Merge Sort

lunedì 30 maggio 2022 14:18

E' divide et impera

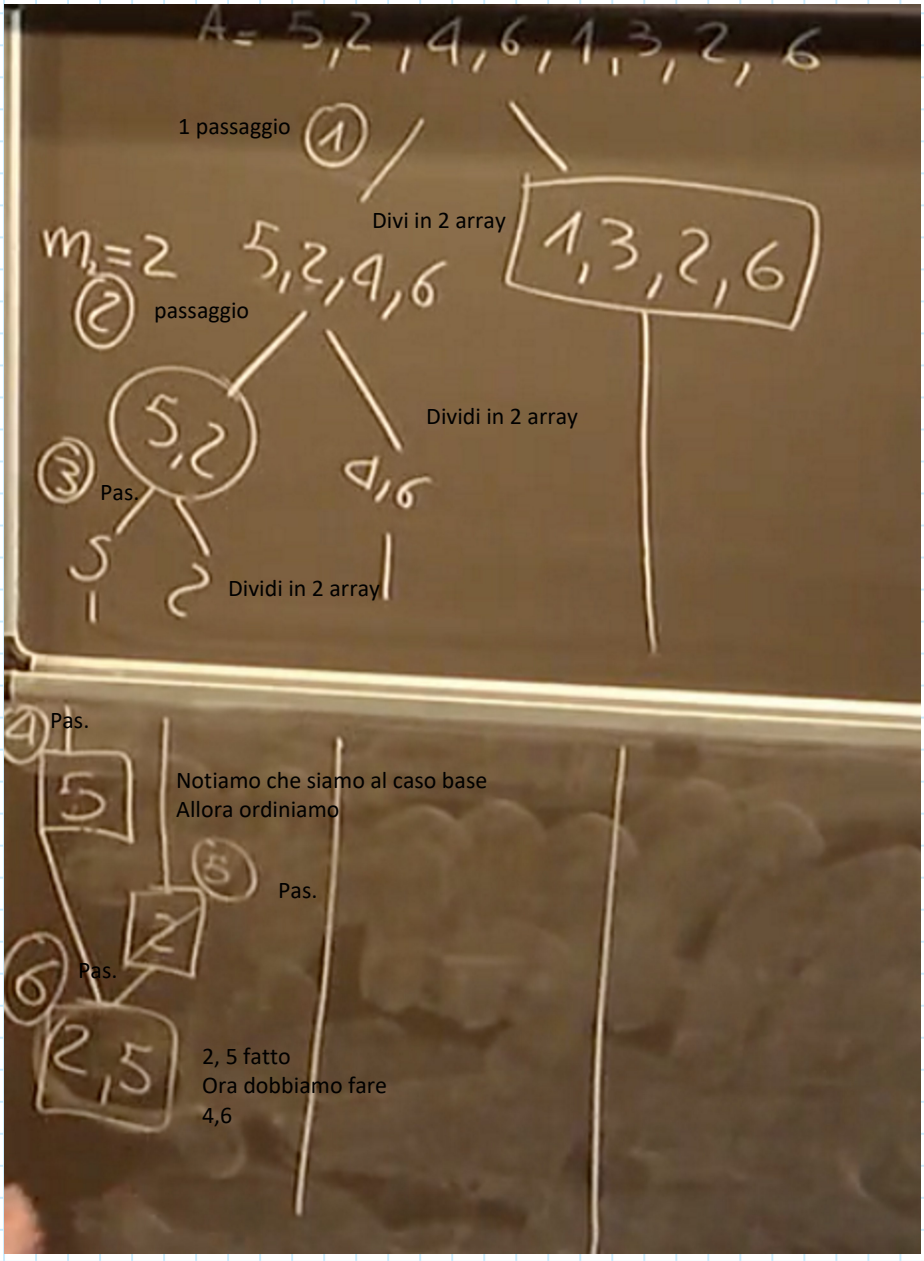
Problema: Ordina vettore A di N vettori

- Divide
  Dividiamo il nostro array in 2 parti

- Impera
  Ordina la prima metà e poi ordina seconda metà

- Combina
  Fonde in modo ordinato le due metà ordinate

A = [5, 2, 4, 6, 1, 3, 2, 6]



Saltiamo fino a che abbiamo [2, 5] e [4, 6]
Ora dobbiamo fare una merge

Fra [2, 5] e [4, 6]
E' più piccolo 2 o 4?
(2)

