Algoritmi e Strutture Dati

Lezione 9

17 ottobre 2022

MERGE o FUSIONE

- Dati B, C array ordinati in modo non decrescente
- Costruire un array X ordinato contenente tutti gli elementi di B e C

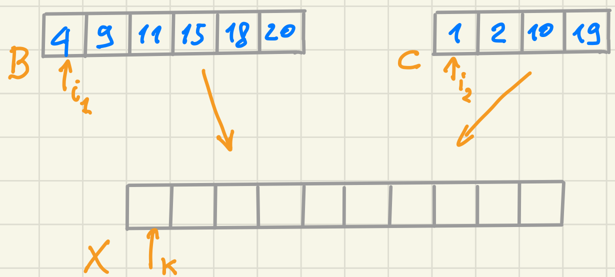
4	9	11	15	18	20

1	2	10	19

1	2	4	9	10	11	15	18	19	20

MERGE o FUSIONE

- Dati B, C array ordinati in modo non decrescente
- Costruire un array X ordinato contenente tutti gli elementi di B e C



$n = \ell_C + \ell_B$ = # tot elementi

$\le n-1$ cfr

$C_{merge}(n) = n-1$
nel caso peggiore

```
ALGORITMO merge (array B[0..ℓB−1], array C[0..ℓC−1]) → array
  Sia X[0..ℓB+ℓC−1] un array
  i1 ← 0, i2 ← 0, k ← 0
  WHILE i1 < ℓB AND i2 < ℓC DO
    IF B[i1] ≤ C[i2] THEN
      X[k] ← B[i1]
      i1 ← i1 + 1
    ELSE
      X[k] ← C[i2]
      i2 ← i2 + 1
    k ← k + 1
  IF i1 < ℓB THEN          // in B è restato qualcosa
    FOR j ← i1 TO ℓB−1 DO
      X[k] ← B[j]
      k ← k + 1
  ELSE                     // in C è restato qualcosa
    FOR j ← i2 TO ℓC−1 DO
      X[k] ← C[j]
      k ← k + 1
  RETURN X
```

MergeSort Array A[0 .. n - 1]

Se $n \leq 1$ A è già ordinato NON FARE NULLA!

altrimenti:

- dividi A in 2 parti della stessa lunghezza
- ordina le due parti separatamente
- fondi i due array ordinati in un unico array

	7	6	8	4	11	1	9	2
ordina	4	6	7	8	1	2	9	11
merge	1	2	4	6	7	8	9	11

ALGORITMO mergeSort (Array $A[0..n-1]$)

IF $n > 1$ THEN

divide-et-impera

$m \leftarrow n/2$

$B \leftarrow A[0..m-1]$ // 1ª metà di A

$C \leftarrow A[m..n-1]$ // 2ª metà di A

mergeSort (B)

mergeSort (C)

$A \leftarrow$ merge (B, C)

```
ALGORITMO mergeSort (Array A[0.. n-1])
  IF n > 1 THEN
    m ← n/2
    B ← A[0.. m-1]      // 1° metà di A
    C ← A[m.. n-1]      // 2° metà di A
    mergeSort(B)
    mergeSort(C)
    A ← merge(B, C)
```

\# cfr per ordinare array di n elementi

$$C(n) = \begin{cases} 0 & \text{se } n = 1 \\ \underbrace{C\left(\left\lfloor \frac{n}{2} \right\rfloor\right)}_{\text{mergeSort}(B)} + \underbrace{C\left(\left\lceil \frac{n}{2} \right\rceil\right)}_{\text{mergeSort}(C)} + \underbrace{\overbrace{C_{merge}(n)}^{n-1}}_{\text{merge}(B,C)} & \text{altrimenti} \end{cases}$$

per n pari:

$$C(n) = \begin{cases} 2C\left(\frac{n}{2}\right) + n - 1 & \text{se } n > 1 \\ 0 & \text{altrimenti} \end{cases}$$

