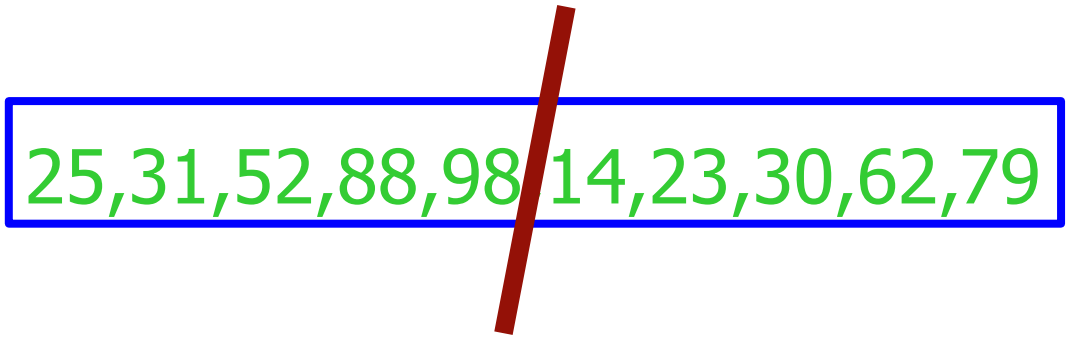
Programmazione e Algoritmica

MergeSort

MergeSort

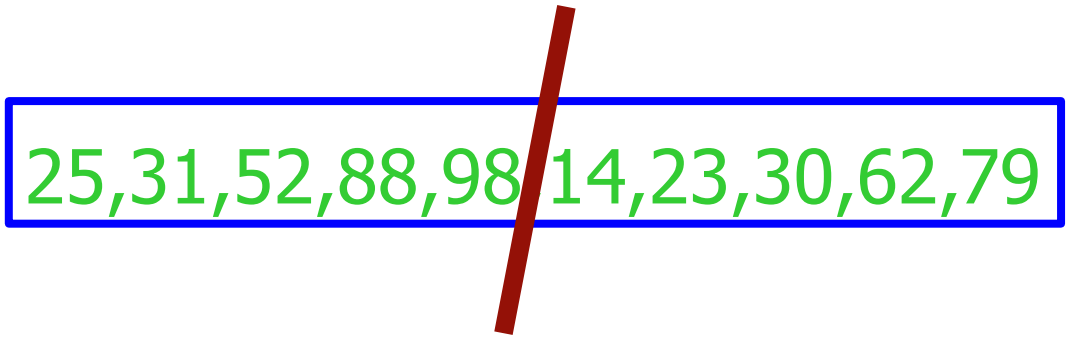
25,31,52,88,98,14,23,30,62,79

MergeSort



Dividi l'insieme da ordinare a metà

MergeSort

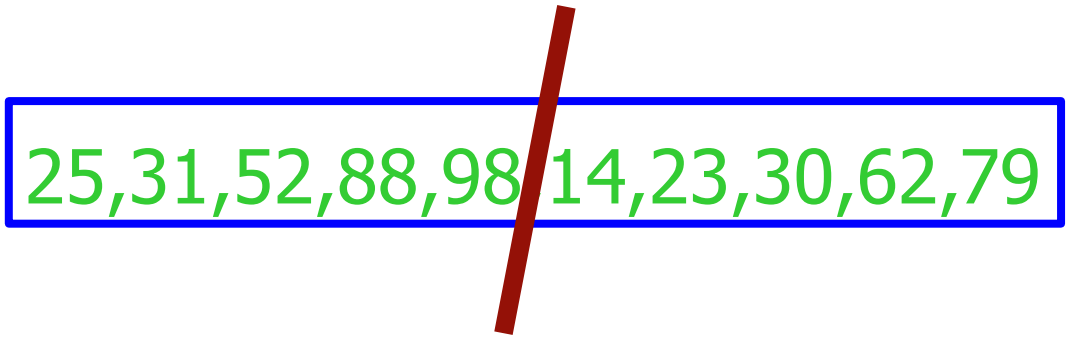


Dividi l'insieme da ordinare a metà

Affida una metà da ordinare ad un amico….

25,31,52,88,98

MergeSort



Dividi l'insieme da ordinare a metà

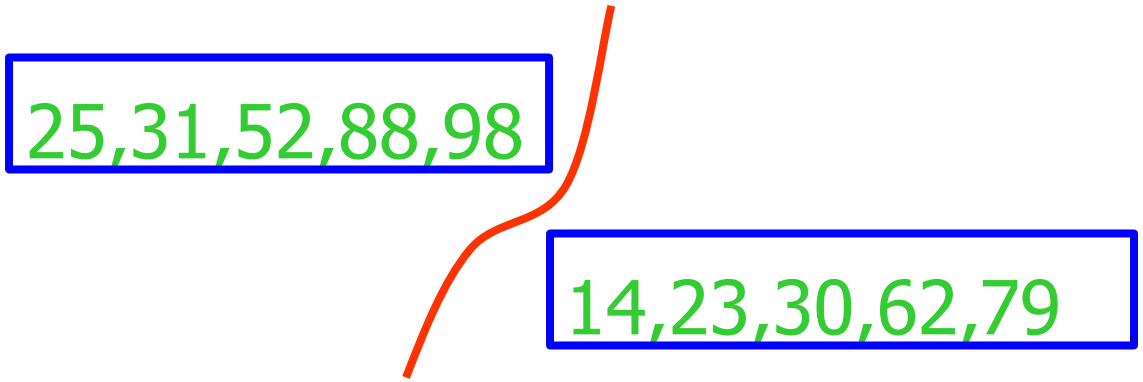
Affida una metà da ordinare ad un amico….

25,31,52,88,98

….e l'altra metà da ordinare ad un altro

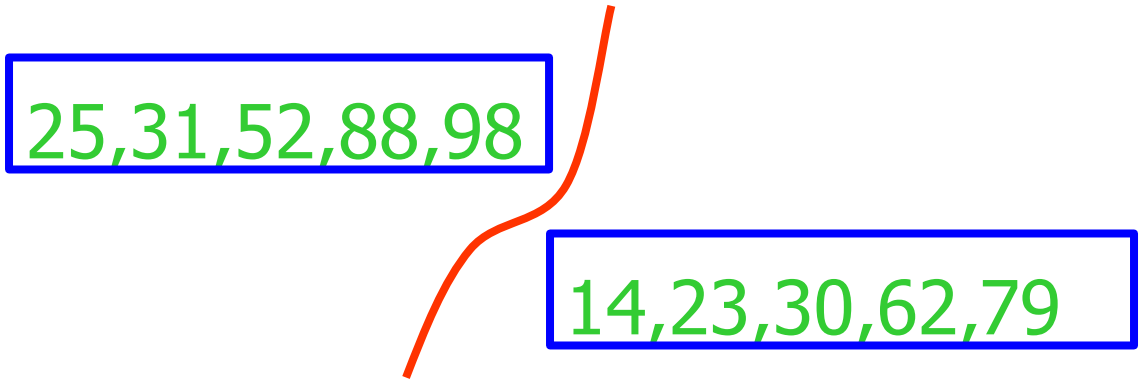
14,23,30,62,79

MergeSort



Quando i due amici hanno terminato…

MergeSort

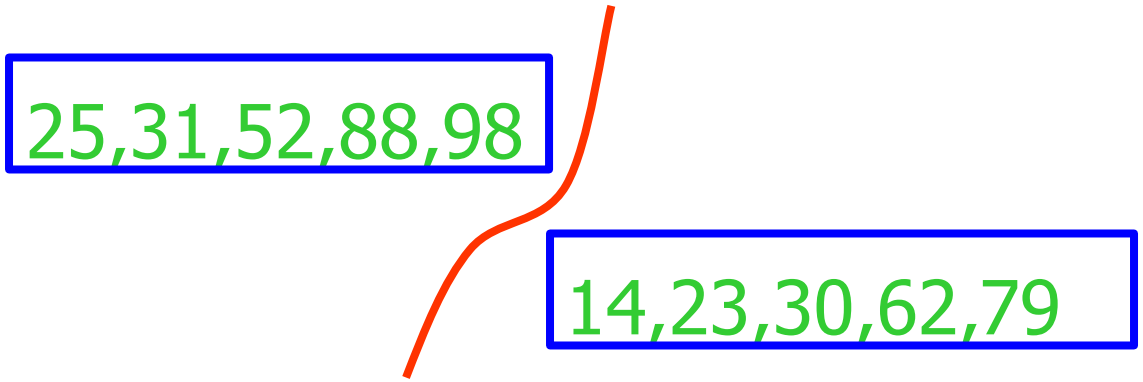


Quando i due amici hanno terminato…

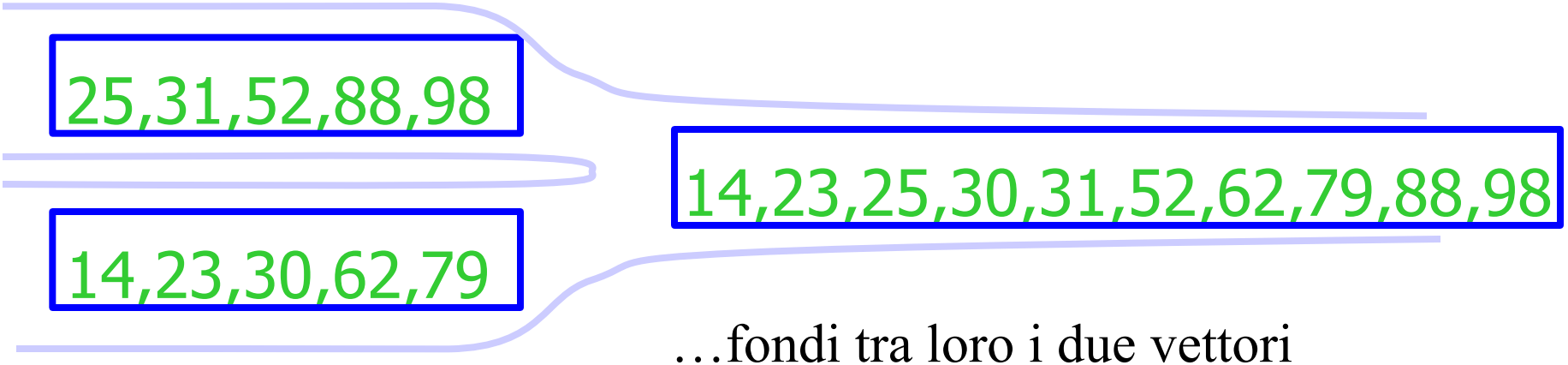
25,31,52,88,98

14,23,30,62,79

MergeSort

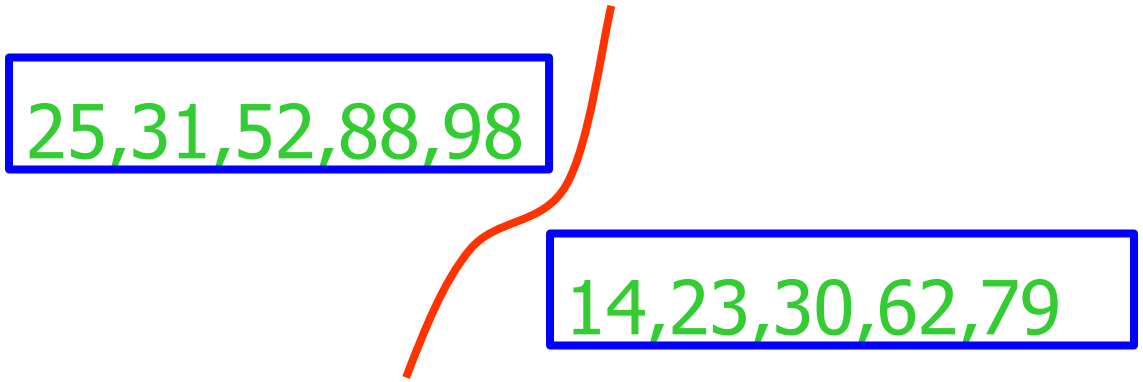


Quando i due amici hanno terminato…



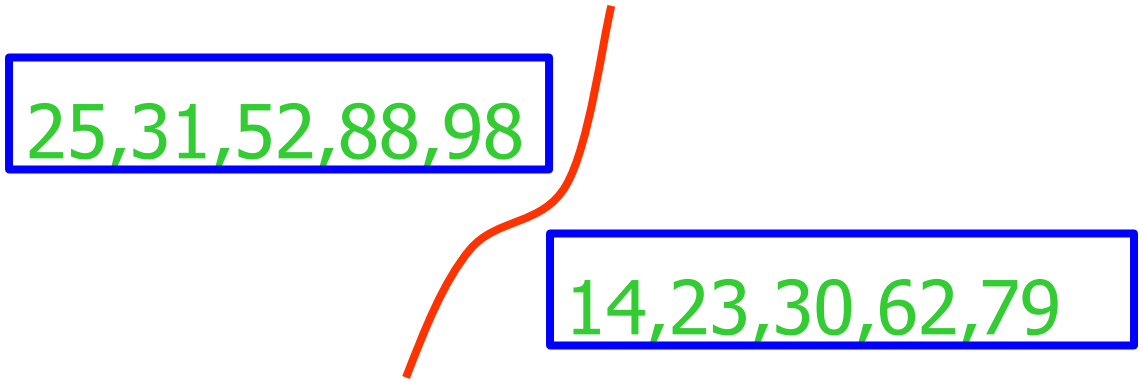
…fondi tra loro i due vettori ordinati ottenuti in modo ordinato!

MergeSort



Cosa rappresentano i due amici? In altre parole, **come implementiamo questa strategia?**

MergeSort



Cosa rappresentano i due amici? In altre parole, **come implementiamo questa strategia?**

Divide-et-Impera!

MergeSort

Divide — Dividi l'insieme da ordinare a metà

Impera — Ordina le due parti

Combina — Fondi

MergeSort - proviamo a scriverlo

Divide

Impera

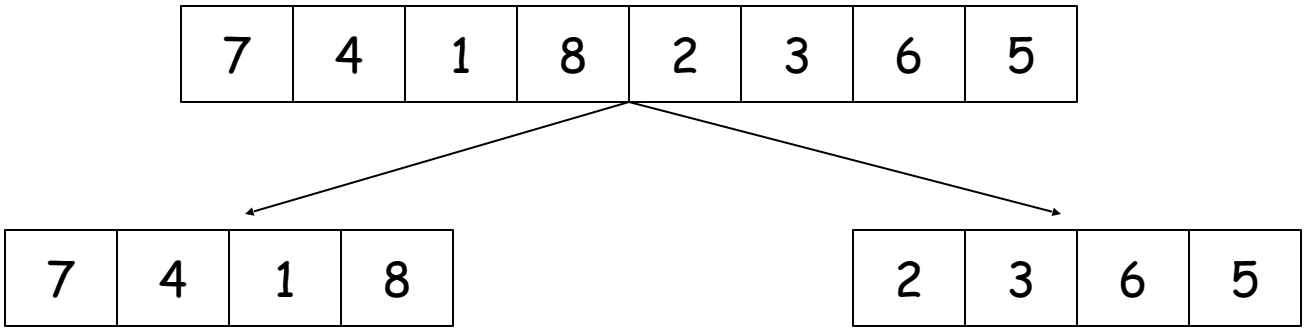
Combina

MergeSort: Divisione

7	4	1	8	2	3	6	5

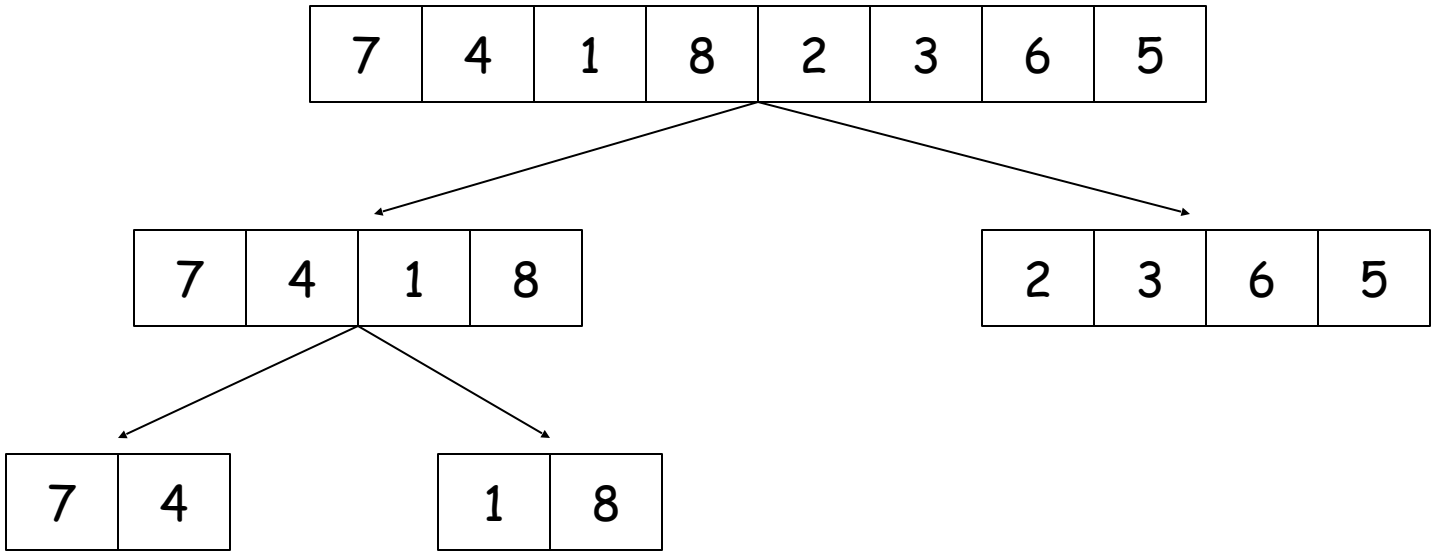
Scomponiamo il vettore in pezzi di un elemento soltanto
(**necessariamente ordinati**!)

MergeSort: Divisione



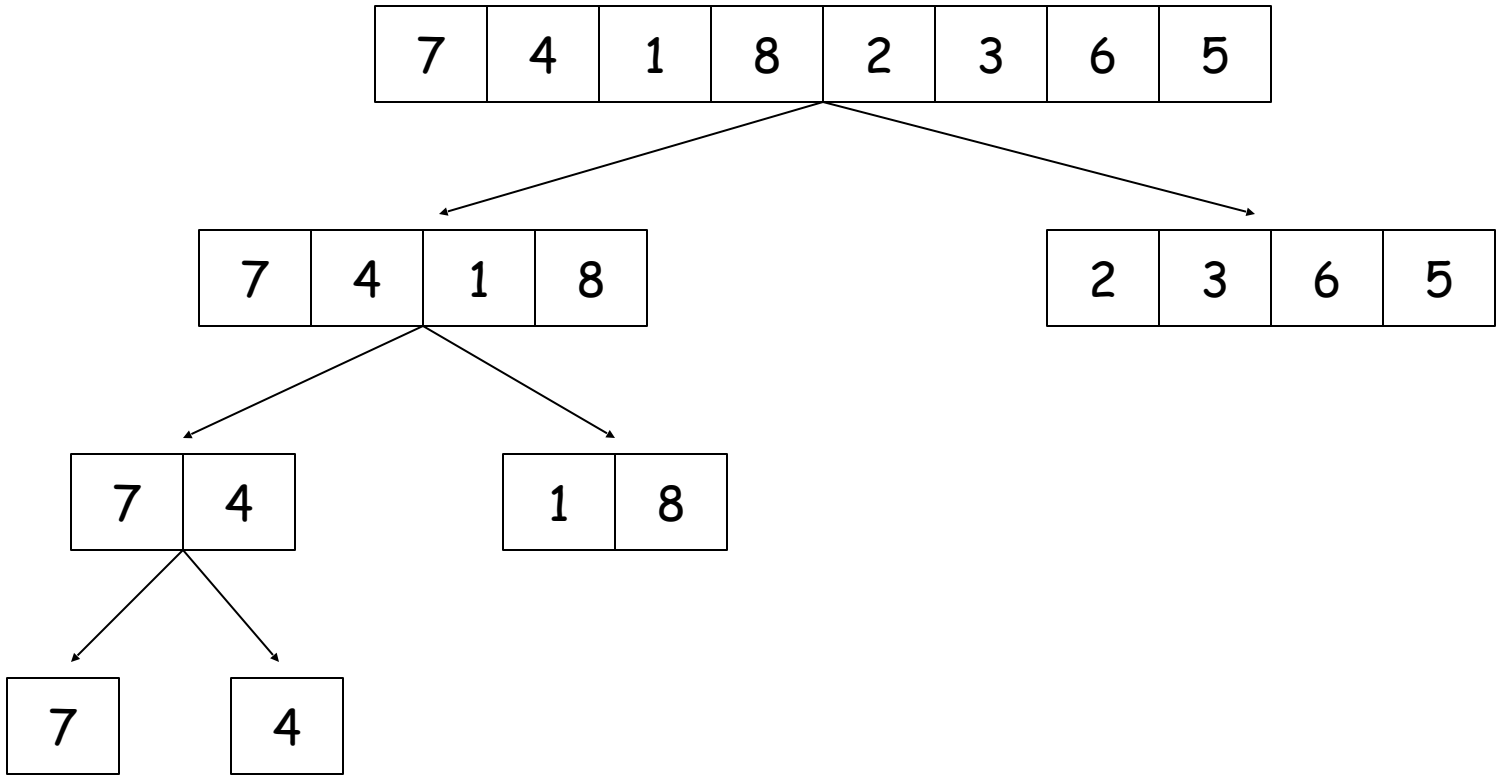
Scomponiamo il vettore in pezzi di un elemento soltanto (**necessariamente ordinati**!)