Più piccolo 4 o 5?
(2, 4)
Più piccolo 5 o 6?
(2, 4, 5)
Ultimo
[2, 4, 5, 6]
Pensa come se eliminassimo. Ora saltiamo a:
[2, 4, 5, 6] e [1, 2, 3, 6]
Chi è il più piccolo fra 1 e 2?
(1)
Eliminiamo l'1: [2a, 4, 5, 6] e [2b, 3, 6]
Chi è il più piccolo fra 2a e 2b? Qui abbiamo 2a <= 2b, è vero quindi
(1, 2a) -> [4, 5, 6] e [2, 3, 6]
Chi è più piccolo fra 2 e 3?
(1, 2, 2) -> [4, 5, 6] e [3, 6]
(1, 2, 2, 3) -> [4, 5, 6], [6]
(1, 2, 2, 3, 4) -> [5, 6], [6]
(1, 2, 2, 3, 4, 5) -> [6a], [6b]
(1, 2, 2, 3, 4, 5, 6a, 6b)
(1, 2, 2, 3, 4, 5, 6, 6)

Merge sort è un algoritmo stabile.
Algoritmo stabile = se abbiamo prima 2a e poi 2b, non succederà mai che 2b sarà prima 2a
Aka, elementi di uguale valore mantengono stesso ordine

Codice:

```
Void mergeSort(A[], int pIniziale, pFinale)
        # Se non sono nel caso base
        If pIn < pFin:
                # Prendo il medio, Divide
                M = (pIn + pFin)/2
                ## Impera
                # Ordino prima metà
                mergeSort(A[], pIn, m)
                # Ordino seconda metà
                mergeSort(A[], m + 1, pFin
                # Fondi la prima e seconda metà, combina
                Merge(A[], pIn, m, pFin)

Void Merge(A[], pIn, meta, pFin):
        i1 = pIn, i2 = meta+1,
        B[] = A[], ib = pIn

        # Quindi abbiamo ancora 1 valore da controllare
        While i1 <= meta and i2 <= fine:
                If A[i1] <= A[i2]
                        B[ib] = A[i1]
                        Ib++; i1++;
                Else:
                        B[ib] = A[i2]
                        Ib++; i2++;

        # Controlliamo gli elementi avanzati
        While i1 <= m:
                B[ib] = A[i1]
                I1++; Ib++;
        While i2 <= m:
                B[ib] = A[i2]
                I2++; Ib++;

        # Ora metto Ib dentro Ia
        Ib = In
        While ib<=pFin:
                A[ib] = B[ib]
                Ib++;
```

Analisi merge:

Abbiamo 4 while:

- W1
  -> Fusione 2 metà già ordinate
  -> Continua fino a che una delle 2 metà non è vuota
- W2
  -> Svuota la prima metà
- W3
  -> Svuota la seconda metà
- W4
  -> Porta i valori dell'array di appoggio in quello originale

Per il costo, noi sappiamo che:

- Siccome W4 fa da pInizio fino a pFine, lo possiamo
  Sostituire con un for che viene iterato N volte
- W1 verrà eseguito X volte
  W2 non sappiamo quante volte verrà eseguito
  W3 neanche
  Però Sappiamo che W1+W2+W3 deve fare N iterazioni siccome deve iterare
  Da pIn fino a pfine

Il tempo finale quindi sarà:

- K1*n+K2*n
  K1 = W1+W2+W3
  K2 = W4
  $\rightarrow \theta(n)$

Analisi merge sort:

- D(n)+I(n)*C(N)
- Sappiamo che Divide e Combine sono parti iterative
- Impera è la parte ricorsiva

Il nostro tempo è:
Caso base -> $\theta(1)$
Divido $\rightarrow \theta(1)$
Combina=Merge $\rightarrow \theta(n)$
Impera: $divide + \cdots + combina$
Dentro ai puntini dobbiamo mettere le chiamate ricorsive
E ogni chiamata ricorsiva con la quantità di elementi.
Abbiamo 2 chiamate ricorsive ed ognuna ottiene metà elementi

$$\rightarrow \theta(1) + 2T\left(\frac{n}{2}\right) + \theta(n) \sim 2T\left(\frac{n}{2}\right) + \theta(n)$$

$$T(n) = \begin{cases} \theta(1) \rightarrow n = 1 \\ 2t\left(\frac{n}{2}\right) + \theta(n) \rightarrow n > 1 \end{cases}$$