$$C(n) = \begin{cases} 2C\left(\frac{n}{2}\right) + n - 1 & \text{se } n > 1 \\ 0 & \text{altrimenti} \end{cases}$$

$$C(n) = 2C\left(\frac{n}{2}\right) + n - 1 = 2\Big[\overbrace{2C\left(\frac{n}{2^2}\right) + \frac{n}{2} - 1}^{C\left(\frac{n}{2}\right)}\Big] + n - 1 = 2^2 C\left(\frac{n}{2^2}\right) + n - 2 + n - 1$$

$$= 2^2\Big[\overbrace{2C\left(\frac{n}{2^3}\right) + \frac{n}{2^2} - 1}^{C\left(\frac{n}{2^2}\right)}\Big] + n - 2 + n - 1$$

$$= 2^3 C\left(\frac{n}{2^3}\right) + n - 2^2 + n - 2^1 + n - 2^0$$

$$\dots$$

$$= 2^k C\left(\frac{n}{2^k}\right) + n \cdot k - \sum_{i=0}^{k-1} 2^i = 2^k C\left(\frac{n}{2^k}\right) + n \cdot k - 2^k + 1$$

$$\frac{n}{2^k} = 1 \qquad n = 2^k \qquad k = \log_2 n \qquad \longrightarrow \qquad C(n) = n\,\overbrace{C(1)}^{0} + n \log_2 n - n + 1$$

$$C(n) = n \log_2 n - n + 1$$

per n potenza di 2

- Per n potenza di n

$$C(n) = n\log_2 n - n + 1 = \Theta(n\log n)$$

- In generale $\exists N$ potenza di 2 con $n \le N < 2n$

$$C(n) \le C(N) = N\log_2 N - N + 1 < 2n\log_2 2n - n + 1$$

$$= 2n\,(1 + \log_2 n) - n + 1$$

$$= 2n + 2n\log_2 n - n + 1$$

$$= 2n\log_2 n + n + 1 = \Theta(n\log n)$$

ALGORITMO mergeSort (Array A[0.. n-1])
IF $n > 1$ THEN
  $m \leftarrow n/2$
  $B \leftarrow A[0..m-1]$ //1° metà di A
  $C \leftarrow A[m..n-1]$ //2° metà di A
  mergeSort(B)
  mergeSort(C)
  $A \leftarrow$ merge(B, C)

Tempo

$T(n)$

$\rightarrow$ tempo: $\Theta(n)$

$\rightarrow T(\lfloor \frac{n}{2} \rfloor) + T(\lceil \frac{n}{2} \rceil)$

$\rightarrow$ tempo merge + tempo per copiare il risultato in A

se che costano costante $\Theta(n)$

$\Theta(n)$

se $n=1$

$T(n) = a$ costante

$$\Theta(n) + T(\lfloor \tfrac{n}{2} \rfloor) + T(\lceil \tfrac{n}{2} \rceil) + \Theta(n)$$

$bn + c$

$$T(n) = \begin{cases} 2T(\frac{n}{2}) + bn + c & \text{se } n > 1 \\ a & \text{altrimenti} \end{cases}$$

$$T(n) = bn \log_2 n + an + c(n-1) = \Theta(n \log n)$$

```
ALGORITMO mergeSort (Array A[0.. n-1])
  IF n > 1 THEN
    m ← n/2
    B ← A[0.. m-1]      // 1° metà di A
    C ← A[m.. n-1]      // 2° metà di A
    mergeSort(B)
    mergeSort(C)
    A ← merge(B, C)
```

$H(n)$ = altezza dello stack ricorsione su array di lunghezza n

se $n > 1$

$$H(n) = 1 + \max\left(H\left(\left\lfloor \frac{n}{2} \right\rfloor\right), H\left(\left\lceil \frac{n}{2} \right\rceil\right)\right)$$

se $n \leq 1$ $H(n) = 1$

$$H(n) = \begin{cases} 1 + \max\left(H\left(\left\lfloor \frac{n}{2} \right\rfloor\right), H\left(\left\lceil \frac{n}{2} \right\rceil\right) \right) & \text{se } n > 1 \\ 1 & \text{altrimenti} \end{cases}$$

per n pari $\quad 1 + H\left(\frac{n}{2}\right)$

risoluzione

$$H(n) = \begin{cases} 1 + H\left(\frac{n}{2}\right) & \text{se } n > 1 \\ 1 & \text{altrimenti} \end{cases}$$