MergeSort: Divisione



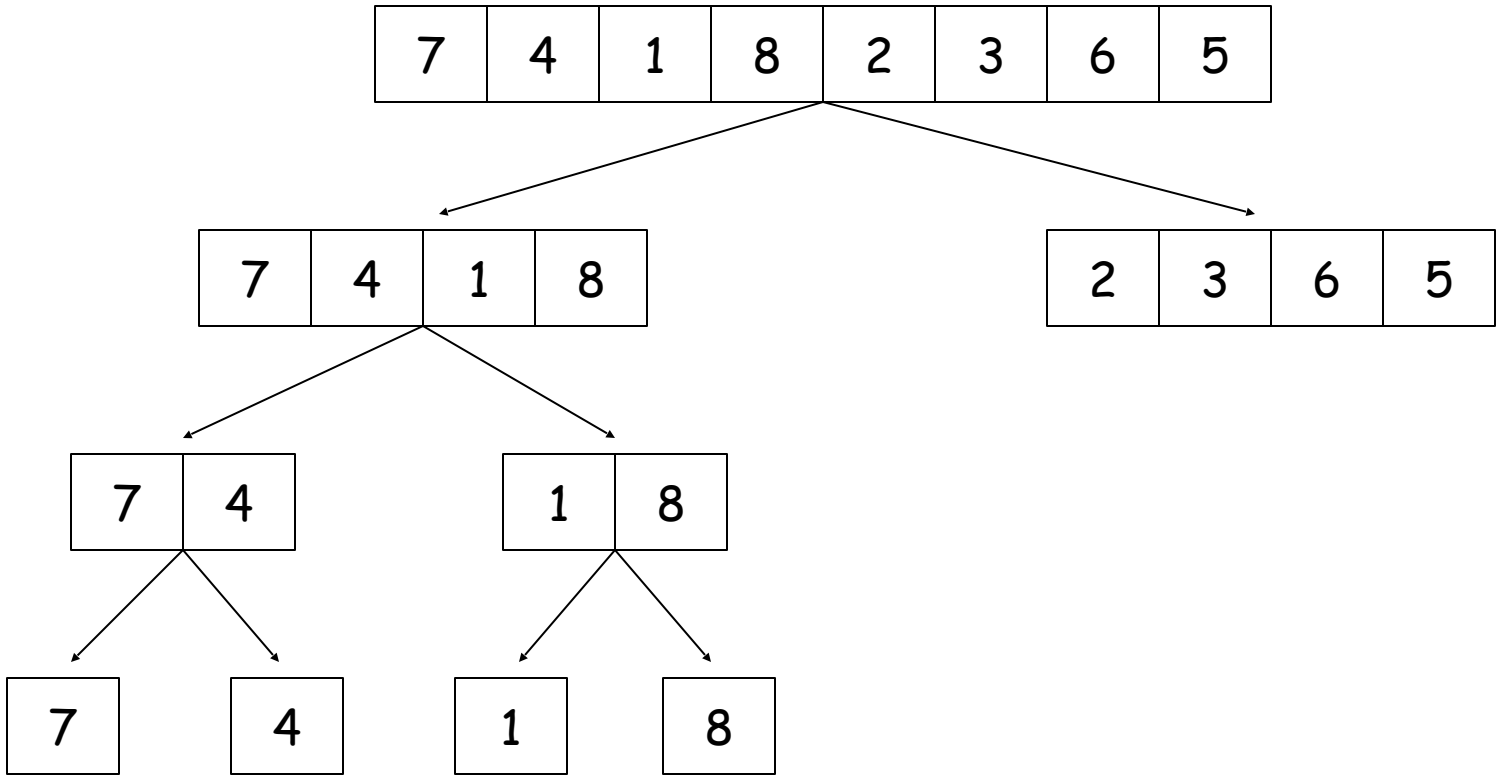
Scomponiamo il vettore in pezzi di un elemento soltanto (**necessariamente ordinati**!)

MergeSort: Divisione



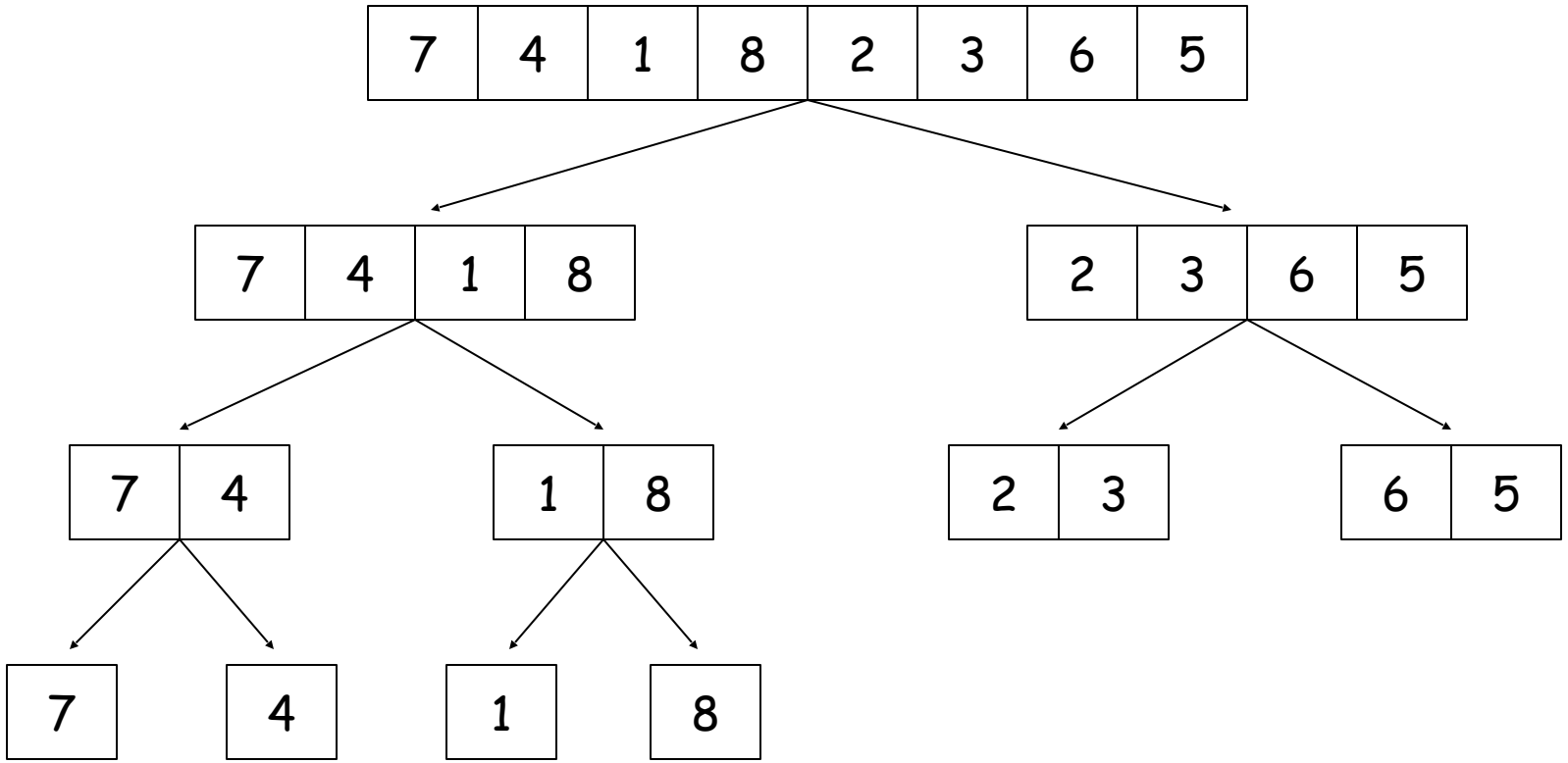
Scomponiamo il vettore in pezzi di un elemento soltanto (**necessariamente ordinati**!)

MergeSort: Divisione



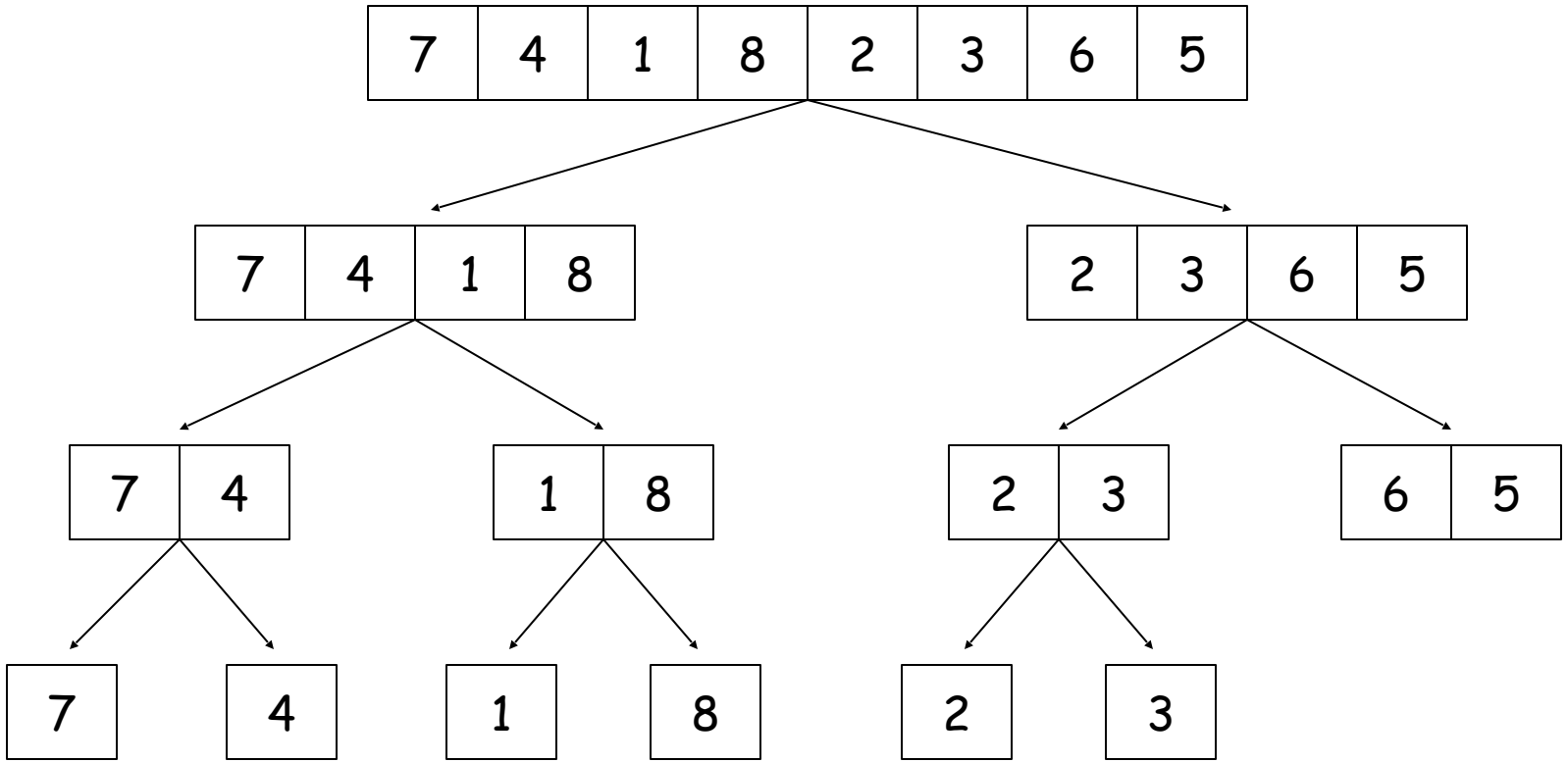
Scomponiamo il vettore in pezzi di un elemento soltanto (**necessariamente ordinati**!)

MergeSort: Divisione



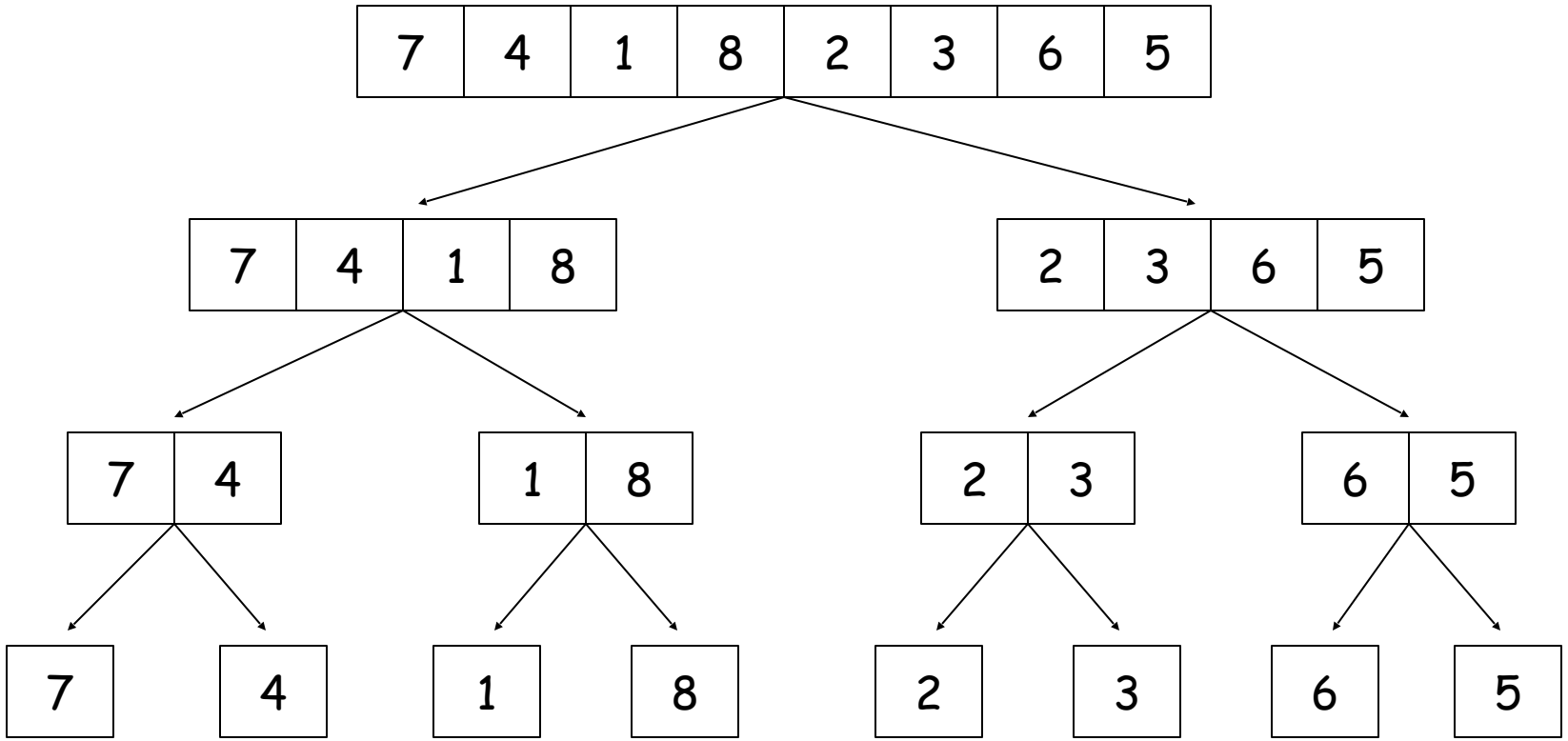
Scomponiamo il vettore in pezzi di un elemento soltanto (**necessariamente ordinati**!)

MergeSort: Divisione



Scomponiamo il vettore in pezzi di un elemento soltanto (**necessariamente ordinati**!)

MergeSort: Divisione

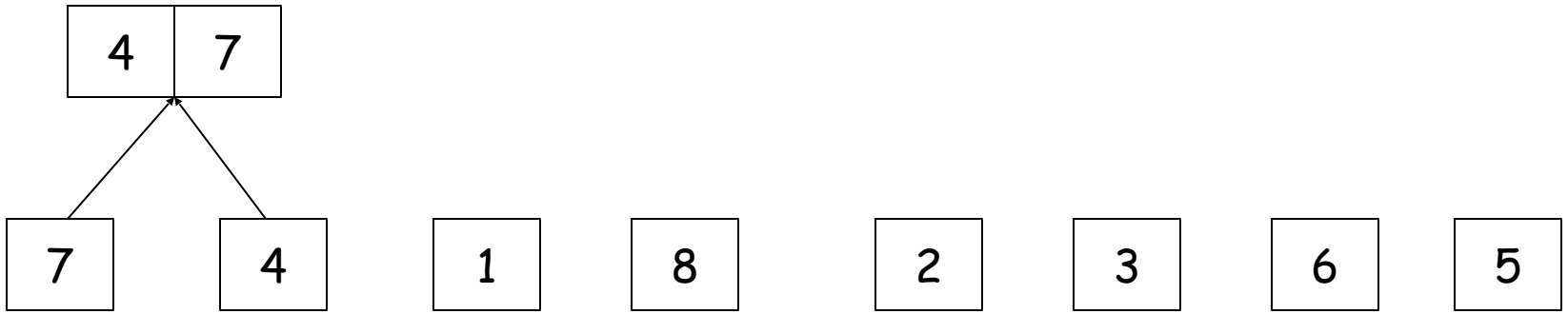


Scomponiamo il vettore in pezzi di un elemento soltanto (**necessariamente ordinati**!)

MergeSort: Fusione

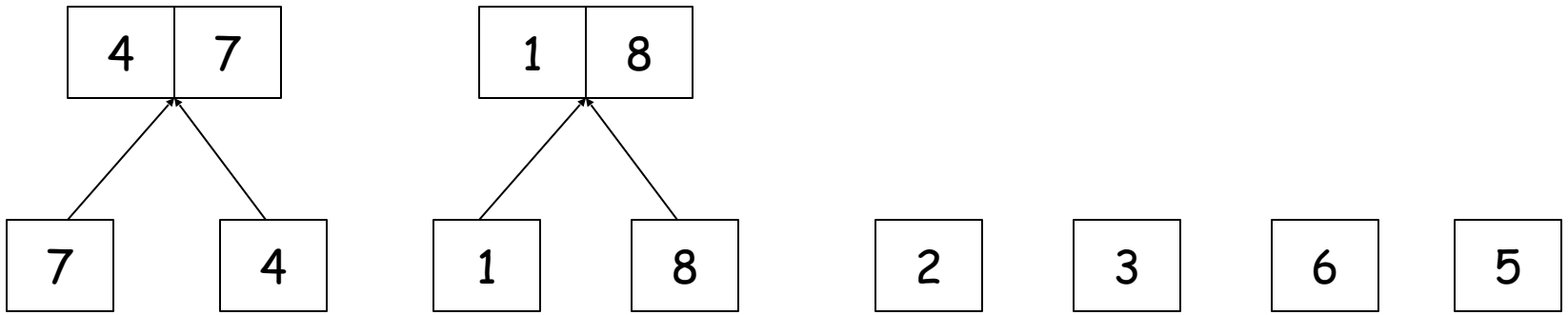
Fondiamo tra loro i segmenti ordinati in segmenti più grandi, preservando l'ordine, fino a **ricostruire il vettore**

MergeSort: Fusione



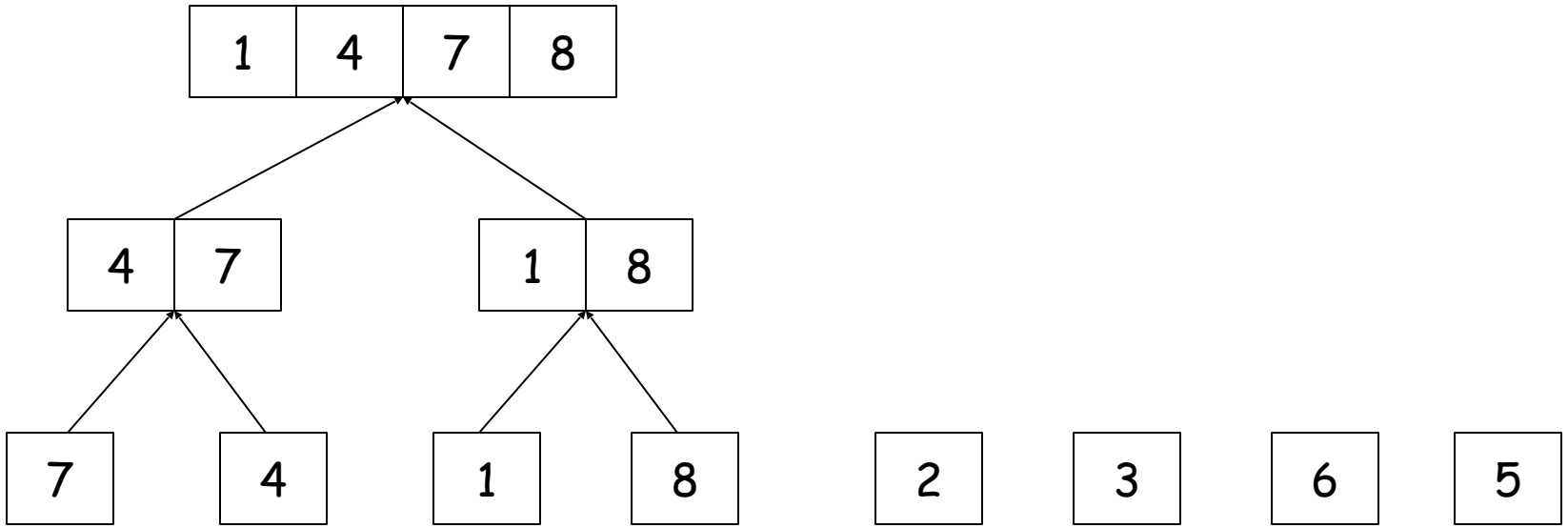
Fondiamo tra loro i segmenti ordinati in segmenti più grandi, preservando l'ordine, fino a **ricostruire il vettore**

