore; ind_primo,
                    ind_ultimo: intero)           T(n)
if (ind_primo < ind_ultimo)                        $\Theta(1)$ 
    ind_medio  $\leftarrow$  (ind_primo+ind_ultimo) DIV 2  $\Theta(1)$ 
    Merge_sort (A, ind_primo, ind_medio)          T(n/2)
    Merge_sort (A, ind_medio + 1, ind_ultimo)    T(n/2)
    Fondi (A, ind_primo, ind_medio, ind_ultimo)  $S(n)$ 
return
```

$$T(n) = \Theta(1) + 2T(n/2) + S(n) \quad \text{dove } S(n) = \text{costo di Fondi()}$$
$$T(1) = \Theta(1)$$

Merge Sort - 5

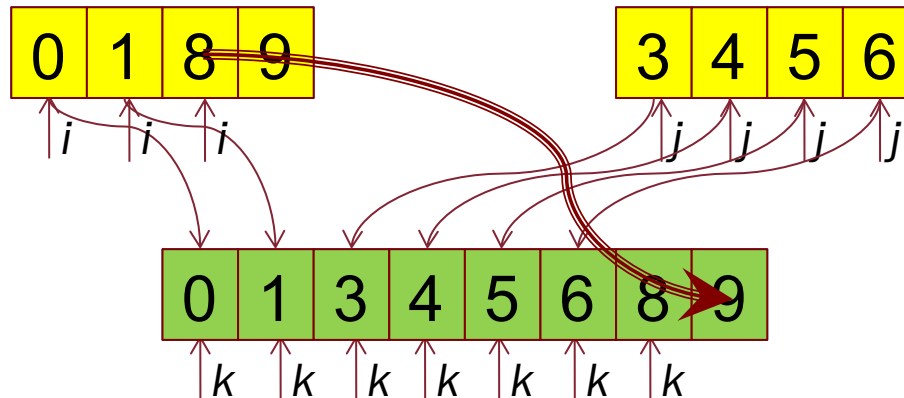
Funzionamento della funzione `Fondi()`:

- la funzione sfrutta il fatto che le due sottosequenze sono ordinate;
- il minimo della sequenza complessiva non può che essere ***il più piccolo fra i minimi delle due sottosequenze*** (se essi sono uguali, scegliere l'uno o l'altro non fa differenza);
- dopo aver eliminato da una delle due sottosequenze tale minimo, la proprietà rimane: il prossimo minimo non può che essere il più piccolo fra i minimi delle due parti rimanenti delle due sottosequenze.



Merge Sort - 6

Esempio di funzionamento della funzione `Fondi()`:



Dopo aver ricopiato anche l'ultimo elemento il vettore complessivo risulta ordinato.

Merge Sort - 7

```
Funzione Fondi (A: vettore; ind_primo, ind_medio, ind_ultimo: int)
  i ← ind_primo; j ← ind_medio + 1; k ← 1;
  while ((i ≤ ind_medio) and (j ≤ ind_ultimo))
    if (A[i] < A[j])
      B[k] ← A[i]; i ← i + 1
    else
      B[k] ← A[j]; j ← j + 1
    k ← k + 1
  while (i ≤ ind_medio) //il primo sottovettore non è terminato
    B[k] ← A[i]; i ← i + 1; k ← k + 1
  while (j ≤ ind_ultimo) //il secondo sottovettore non è terminato
    B[k] ← A[j]; j ← j + 1; k ← k + 1
  ricopia B[1..k-1] su A[ind_primo..ind_ultimo]
  return
```

Merge Sort - 8

Valutiamo il costo computazionale della funzione `Fondi()`:

- inizializzazione delle variabili: $\Theta(1)$;
- primo ciclo `while`
 - ogni iterazione ha costo $\Theta(1)$ e incrementa di 1 l'indice i oppure l'indice j . Quindi il costo del `while` varia da un minimo di $n/2$ a un massimo di n , ossia è $\Theta(n)$.
- secondo e terzo `while` (mai eseguiti entrambi):
 - si ricopia nel vettore B l'eventuale "coda" di una delle due sottosequenze: $O(n)$;
- copia del vettore B nell'opportuna porzione del vettore A: $\Theta(n)$.

Dunque il costo $S(n)$ della funzione `Fondi()` è:

$$S(n) = \Theta(1) + \Theta(n) + O(n) + \Theta(n) = \Theta(n)$$

Merge Sort - 9

Quindi il costo computazionale del Merge Sort:

```
Funzione Merge_sort (A: vettore; ind_primo, ind_ultimo)
  if (ind_primo < ind_ultimo)
    ind_medio ← (ind_primo + ind_ultimo) DIV 2
    Merge_sort (A, ind_primo, ind_medio)
    Merge_sort (A, ind_medio + 1, ind_ultimo)
    Fondi (A, ind_primo, ind_medio, ind_ultimo)
  return
```

è

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

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



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

Merge Sort - 10

OSSERVAZIONE

L'operazione di fusione non si può fare "in loco", cioè aggiornando direttamente il vettore A, senza incorrere in un aggravio del costo.

Infatti, in A bisognerebbe fare spazio via via al minimo successivo, ma questo costringerebbe a spostare di una posizione tutta la sottosequenza rimanente per ogni nuovo minimo, il che costerebbe $\Theta(n)$ operazioni elementari per ciascun elemento da inserire, facendo lievitare quindi il costo computazionale della fusione da $\Theta(n)$ a $\Theta(n^2)$.

Ciò a sua volta risulterebbe nell'equazione di ricorrenza:

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

la cui soluzione, come sappiamo, è:

$$T(n) = \Theta(n^2)$$

Alternativamente... [Cormen]

