UNITELMA SAPIENZA

SAPIENZA
Università di Roma
Dipartimento di Informatica

# Corso di Introduzione agli algoritmi
# Prof.ssa Tiziana Calamoneri

# Il problema dell'Ordinamento: il Merge Sort

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 proprio $\Theta(n \log n)$ e sono, quindi ottimi?

# Merge sort (1)

UNITELMA SAPIENZA

SAPIENZA Università di Roma Dipartimento di Informatica

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:

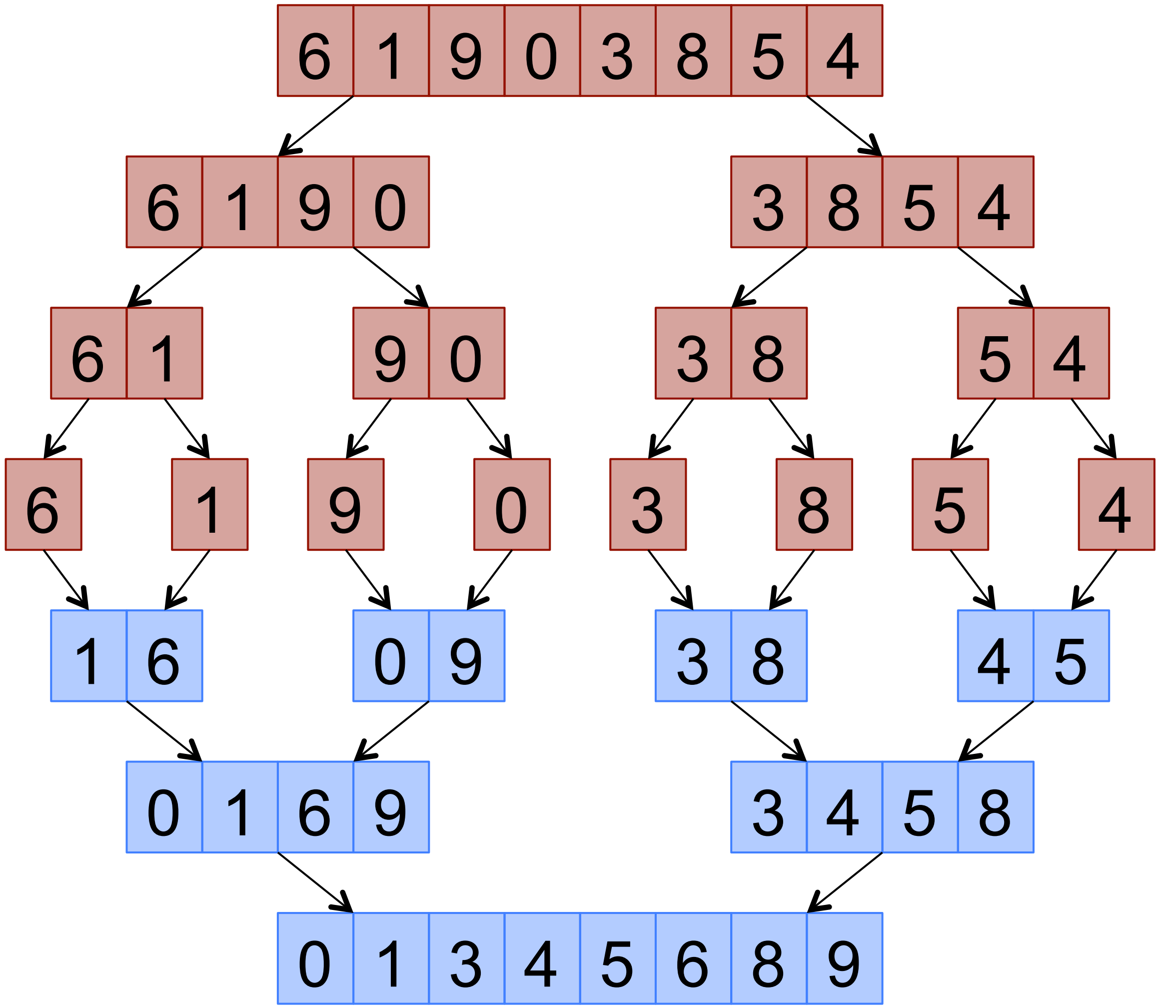
- il problema complessivo si suddivide in sottoproblemi di dimensione inferiore (***divide***);
- i sottoproblemi si risolvono ricorsivamente (***impera***);
- le soluzioni dei sottoproblemi si compongono per ottenere la soluzione al problema complessivo (***combina***).