MergeSort: Fusione



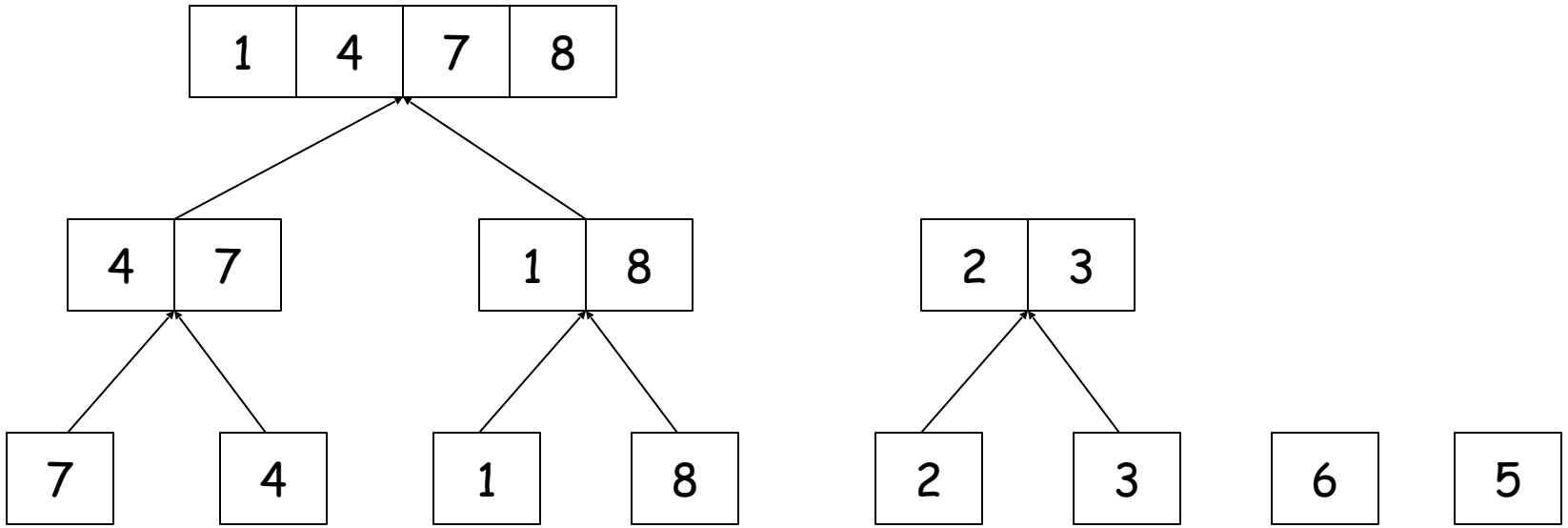
Fondiamo tra loro i segmenti ordinati in segmenti più grandi, preservando l'ordine, fino a **ricostruire il vettore**

MergeSort: Fusione



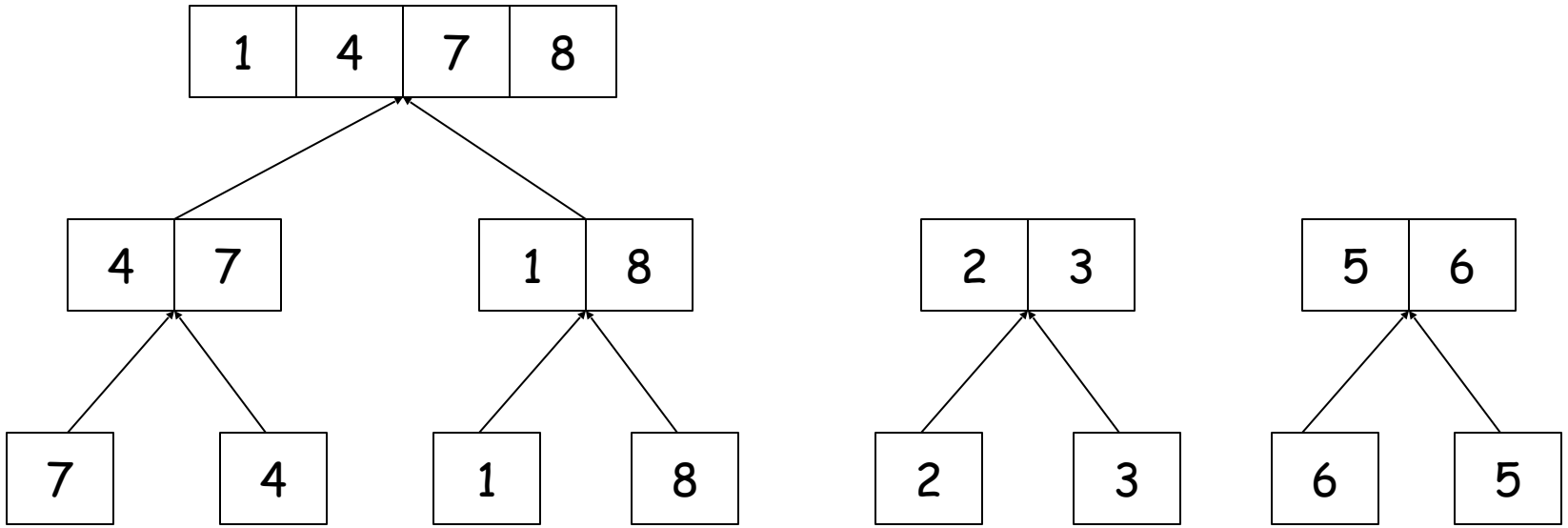
Fondiamo tra loro i segmenti ordinati in segmenti più grandi, preservando l'ordine, fino a **ricostruire il vettore**

MergeSort: Fusione



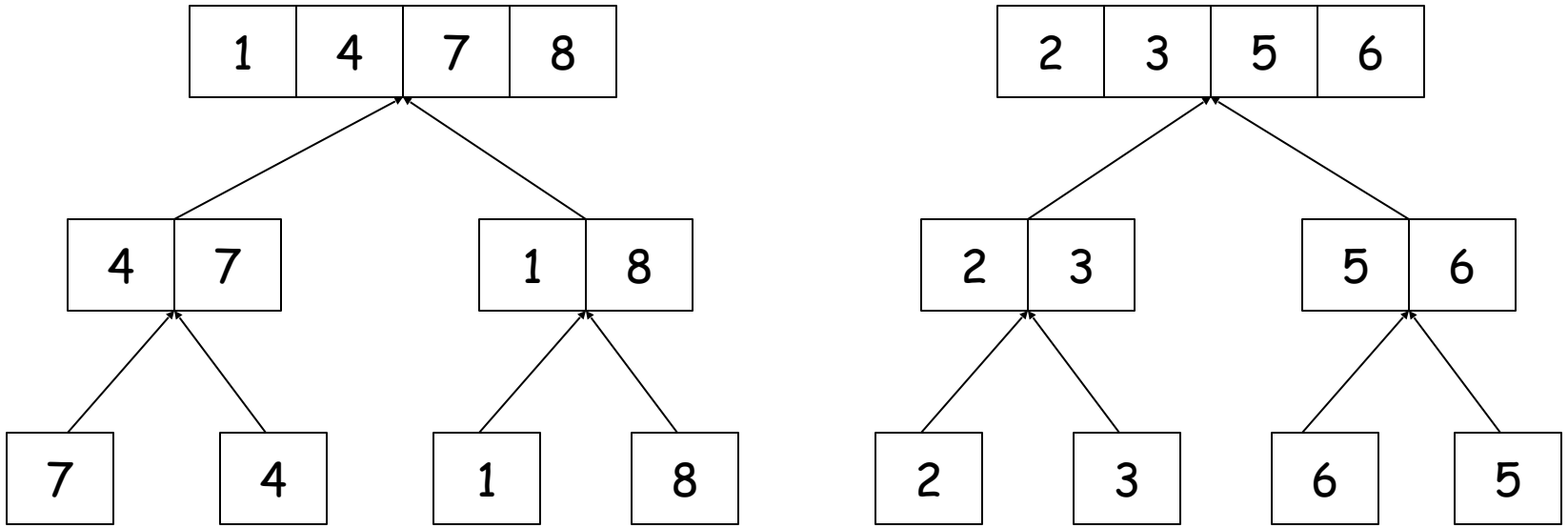
Fondiamo tra loro i segmenti ordinati in segmenti più grandi, preservando l'ordine, fino a **ricostruire il vettore**

MergeSort: Fusione



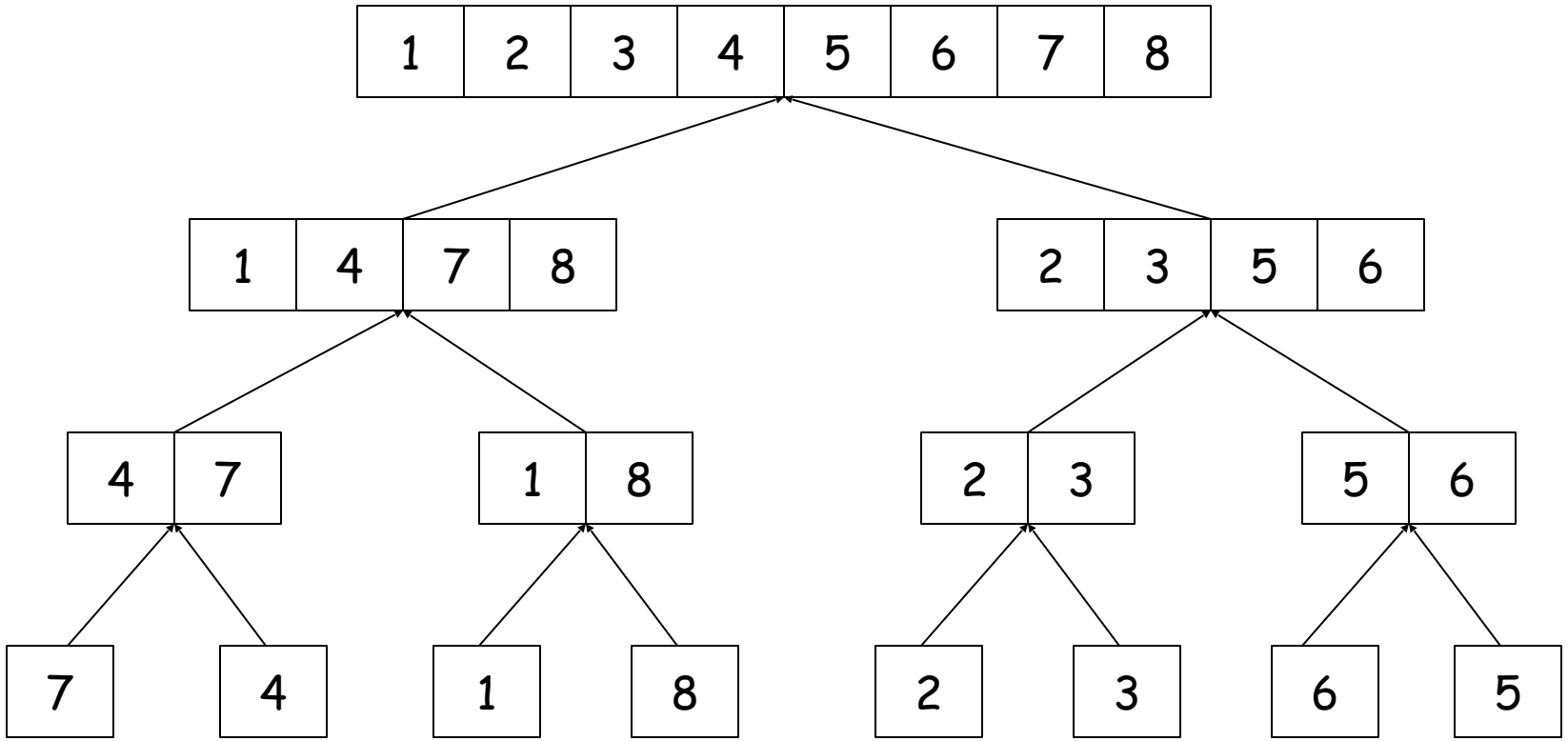
Fondiamo tra loro i segmenti ordinati in segmenti più grandi, preservando l'ordine, fino a **ricostruire il vettore**

MergeSort: Fusione



Fondiamo tra loro i segmenti ordinati in segmenti più grandi, preservando l'ordine, fino a **ricostruire il vettore**

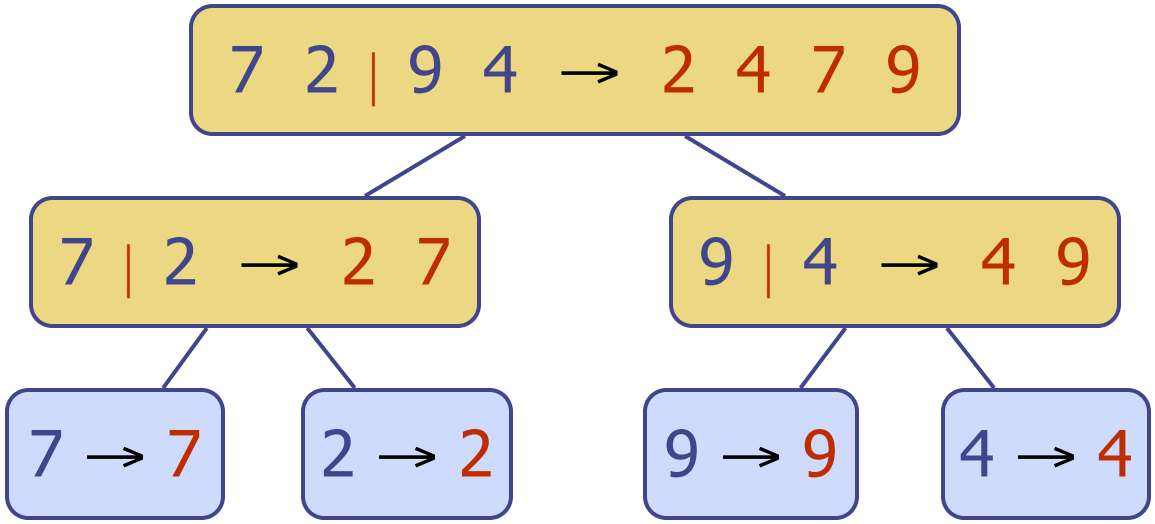
MergeSort: Fusione



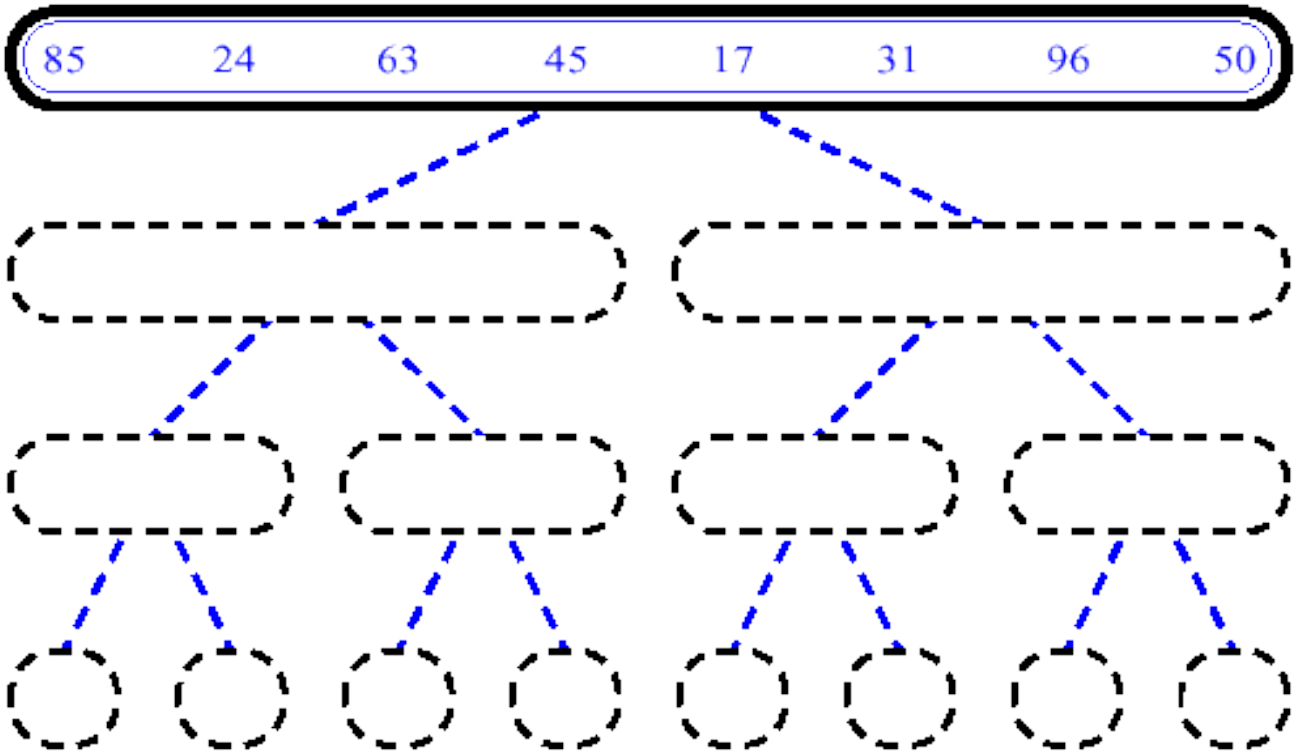
Fondiamo tra loro i segmenti ordinati in segmenti più grandi, preservando l'ordine, fino a **ricostruire il vettore**

MergeSort Tree

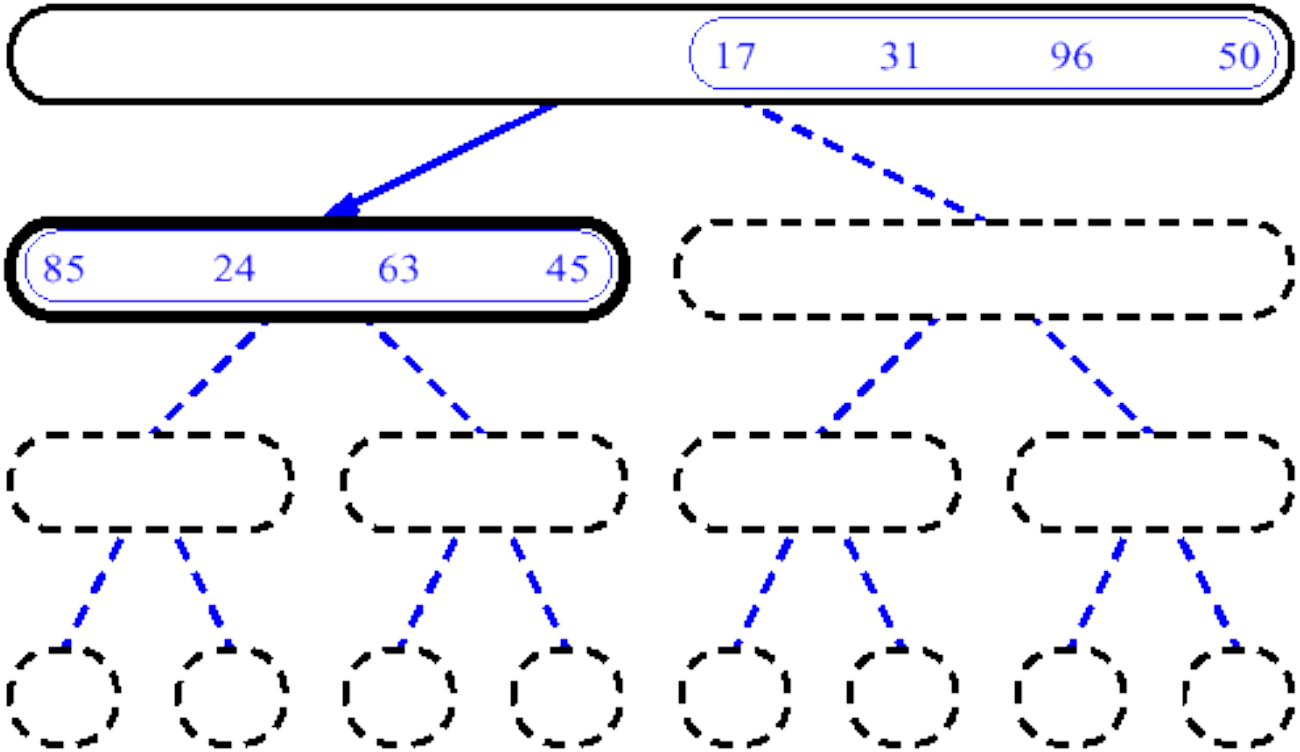
- Se proviamo ad eseguire questa semplice strategia, notiamo che può essere **rappresentata da un albero**
  - Ogni nodo rappresenta una chiamata ricorsiva
  - La radice rappresenta la chiamata iniziale
  - Le foglie rappresentano chiamate su sequenze di dimensione 1