$\Rightarrow H(n) = 1 + \log_2 n$ per n potenza di 2

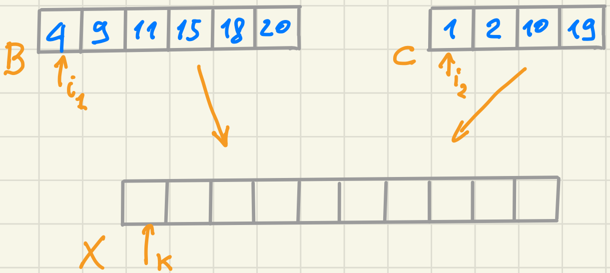
in generale $H(n) = 2 + \log_2 n = \Theta(\log n)$

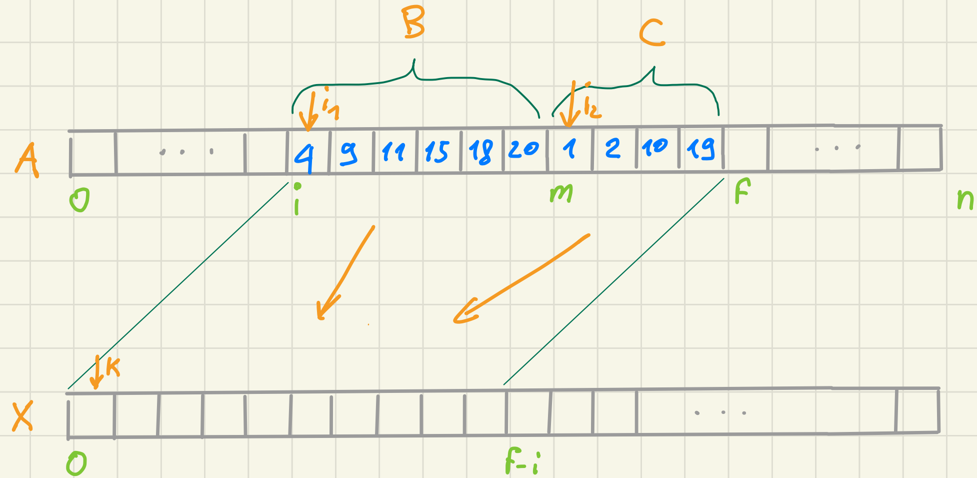
MergeSort: implementazione

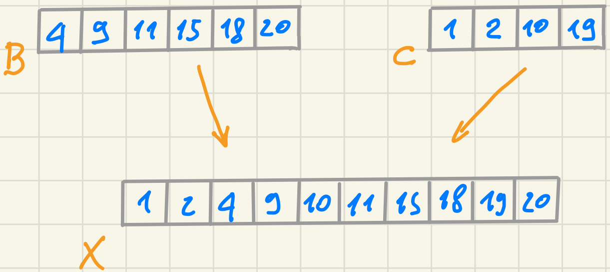
Implementando DIRETTAMENTE MergeSort come è stato scritto, ad ogni chiamata su array A di lunghezza > 1:

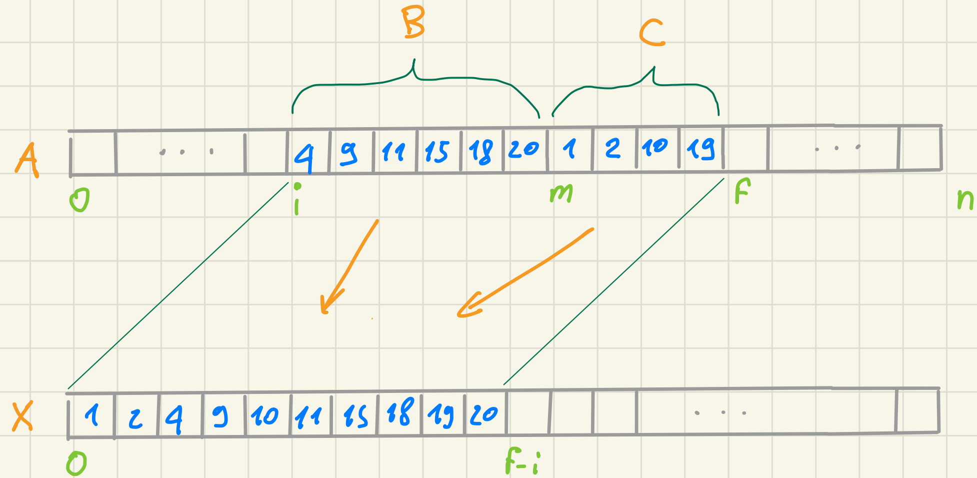
- si creano due nuovi array B e C
  $\Rightarrow$ spreco di spazio
- si copia in essi il contenuto di A
  $\Rightarrow$ spreco di tempo

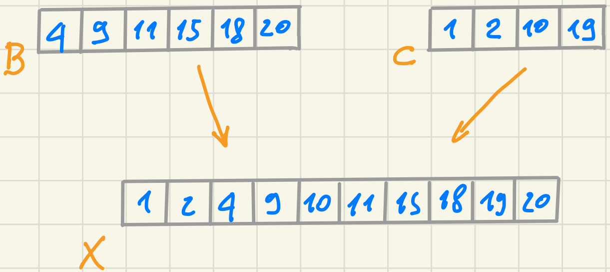
$\Rightarrow$ Soluzione alternativa con un unico array ausiliario X per il merge