## 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.

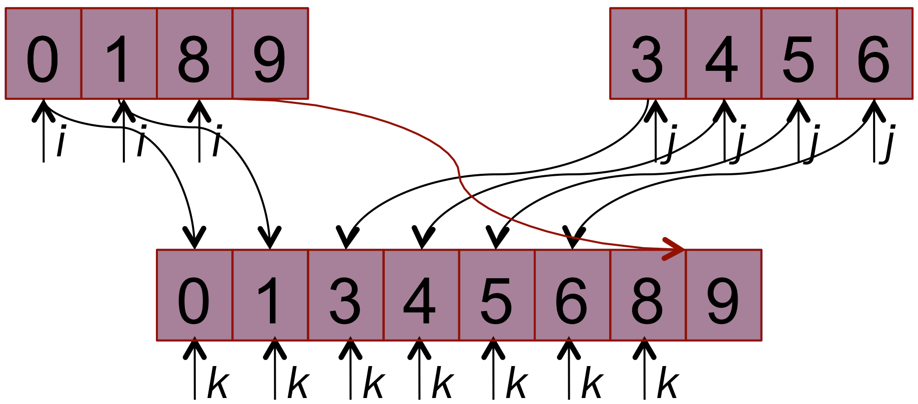
# Merge sort (4)

```
Funzione Merge_sort (A: vettore; ind_primo, ind_ultimo: intero)
    if (ind_primo < ind_ultimo)
        ind_medio ← (ind_primo +ind_ultimo)/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
```

## Merge sort (5)

Funzionamento di Fondi:

- 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)

```
Funzione Fondi (A: vettore; ind_primo, ind_medio, ind_ultimo)
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 sottovett. non è terminato
    B[k] ← A[i]; i ← i + 1; k ← k + 1
while (j ≤ ind_ultimo) //il secondo sottovett. 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
```

Costo: $\Theta(n)$

## Merge sort (7)

Costo computazionale del Merge Sort:

| | |
|---|---|
| Funzione Merge_sort (A: vettore; ind_primo, ind_ultimo) | $T(n)=$ |
|     if (ind_primo < ind_ultimo) | $\Theta(1)+$ |
|         ind_medio ← (*ind_primo +ind_ultimo)/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) | $\Theta(n)+$ |
|     return | $\Theta(1)$ |

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

OSS.

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 $\Theta(n^2)$.

# Merge sort (9)



Ecco una visualizzazione inusuale del Merge sort:

https://www.youtube.com/watch?v=dENca26N6V4

# Esercizi (1)

- 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?

# Esercizi (2)

- Si progetti un algoritmo il più efficiente possibile per i seguenti problemi:
  - Data una matrice *m x n,* si vogliono rimescolare i suoi elementi in modo che tutti i vettori riga e tutti i vettori colonna siano ordinati in senso non decrescente.
    (il costo computazionale dovrebbe essere strettamente inferiore di *nm log max{n,m}*).
  - Data una matrice *n x n,* si vogliono rimescolare i suoi elementi in modo che tutti gli elementi posizionati al di sopra della diagonale principale siano minori o uguali di tutti gli elementi che giacciono sulla diagonale principale che, a loro volta, siano minori o uguali di tutti gli elementi posizionati al di sotto della diagonale principale.