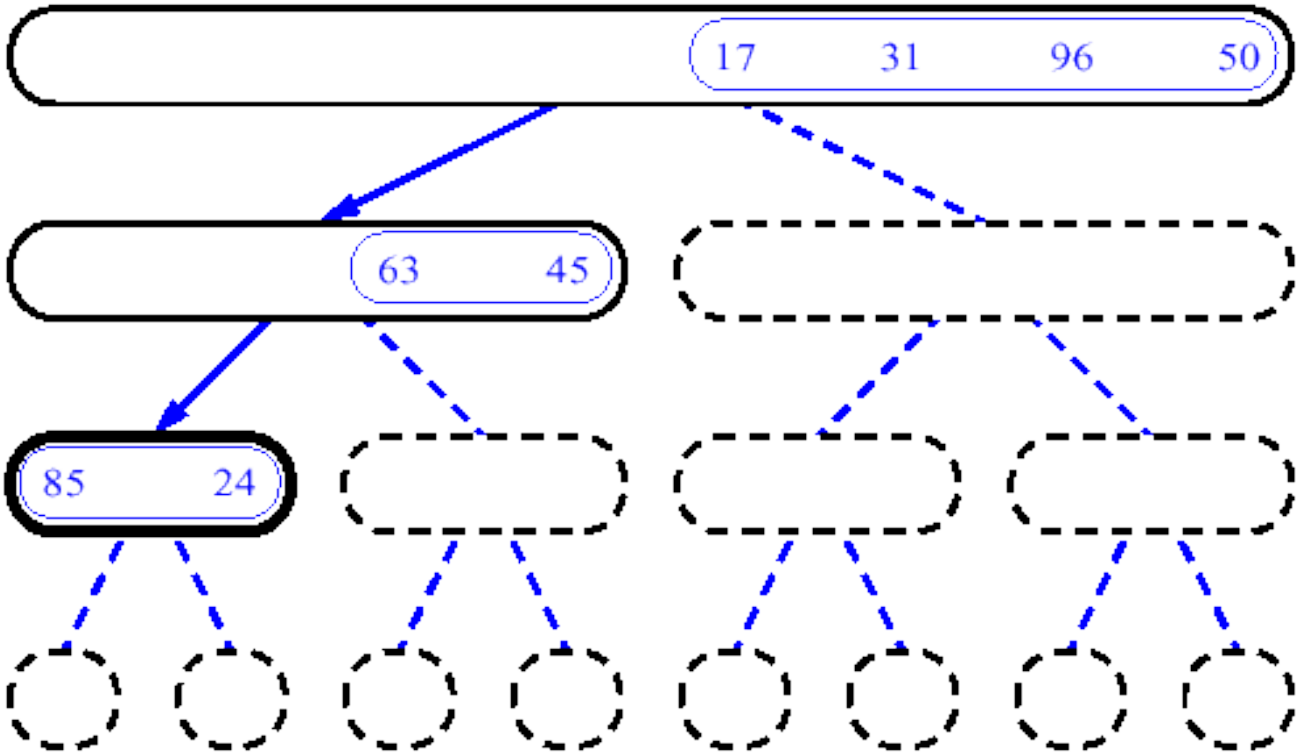
MergeSort



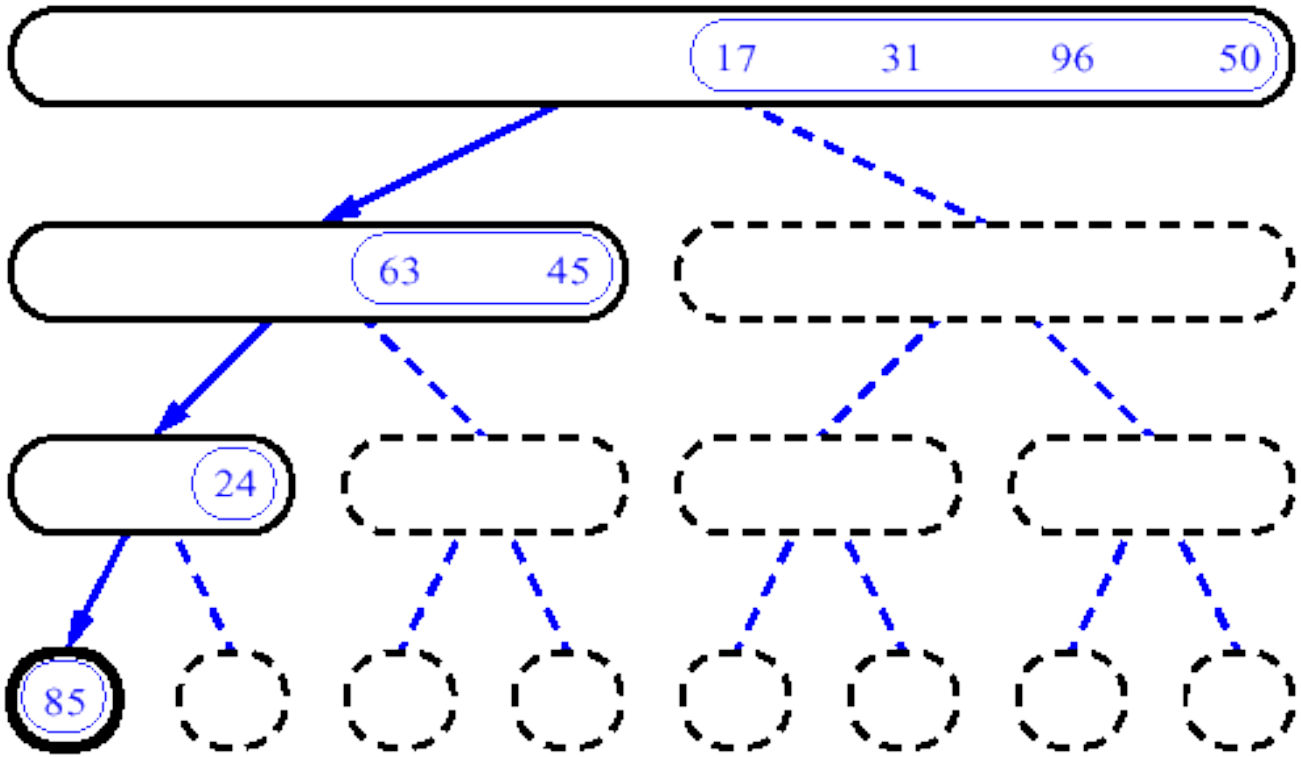
MergeSort



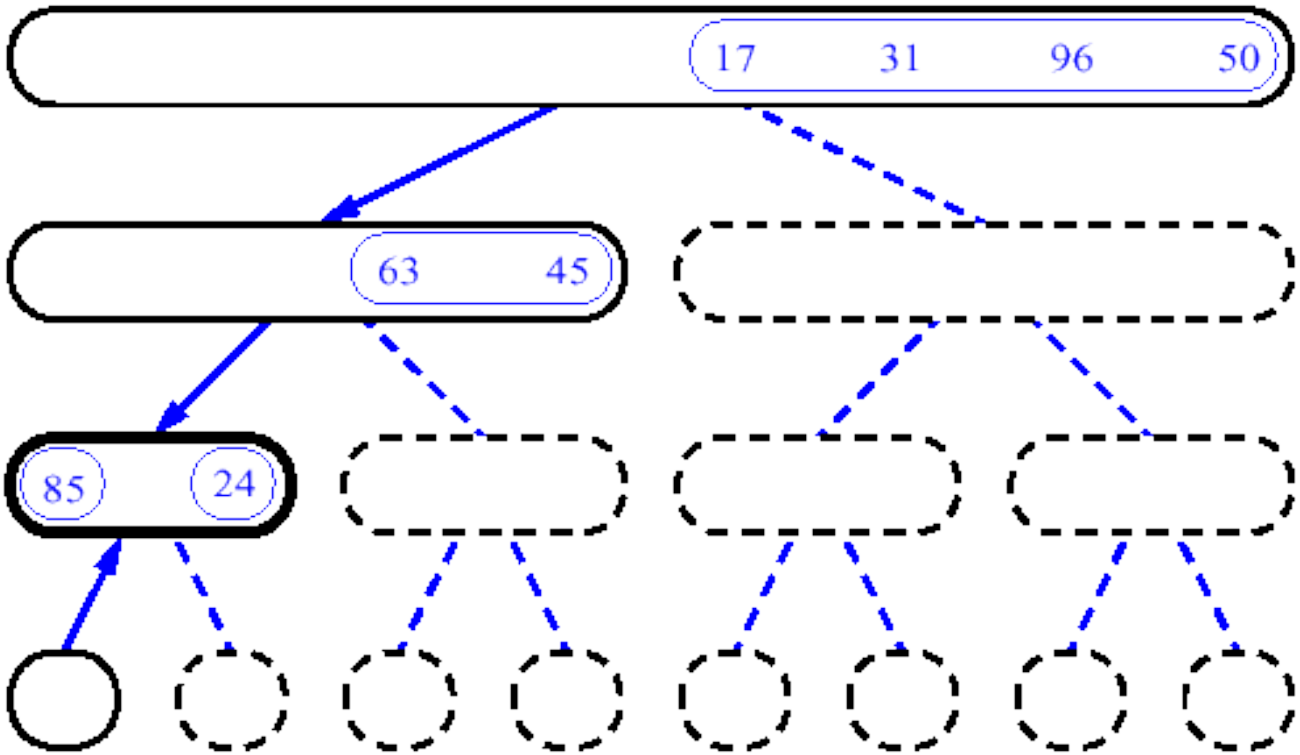
MergeSort



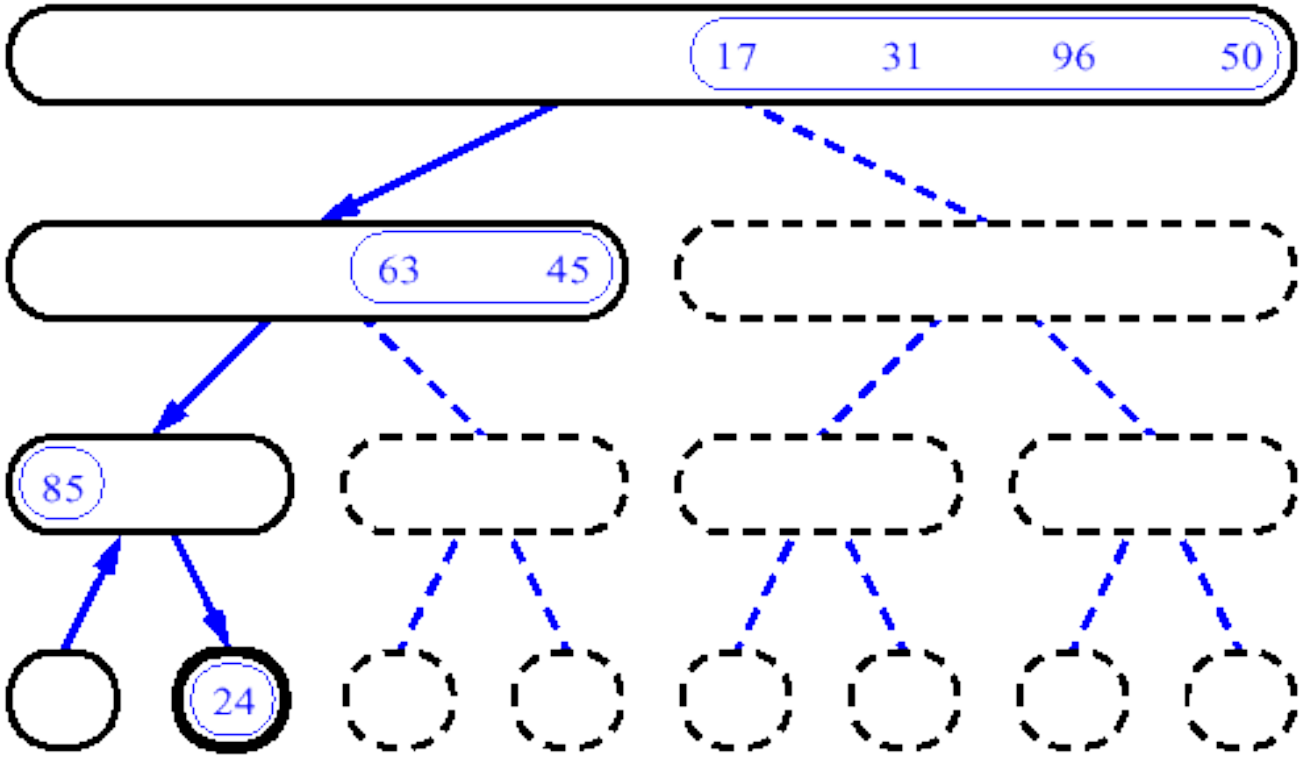
MergeSort



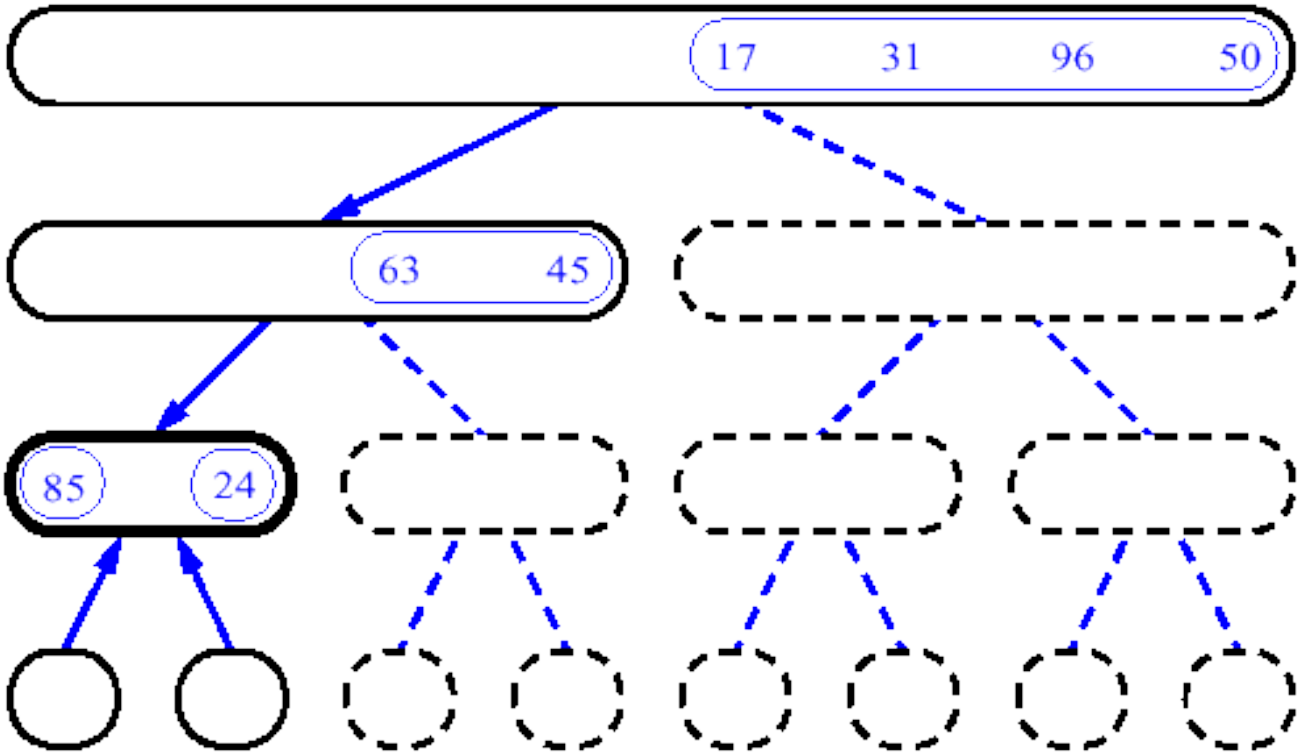
MergeSort



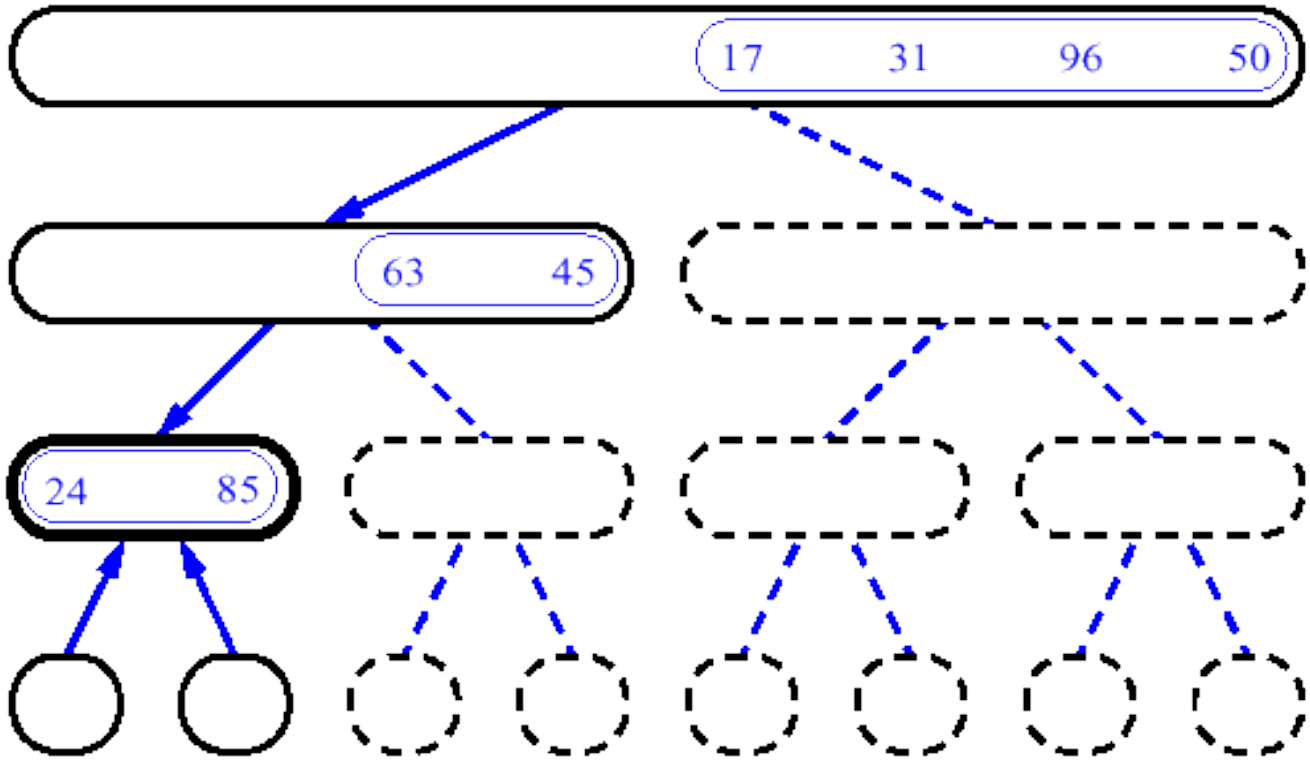
MergeSort



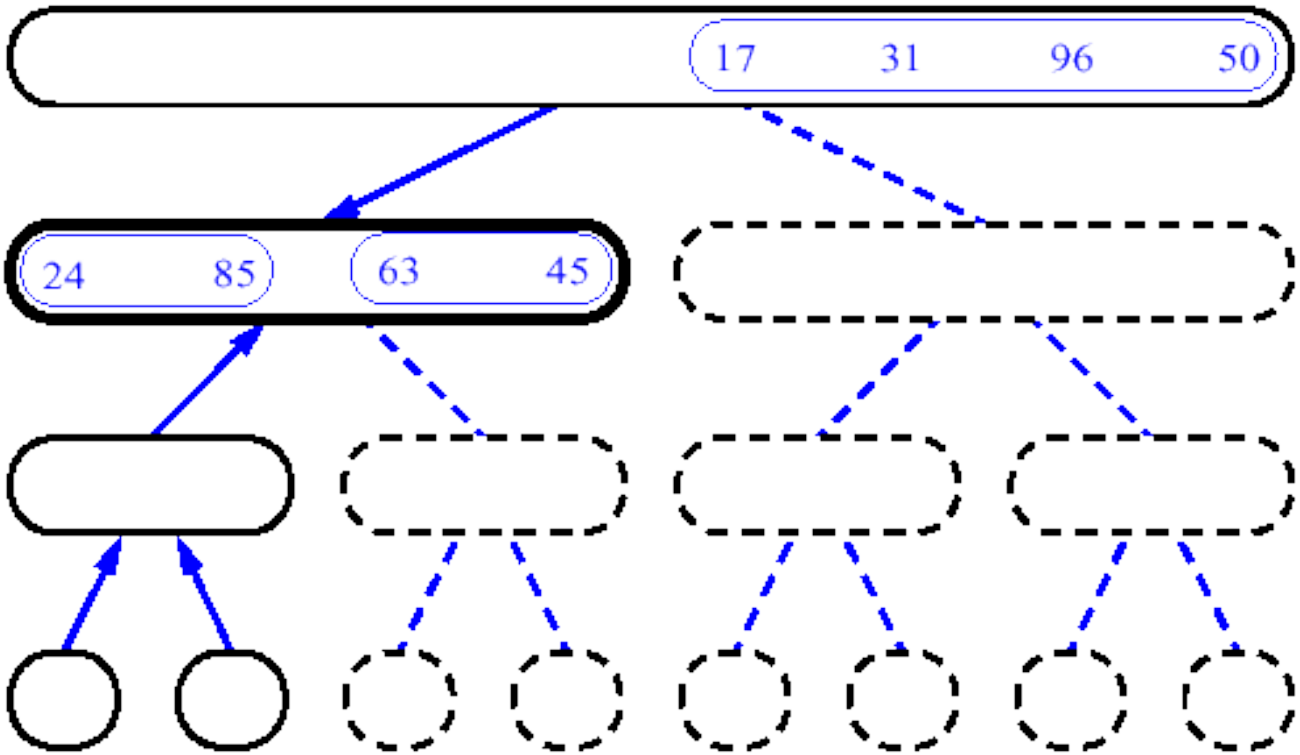
MergeSort



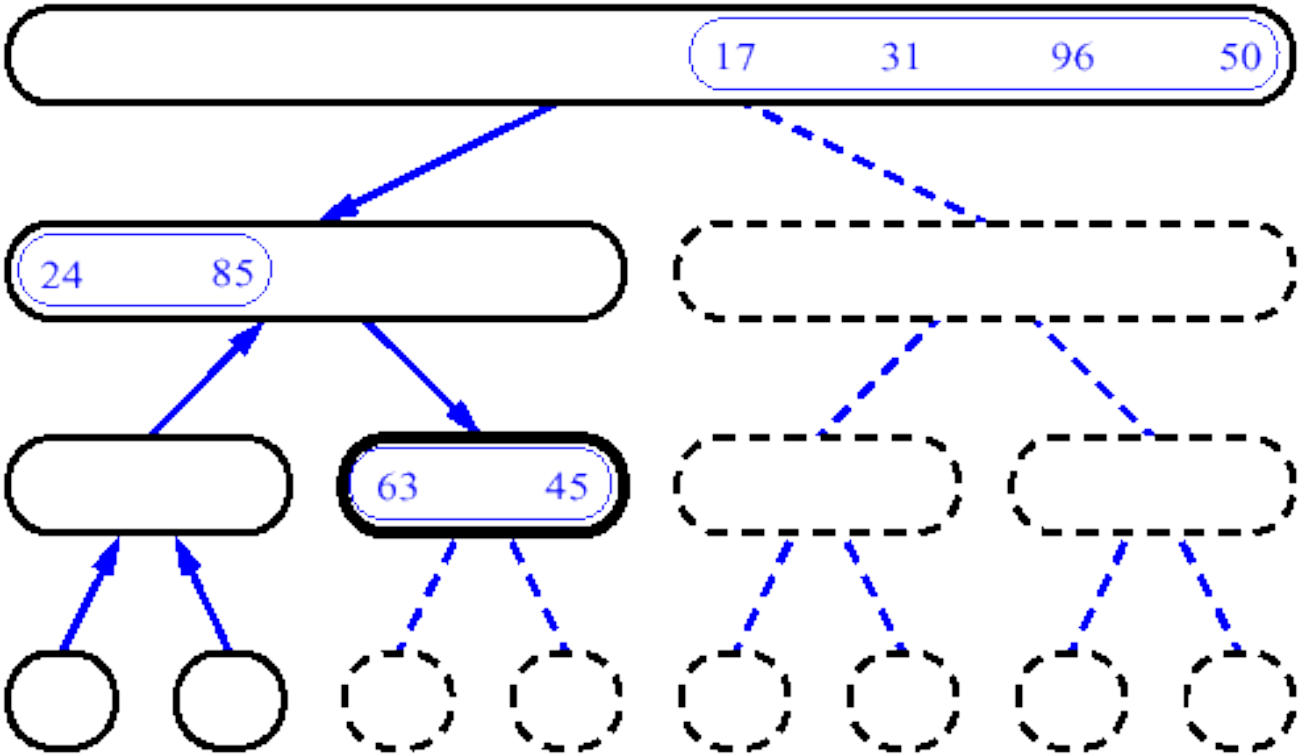