```
fun merge(array A, int inf, med, sup):  
  /* PREC: A[inf, med] e A[med+1, sup] ordinati*/  
  * POST: A[inf, sup] ordinato*/  
  
  alloca(B, med - inf + 2); alloca(C, sup-med+1)  
  copia(A, inf, med, B); copia(A, med+1, sup, C)  
  /* sentinelle per evitare di copiare le code */  
  B[med - inf + 2] ← +∞  
  C[sup - med + 1] ← +∞  
  
  for i=inf to sup do  
    if B[j] ≤ C[k] then A[i]←B[j]; j←j+1  
    else A[i]←C[j]; k←k+1
```

In C, la merge può essere scritta **in una forma più generale** che fonde **due vettori qualsiasi** (compreso due pezzi dello stesso vettore come qui). Vedi gli “Esercizi di Stile” – parte I

Esercizio svolto - 1

Nonostante Merge Sort funzioni in tempo $\Theta(n \log n)$ mentre Insertion Sort in $O(n^2)$, i fattori costanti sono tali che l'Insertion Sort è più veloce del Merge Sort per valori piccoli di n . Quindi, ha senso usare l'Insertion Sort dentro il Merge Sort quando i sottoproblemi diventano sufficientemente piccoli.

- Si consideri una modifica del Merge Sort in cui il caso base si applica ad una porzione del vettore di lunghezza k , che viene ordinata usando Insertion Sort.
- Le porzioni vengono combinate usando il meccanismo standard di fusione.
- Si determini il valore di k come funzione di n per cui l'algoritmo modificato ha lo stesso tempo di esecuzione asintotico del Merge Sort.

Esercizio svolto - 2

Il codice che realizza tale versione è il seguente.

```
Funzione Merge_Insertion (V, k, primo, ultimo)
  dim ← ultimo - primo + 1
  if dim > k
    medio ← (primo + ultimo) DIV 2
    Merge_Insertion (V, k, primo, medio)
    Merge_Insertion (V, k, medio + 1, ultimo)
    Fondi(primo, medio, ultimo)
  else InsertionSort(V, primo, ultimo)
```

La chiamata iniziale sarà:

```
Merge_Insertion (V, k, 1, n)
```

Esercizio svolto - 3

L'equazione di ricorrenza che lo caratterizza è la seguente:

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

$$T(k) = \Theta(k^2)$$

Risolviamola col metodo iterativo:

$$\begin{aligned} T(n) &= 2T(n/2) + \Theta(n) = \\ &= 2[2T(n/2^2) + \Theta(n/2^1)] + \Theta(n/2^0) = \\ &= 2[2[2T(n/2^3) + \Theta(n/2^2)] + \Theta(n/2^1)] + \Theta(n/2^0) = \\ &= 2^3 T(n/2^3) + 2^2 \Theta(n/2^2) + 2^1 \Theta(n/2^1) + 2^0 \Theta(n/2^0) \\ &\quad \dots \\ &= 2^h T(n/2^h) + \sum_{i=0}^{h-1} 2^i \Theta\left(\frac{n}{2^i}\right) \end{aligned}$$

Esercizio svolto - 4

Ci fermiamo incontrando il caso base, il che succede quando

$$n/2^h = k, \text{ ossia } h = \log n/k$$

Sostituendo tale valore nell'espressione precedente otteniamo:

$$\begin{aligned} T(n) &= 2^{\log n/k} T(n/2^{\log n/k}) + \sum_{i=0}^{\log n/k - 1} 2^i \Theta\left(\frac{n}{2^i}\right) = \\ &= \frac{n}{k} \Theta(k^2) + \Theta\left(n \log \frac{n}{k}\right) = \\ &= \Theta(nk) + \Theta\left(n \log \frac{n}{k}\right) = \\ &= \Theta(nk) + \Theta(n \log n - n \log k) \end{aligned}$$

Se $k = O(\log n)$ otteniamo:

$$\begin{aligned} T(n) &= \Theta(n \log n) + \Theta(n \log n - n \log \log n) = \\ &= \Theta(n \log n) \end{aligned}$$

Corso di laurea in Matematica
Insegnamento di Informatica generale
Lezioni a distanza

Esercizi per casa



SAPIENZA
UNIVERSITÀ DI ROMA

Esercizi

- Scrivere la versione iterativa dell'algoritmo di Merge sort.
- Scrivere la versione ricorsiva dell'algoritmo di Fusione.
- Si supponga di scrivere una variante del Merge sort, chiamata 4-Merge sort che, invece di suddividere il vettore da ordinare in 2 parti (e ordinarle separatamente), lo suddivide in 4 parti, le ordina ognuna riapplicando 4-Merge sort, e le riunifica usando un'opportuna variante 4-Fondi di Fondi (che fa la fusione su 4 sottovettori invece che su 2).
- Come cambia, se cambia, il costo computazionale di 4-Merge sort rispetto a quello di Merge sort?
- Come cambia, se cambia, il costo computazionale di un'ulteriore variante k-Merge sort che spezza il vettore in k sottovettori?