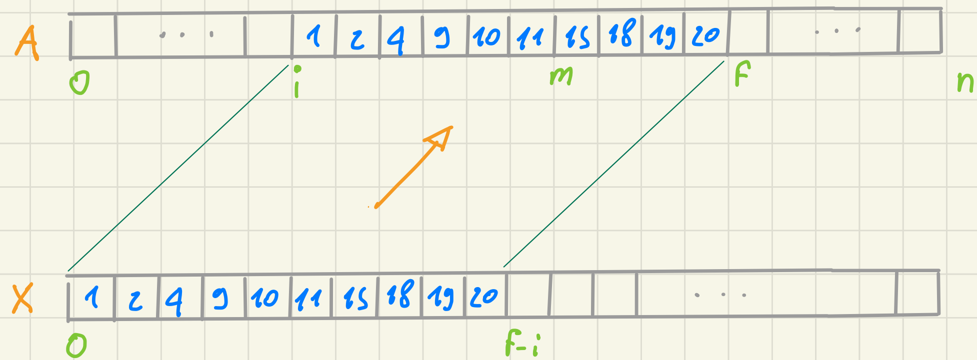

B
4 9 11 15 18 20
i_1
C
1 2 10 19
i_2
X
k


B
C
i_1
i_2
A
4 9 11 15 18 20 1 2 10 19
0
i
m
f
n
X
k
0
f-i

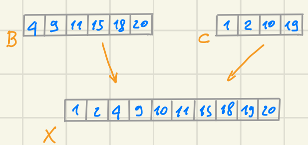

B
4 9 11 15 18 20
C
1 2 10 19
X
1 2 4 9 10 11 15 18 19 20


B
C
A
4 9 11 15 18 20 1 2 10 19
0
i
m
f
n
X
1 2 4 9 10 11 15 18 19 20
0
f-i

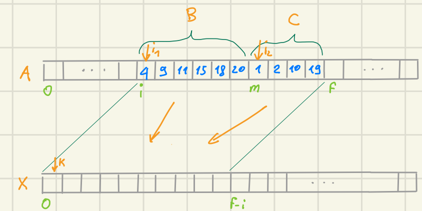

B
4 9 11 15 18 20
C
1 2 10 19
X
1 2 4 9 10 11 15 18 19 20

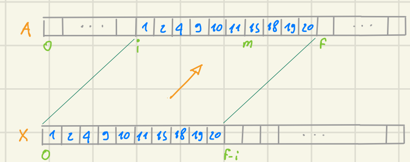

A
. . . 1 2 4 9 10 11 15 18 19 20 . . .
0
i
m
f
n
X
1 2 4 9 10 11 15 18 19 20 . . .
0
f-i

```
ALGORITMO merge (array B[0..ℓB-1], array C[0..ℓC-1]) → array
  Sia X[0.. ℓB+ℓC - 1] un array
  i1 ← 0,  i2 ← 0,  k ← 0
  WHILE  i1 < ℓB  AND  i2 < ℓC  DO
  |  IF  B[i1] ≤ C[i2]  THEN
  |  |  X[k] ← B[i1]
  |  |  i1 ← i1 + 1
  |  ELSE
  |  |  X[k] ← C[i2]
  |  |  i2 ← i2 + 1
  └ k ← k+1
  IF  i1 < ℓB  THEN            // in B è restato qualcosa
  |  FOR  j ← i1  TO  ℓB-1  DO
  |  |  X[k] ← B[j]
  |  └  k ← k+1
  ELSE                          // in C è restato qualcosa
  |  FOR  j ← i2  TO  ℓC-1  DO
  |  |  X[k] ← C[j]
  └  └  k ← k+1
  RETURN  X
```




```
PROCEDURA merge (array A, indice i, indice m, indice f, array X)
  i1 ← i,  i2 ← m,  k ← 0
  WHILE  i1 < m  AND  i2 < f  DO
  |  IF  A[i1] ≤ A[i2]  THEN
  |  |  X[k] ← A[i1]
  |  |  i1 ← i1 + 1
  |  ELSE
  |  |  X[k] ← A[i2]
  |  └  i2 ← i2 + 1
  └ k ← k+1
  IF  i1 < m  THEN        // nella prima porzione è restato qualcosa
  |  FOR  j ← i1  TO  m-1  DO
  |  |  X[k] ← A[j]
  |  └  k ← k+1
  ELSE                    // nella seconda porzione è restato qualcosa
  |  FOR  j ← i2  TO  f-1  DO
  |  |  X[k] ← A[j]
  └  └  k ← k+1
  // copia il risultato del merge
  // in A[i..f-1]
  FOR  k ← 0  TO  f-i-1  DO
     A[i+k] ← X[k]
```

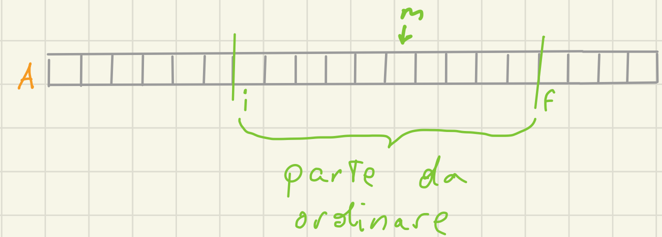




PROCEDURA mergeSort (Array A, indice i, indice f, Array X)

IF $f - i > 1$ THEN

- $m \leftarrow (i+f)/2$
- mergeSort(A, i, m, X)
- mergeSort(A, m, f, X)
- merge(A, i, m, f, X)



ALGORITMO mergeSort (Array A[0.. n-1])

- Sia X un array di lunghezza n
- mergeSort(A, 0, n, X)

Spazio

STACK $\Theta(\log n)$

Array ausiliario X $\Theta(n)$

$\Big\}\ \Theta(n)$

\#CFR $\Theta(n \log n)$

Tempo se cfr costano $O(1)$

$\Theta(n \log n)$