MergeSort



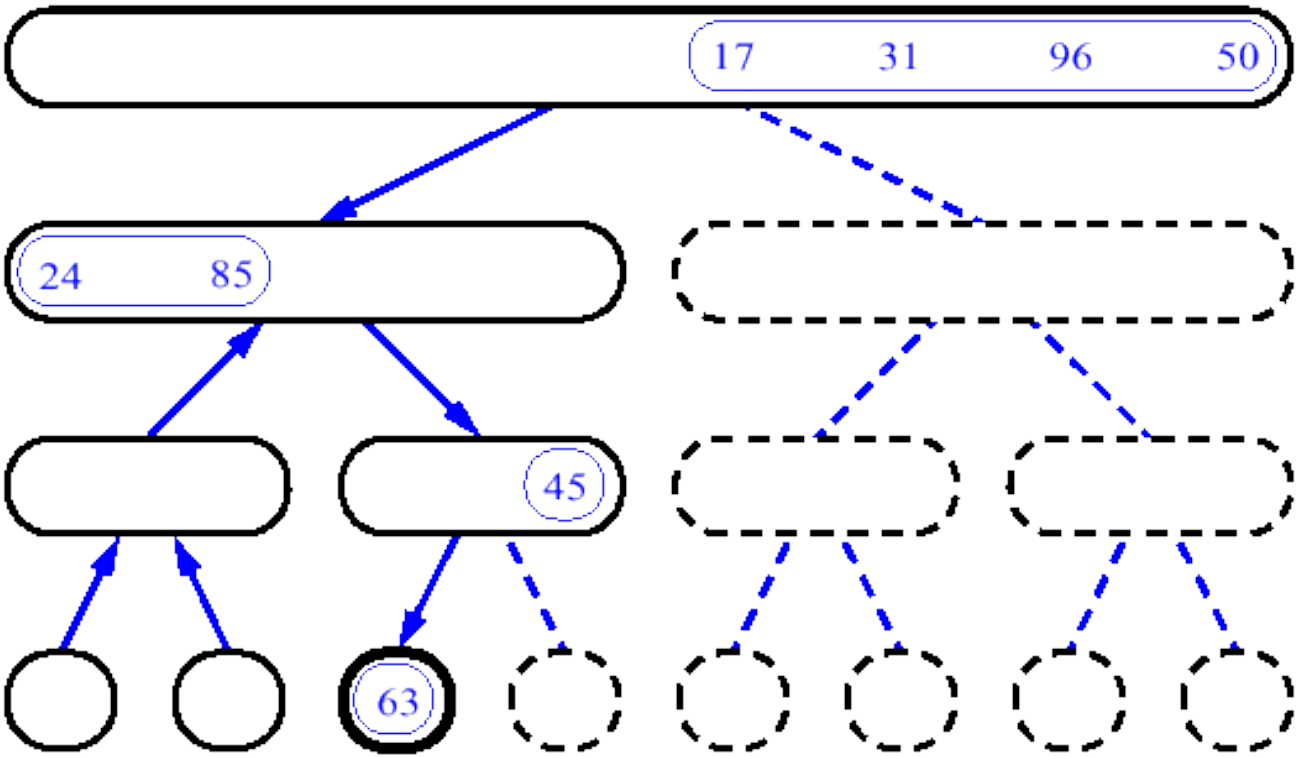
MergeSort



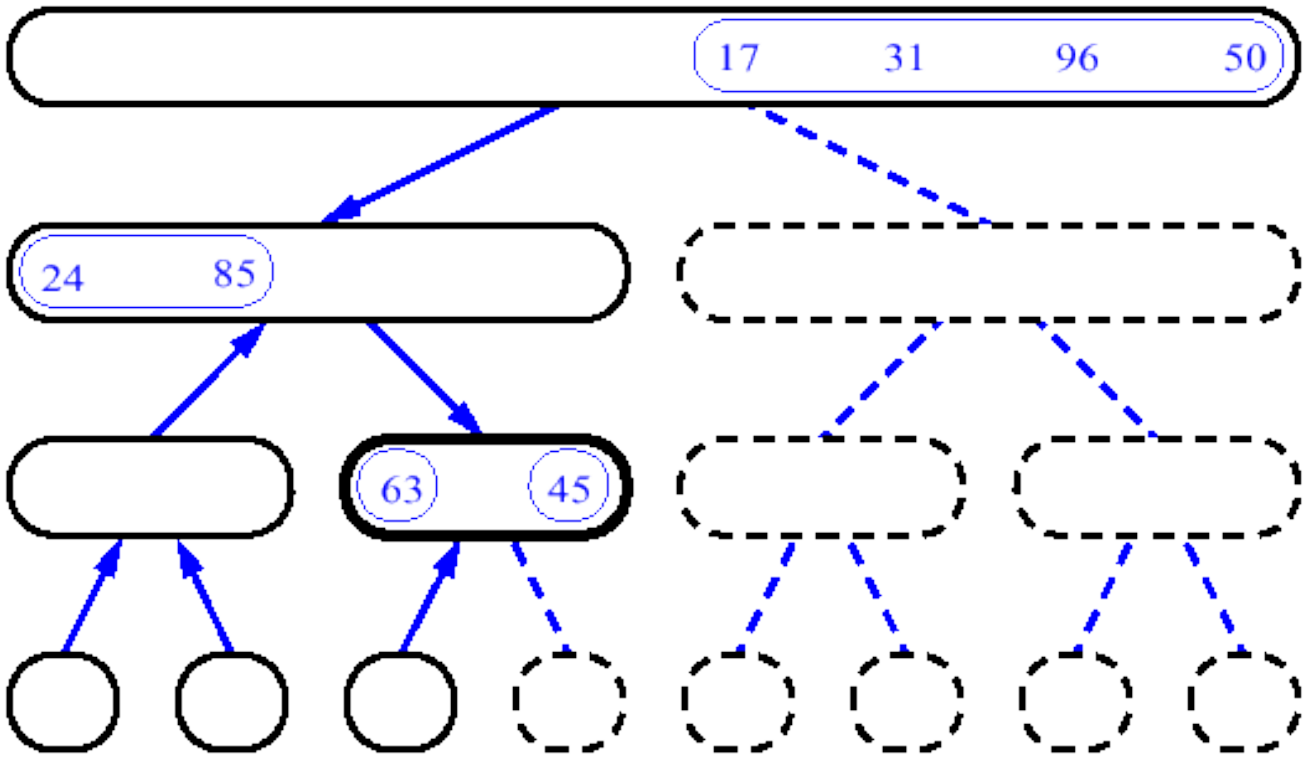
MergeSort



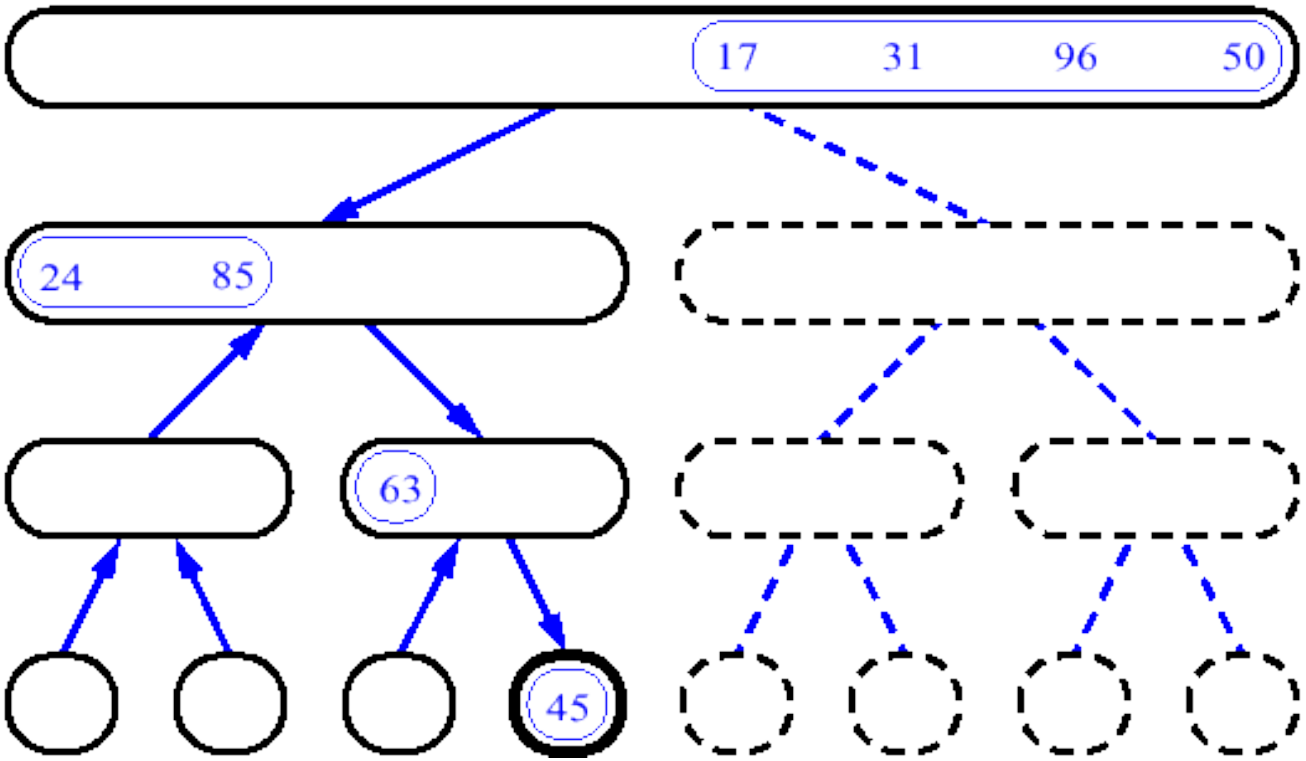
MergeSort



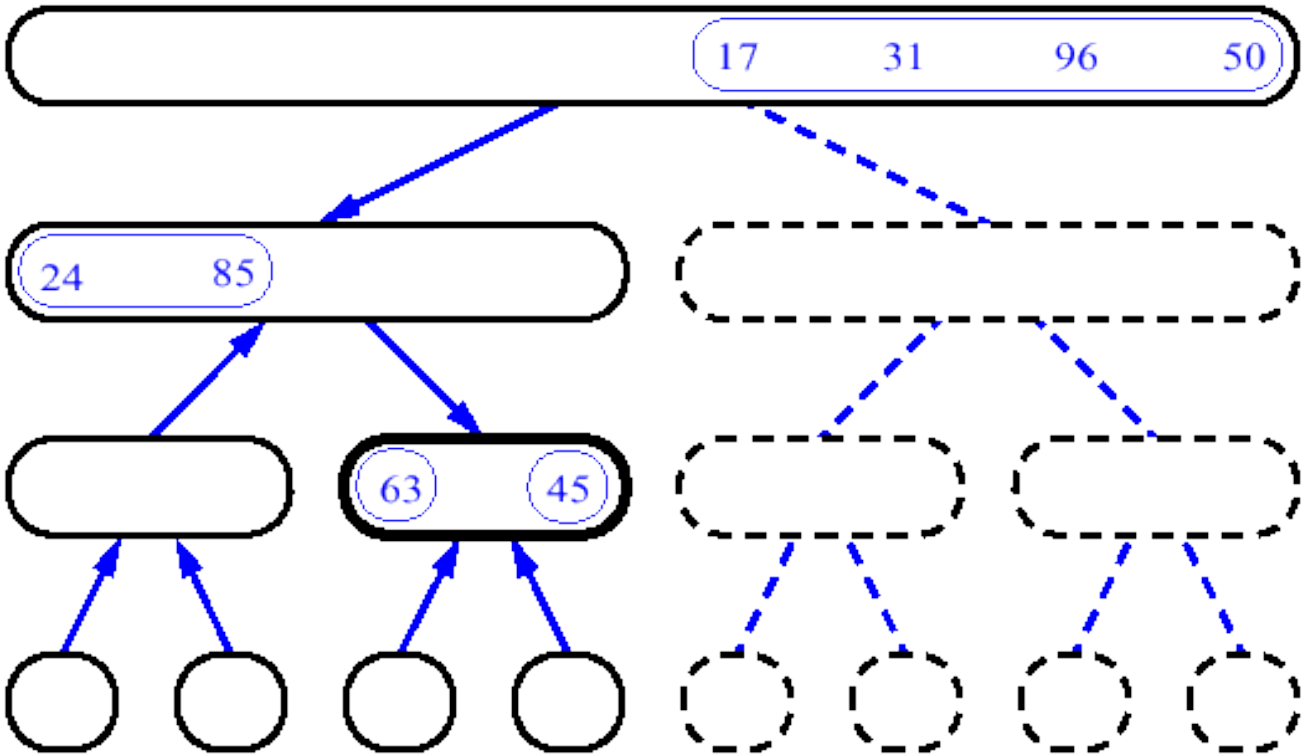
MergeSort



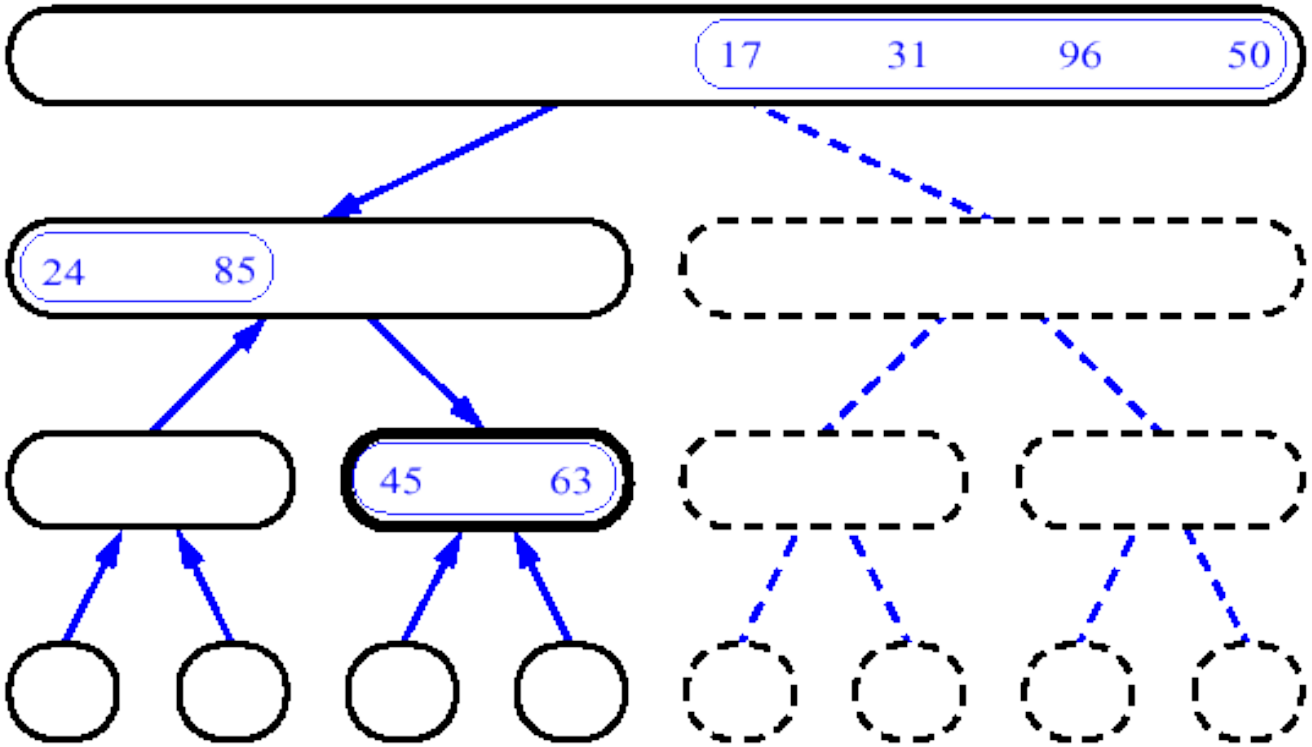
MergeSort



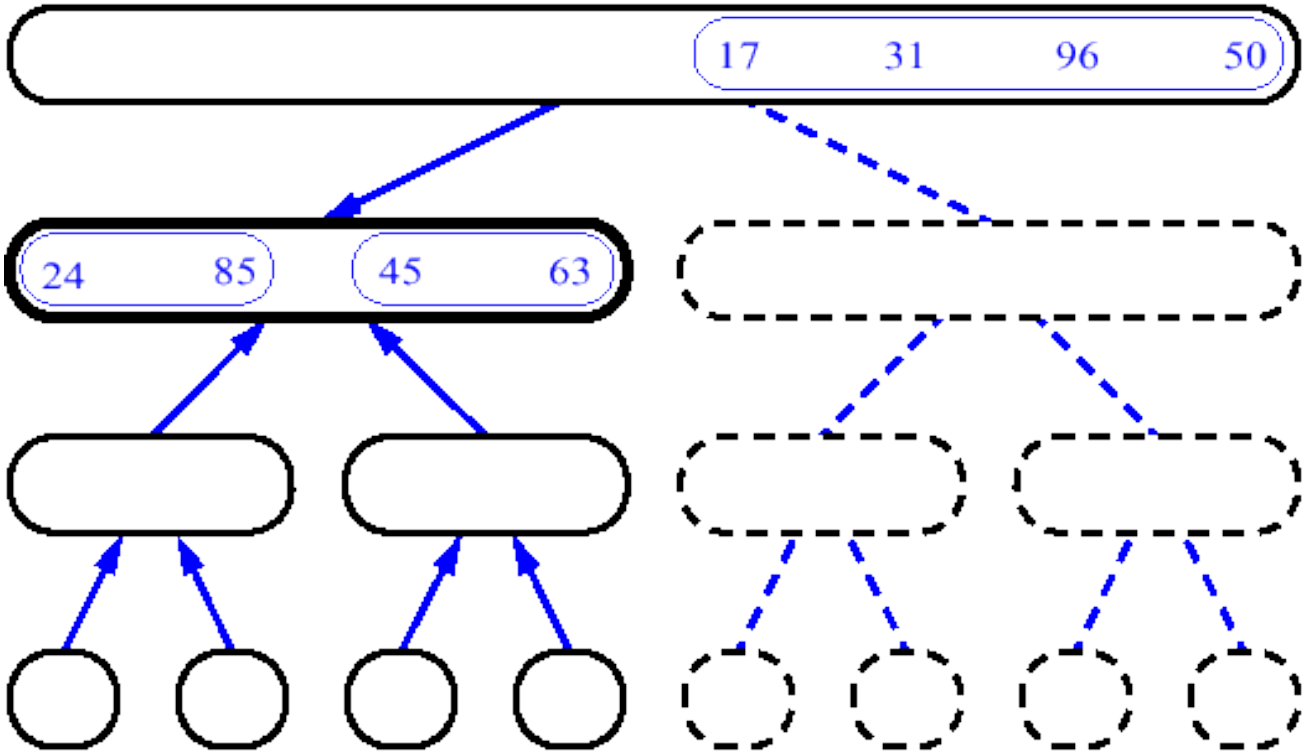
MergeSort



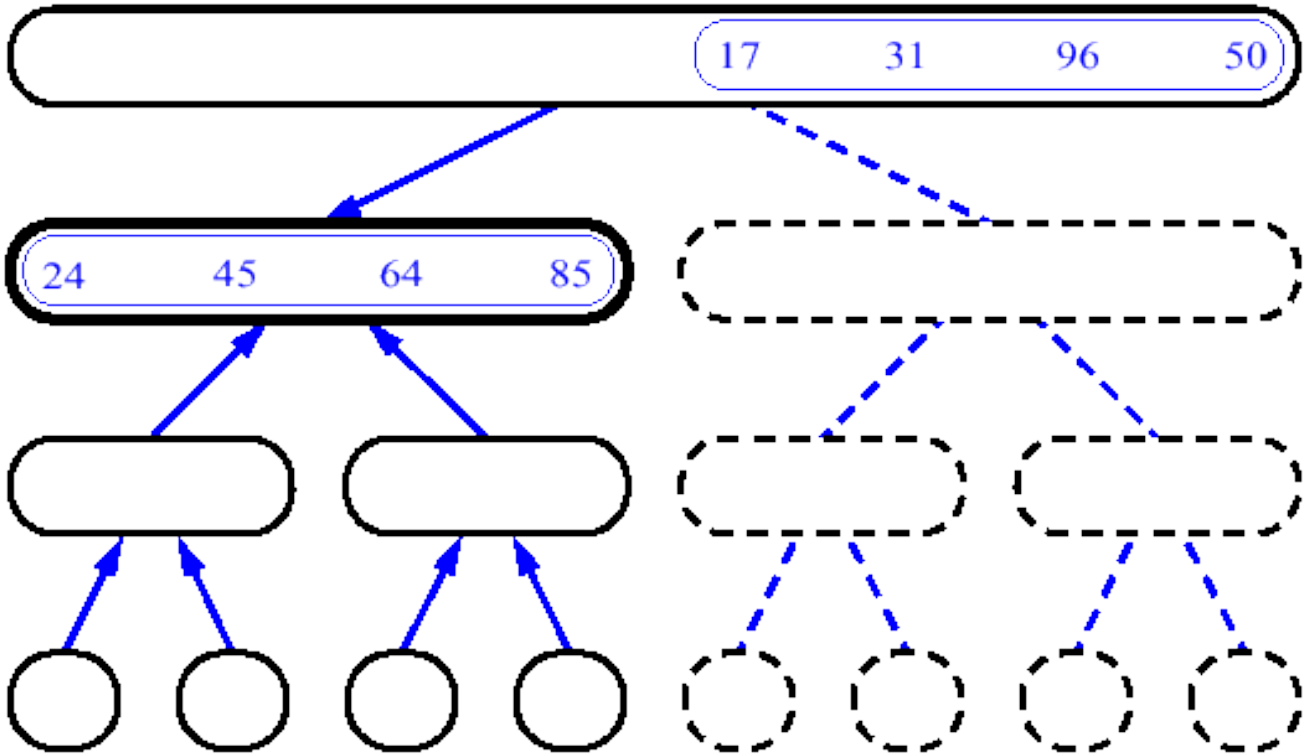
MergeSort



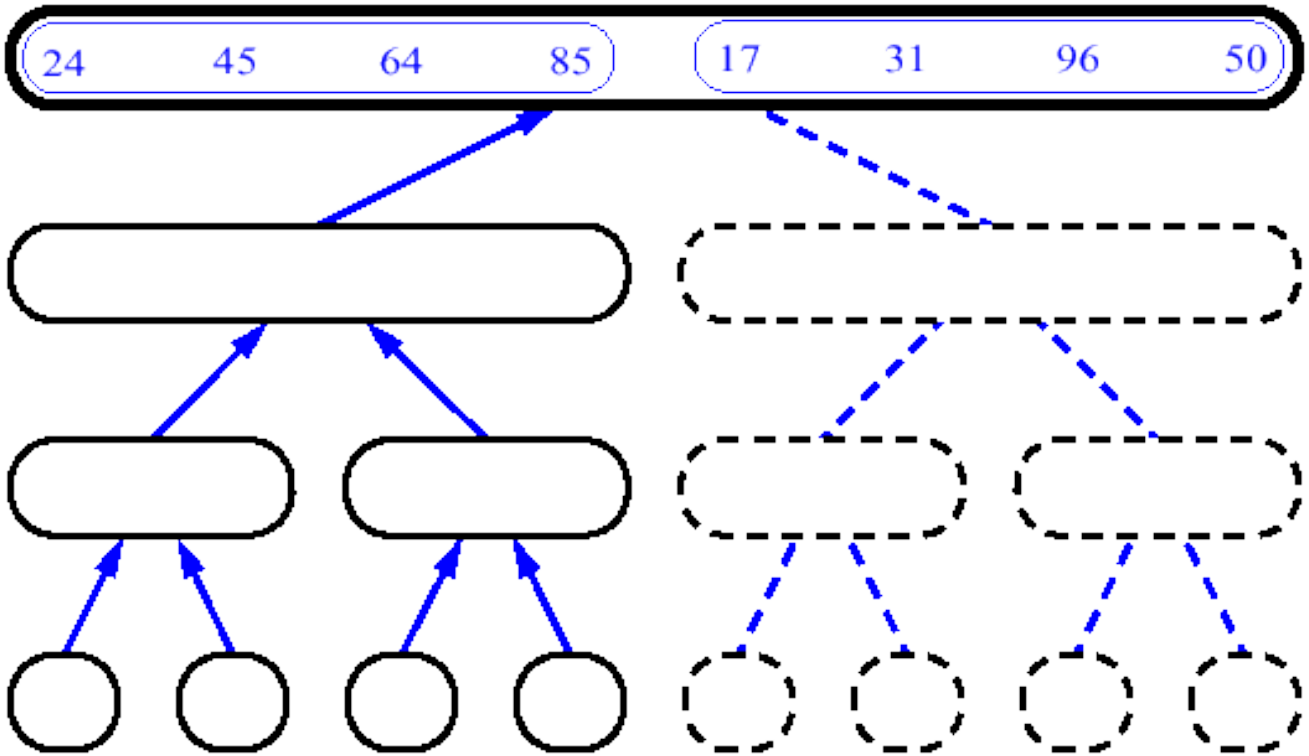
MergeSort



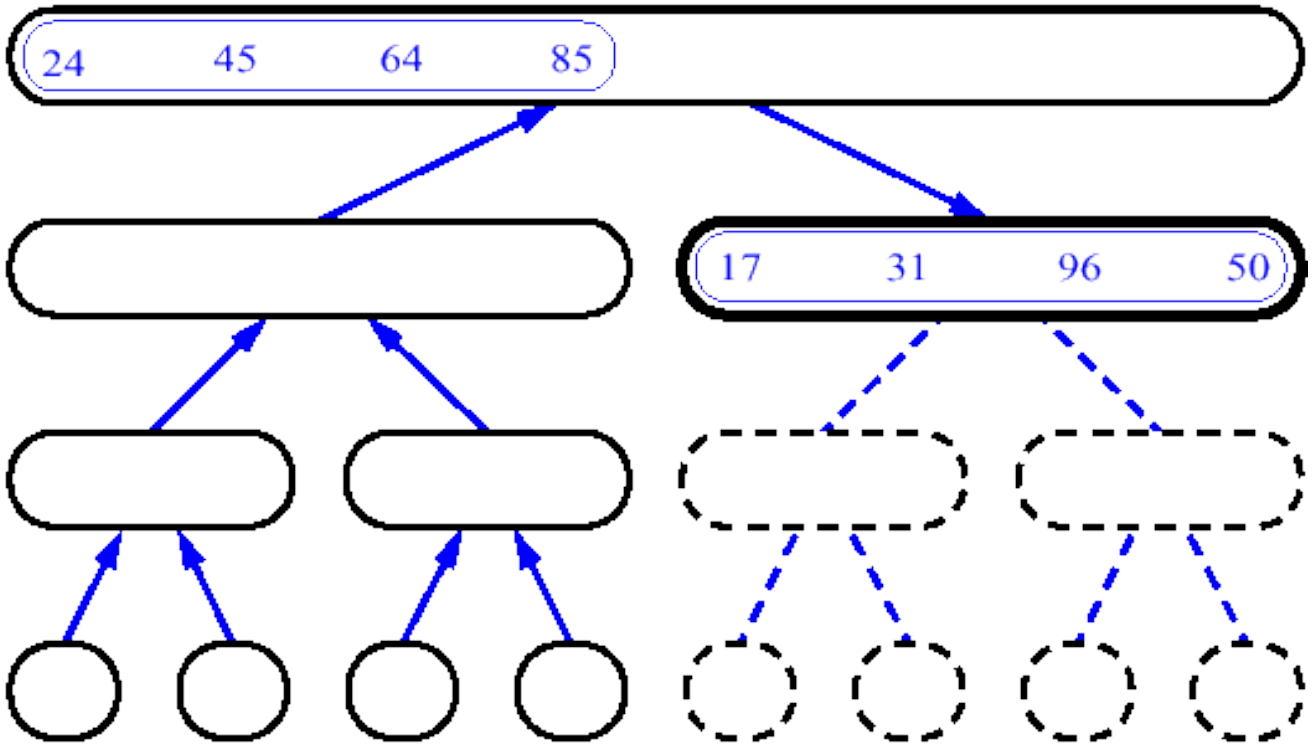
MergeSort



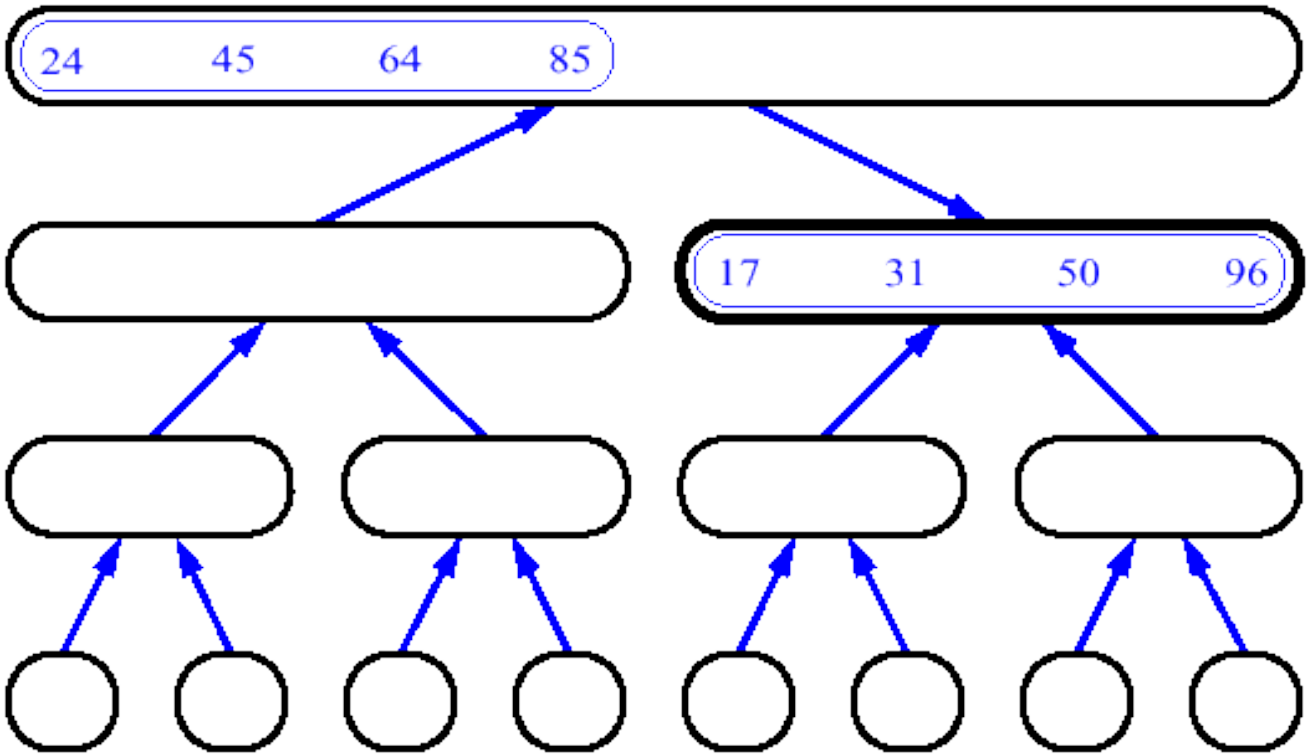
MergeSort



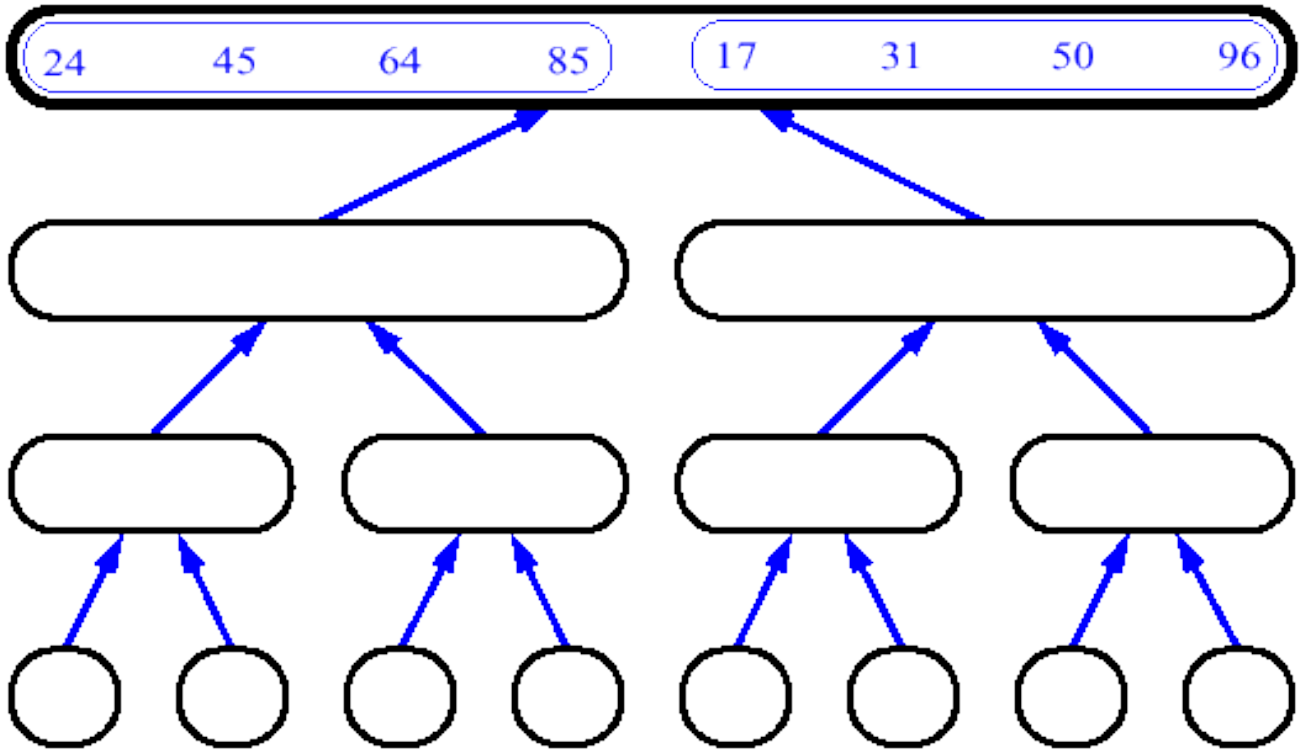
MergeSort



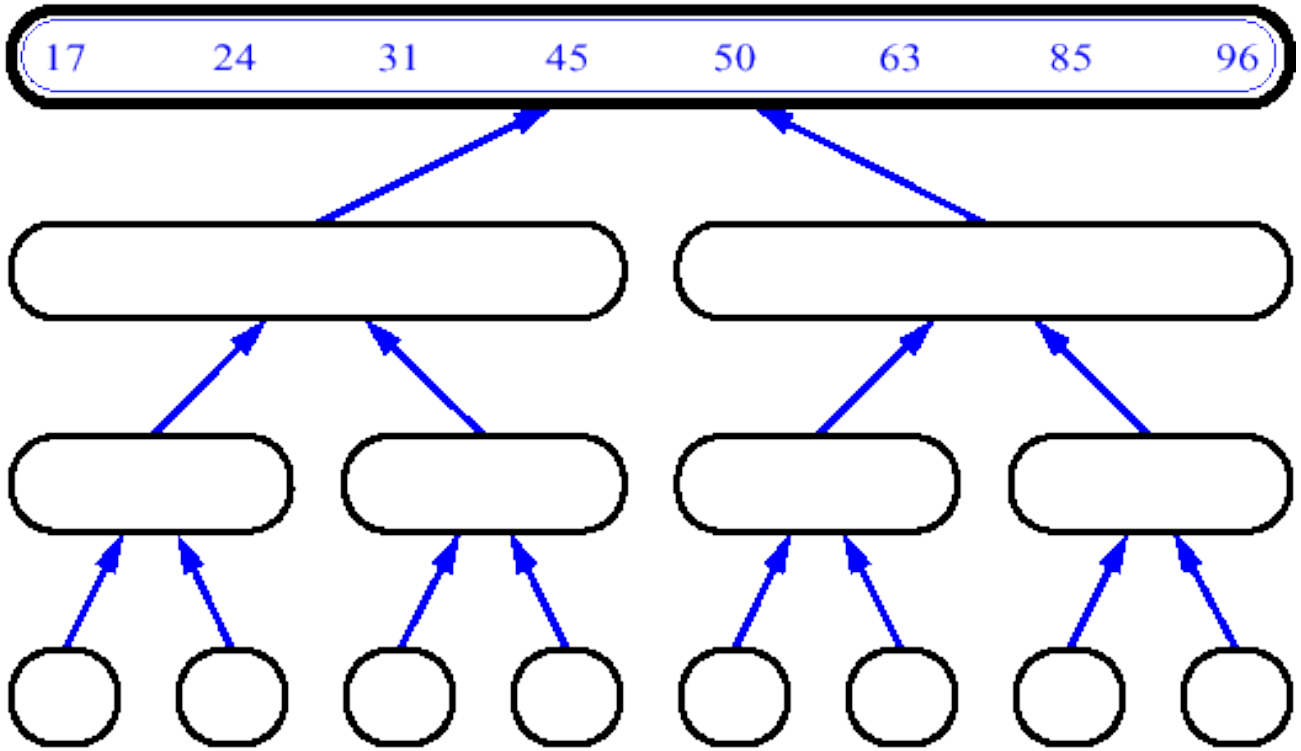
MergeSort



MergeSort



MergeSort

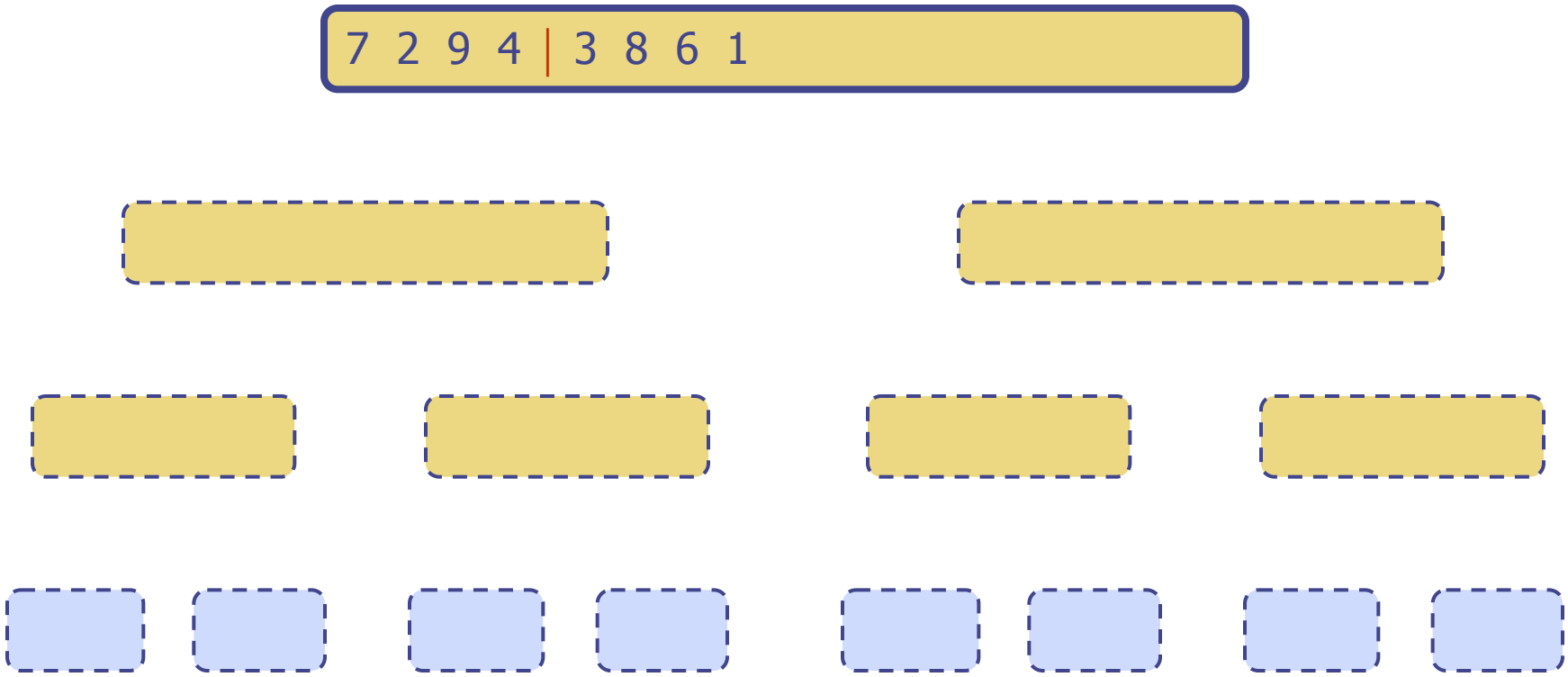


Esempio - Divisione

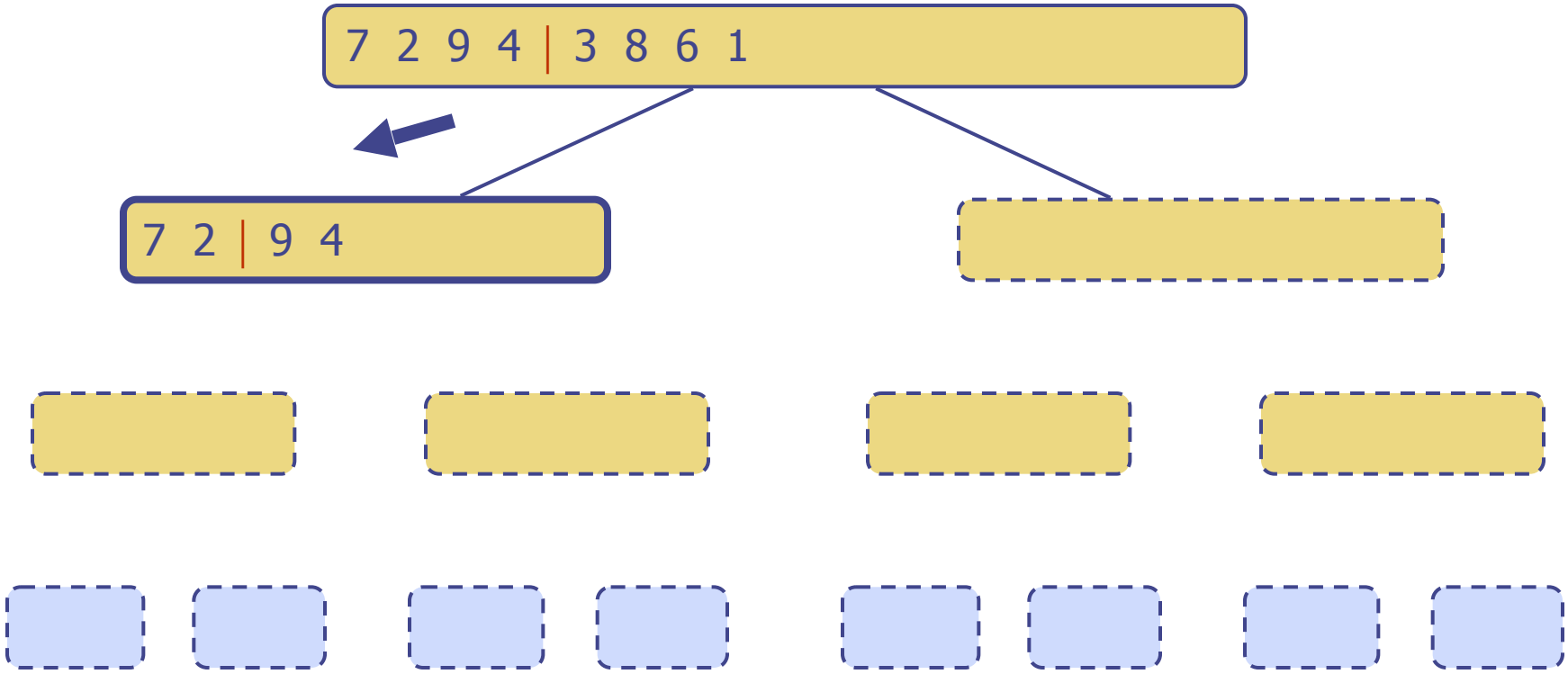
7 2 9 4 3 8 6 1

Provate voi….

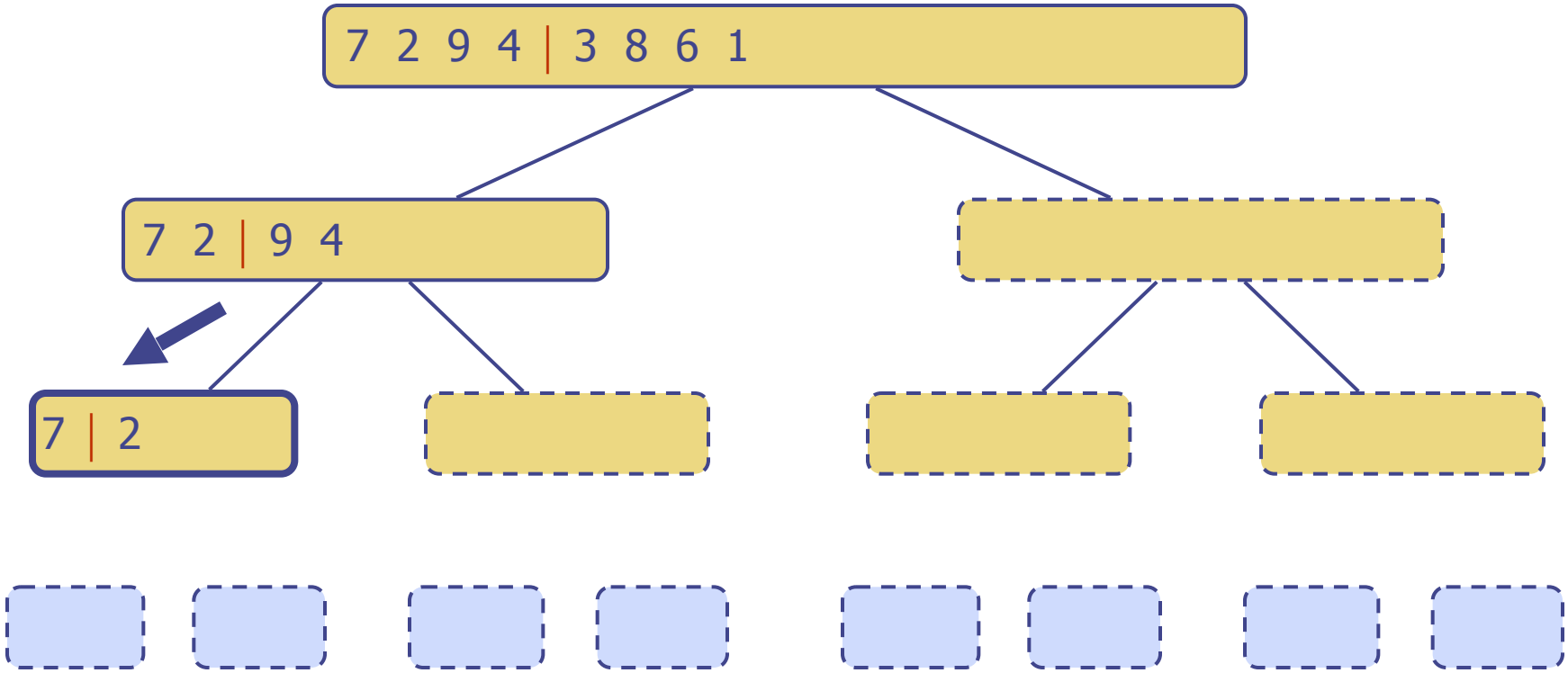
Esempio - Divisione



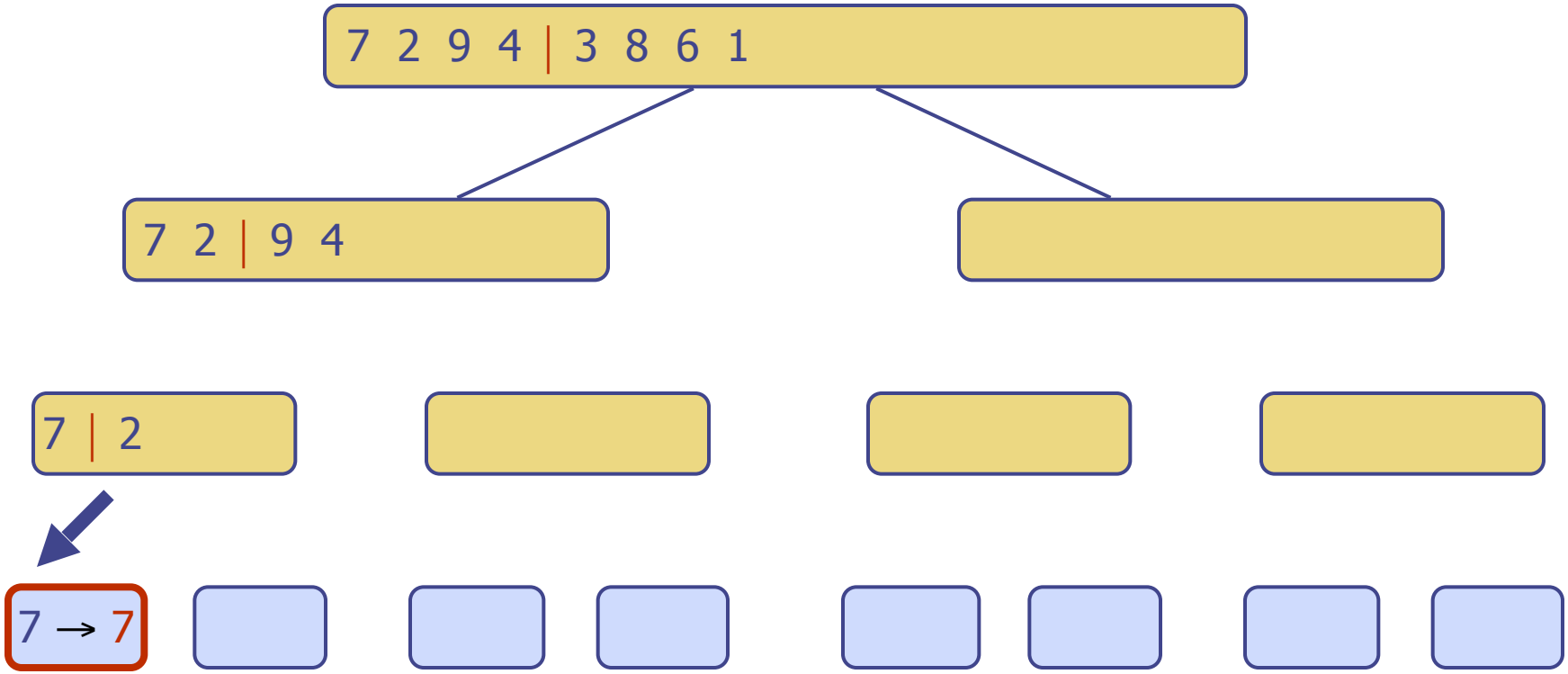
Esempio - Chiamata ric. + Divisione



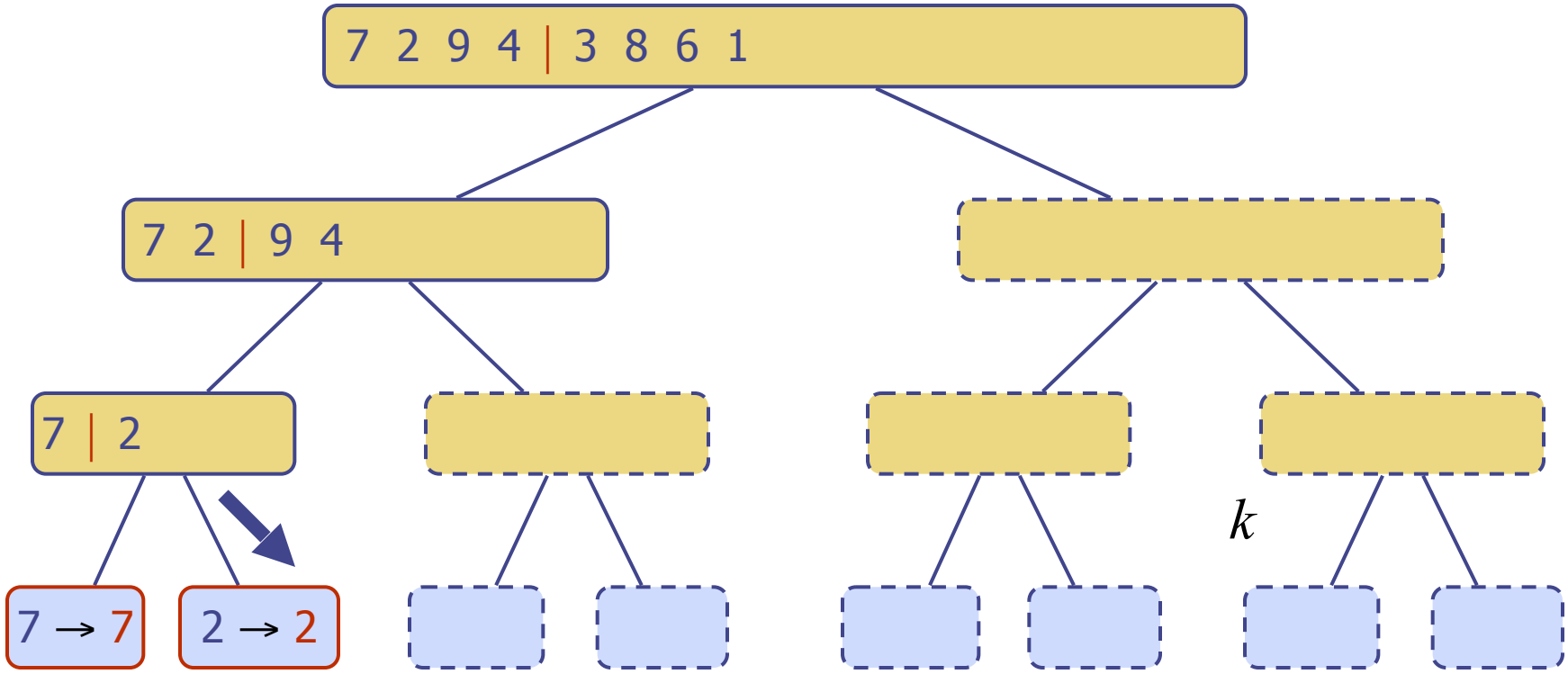
Esempio - Chiamata ric. + Divisione



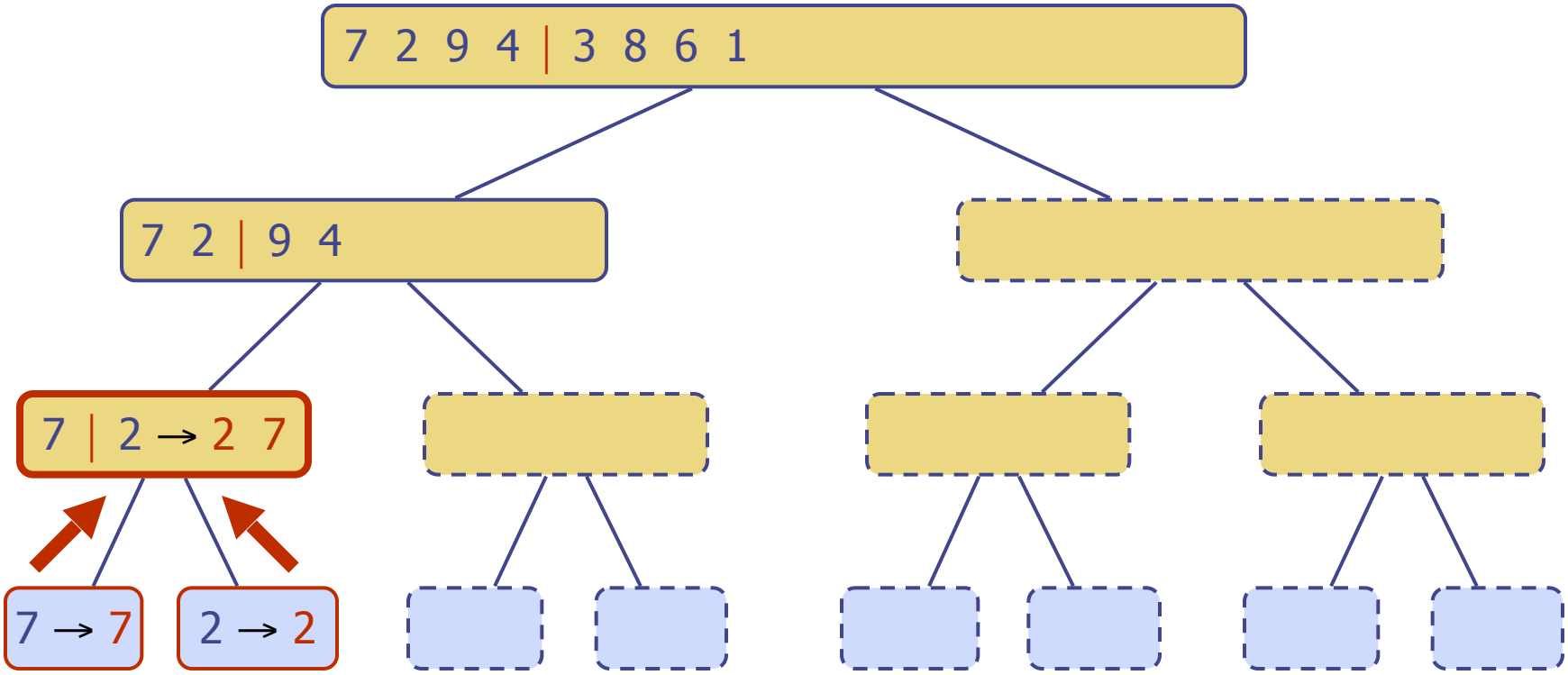
Esempio - Chiamata ric. + Caso Base



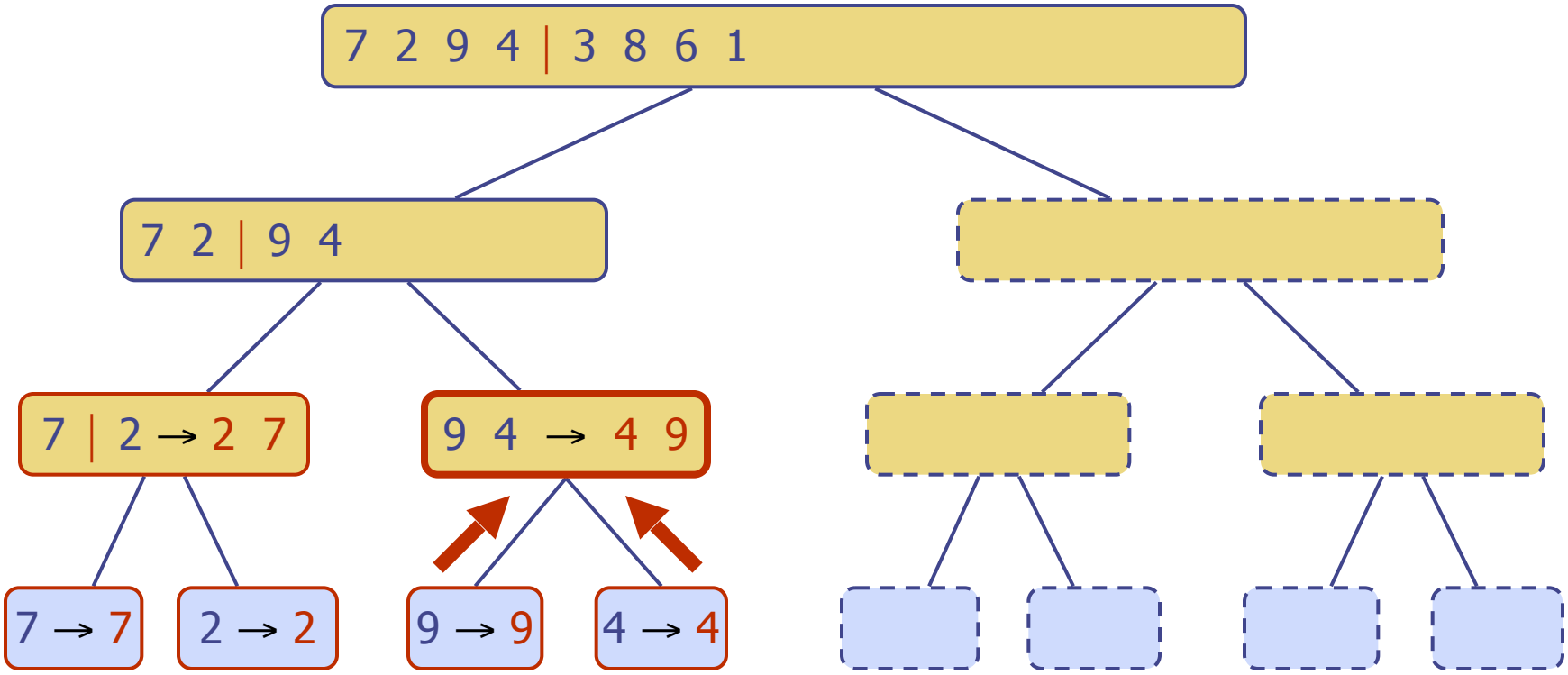
Esempio - Chiamata ric. + Caso Base



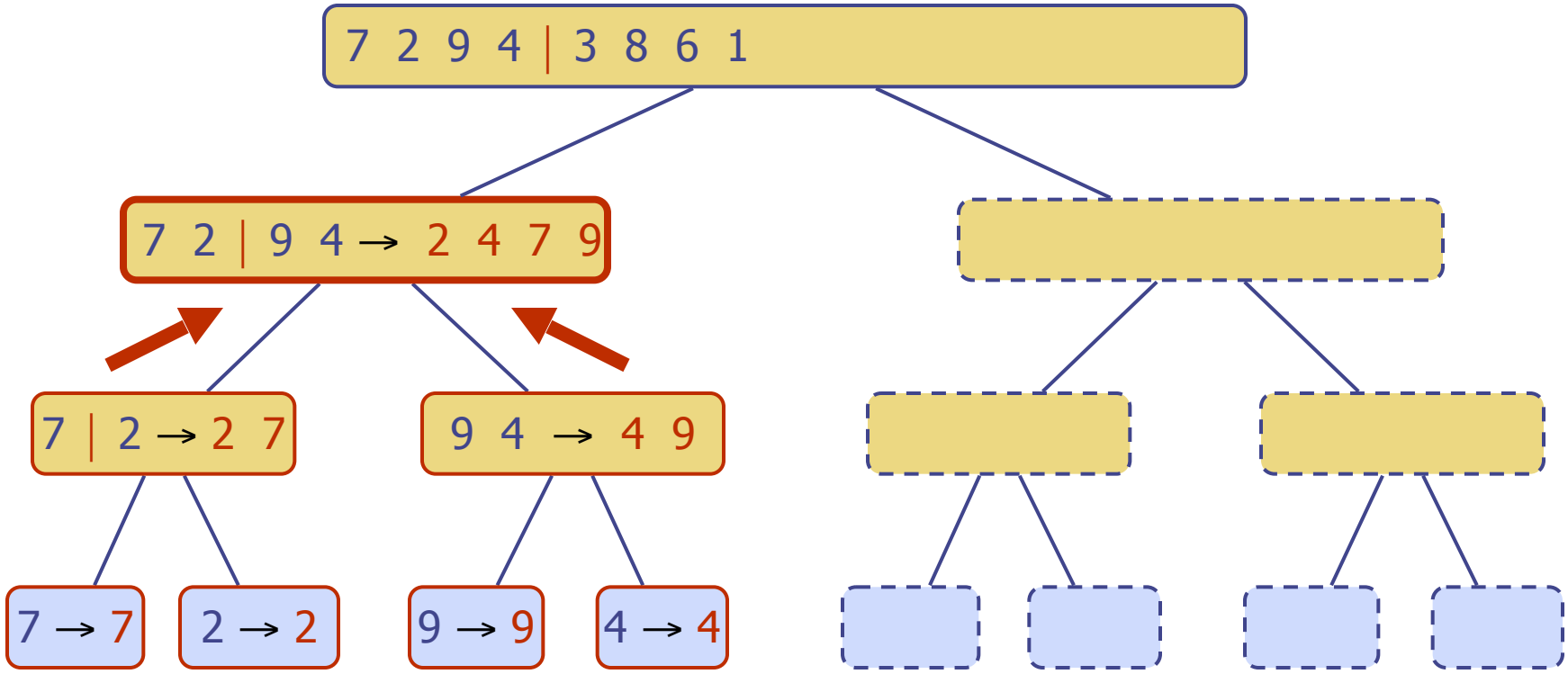
Esempio - Fusione



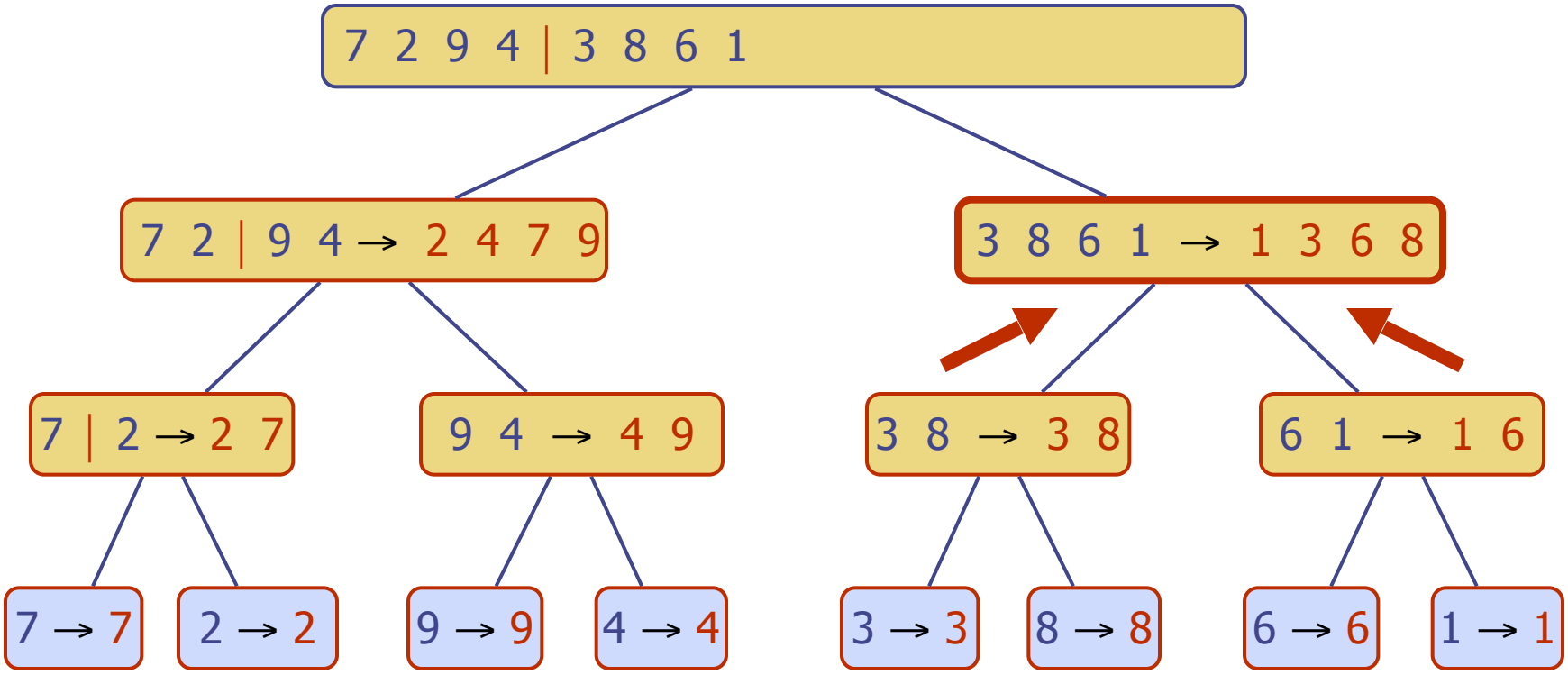
Esempio - Chiamata ric. + Divisione + Caso Base + Fusione



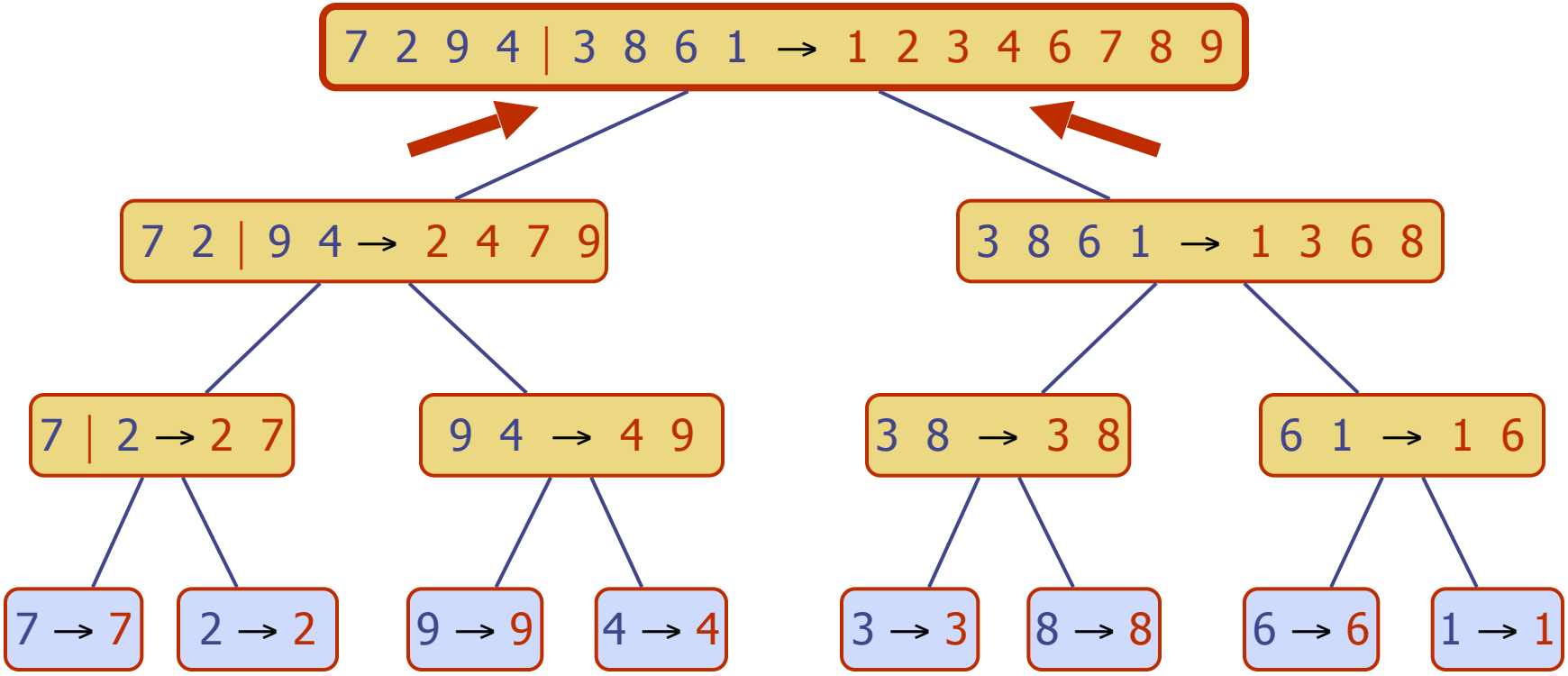
Esempio - Fusione



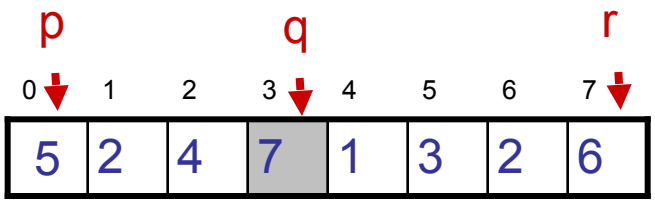
Esempio


7 2 9 4 | 3 8 6 1
7 2 | 9 4 → 2 4 7 9
3 8 6 1 → 1 3 6 8
7 | 2 → 2 7
9 4 → 4 9
3 8 → 3 8
6 1 → 1 6
7 → 7
2 → 2
9 → 9
4 → 4
3 → 3
8 → 8
6 → 6
1 → 1

Esempio - Fusione finale



MergeSort



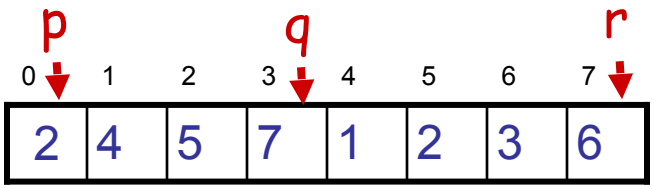
Alg.: MERGE-SORT(*A*, p, r)

```
if p < r                                  ▷  Caso base
  then q = ⌊(p + r)/2⌋                    ▷  Divide
       MERGE-SORT(A, p, q)                ▷        Impera
       MERGE-SORT(A, q + 1, r)            ▷  Impera
       MERGE(A, p, q, r)                  ▷  Combina
```

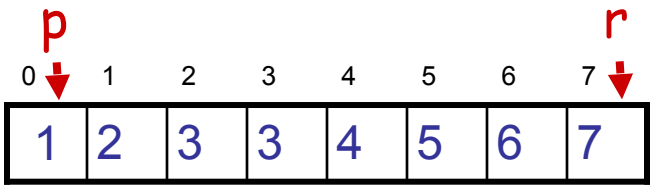
Chiamata Iniziale: MERGE-SORT(A, 0, n)

Merge

- Input: Array A e indici p, q, r tali che $p \leq q < r$
  - Sottoarray $A[p\,..\,q]$ e $A[q+1\,..\,r]$ sono ordinati

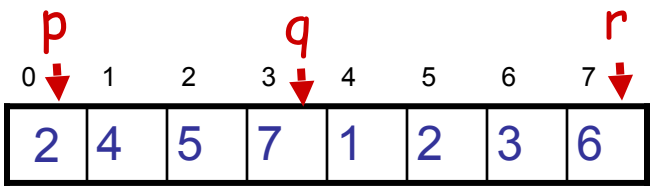


- Output: Un singolo sottoarray ordinato $A[p\,..\,r]$



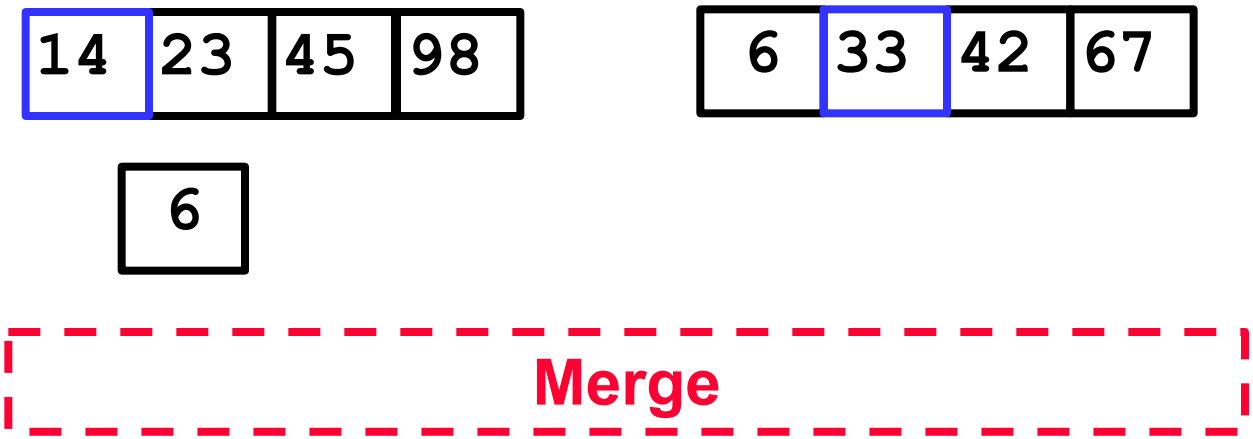
Merge

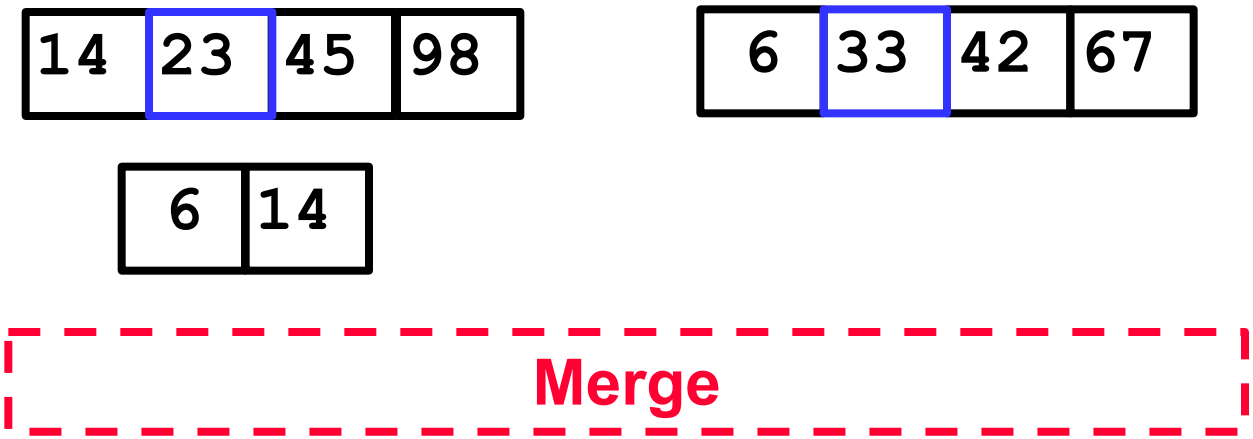
- Idea per il merging:
  - Due pile di carte ordinate
    - Scegli la più piccola dalla cima delle due pile
    - Rimuovila e mettila in output
  - Ripeti il processo fino a che una pila non è vuota
  - Prendi le carte rimanenti dalla pila non vuota (se esistono), e mettile in output

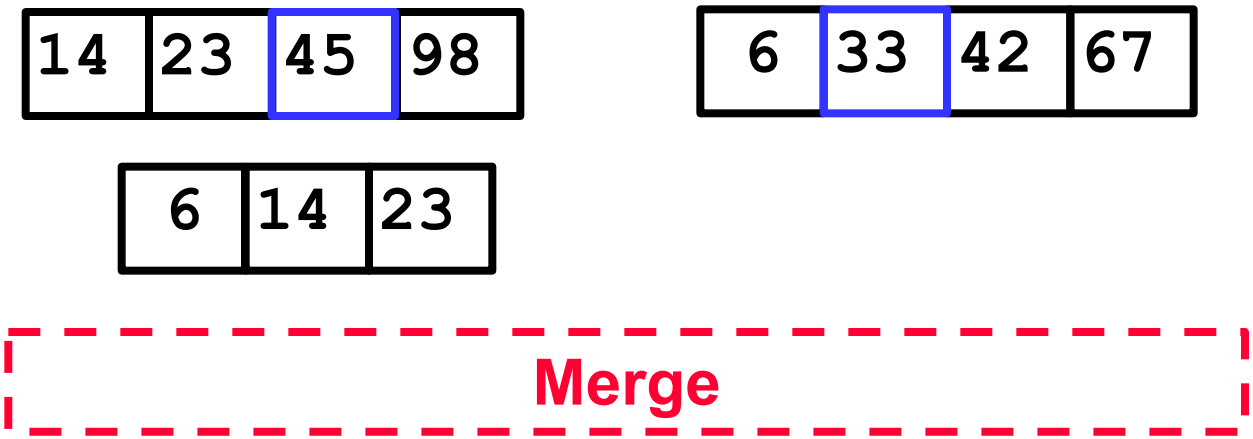


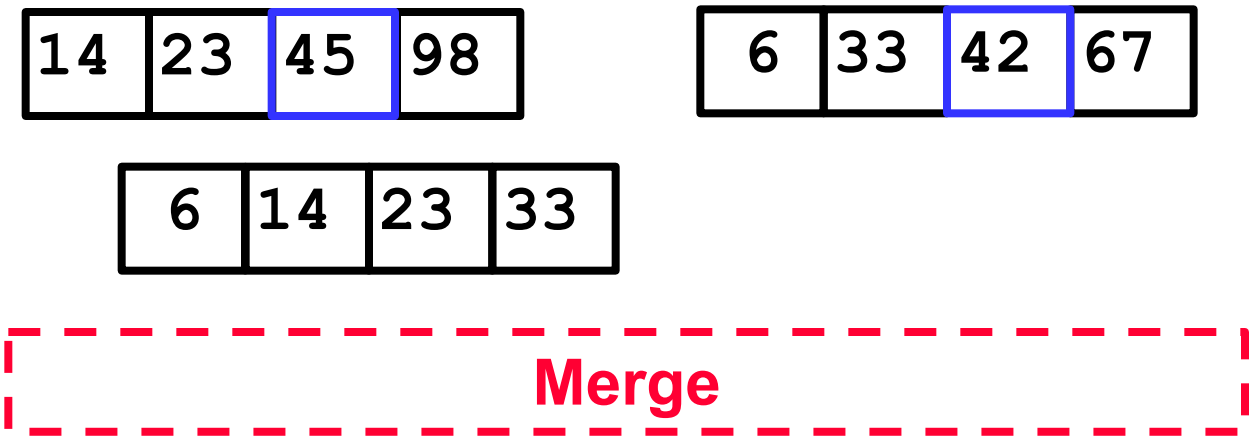

14
23
45
98
6
33
42
67
Merge

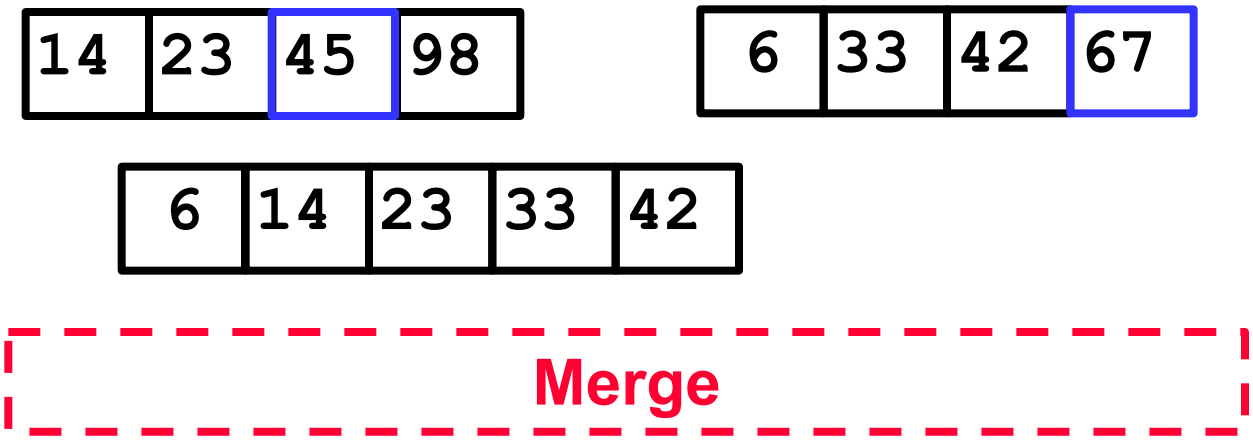
Merge

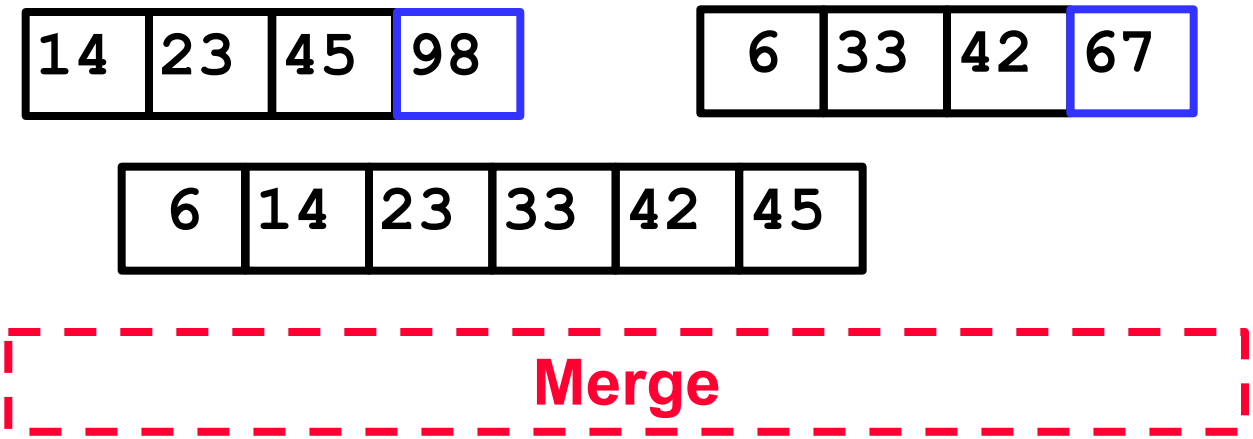

14
23
45
98
6
33
42
67
6
Merge

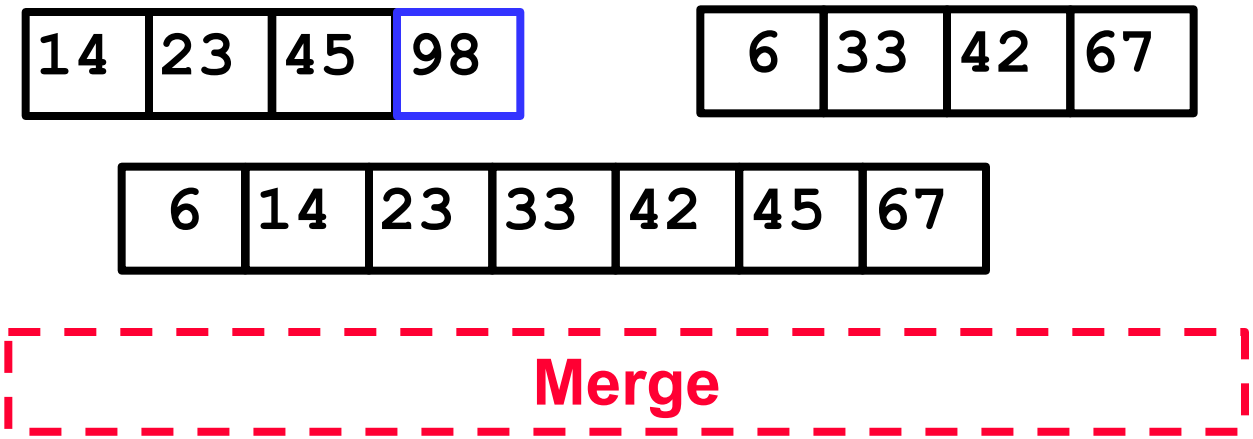

14 23 45 98
6 33 42 67
6 14
Merge

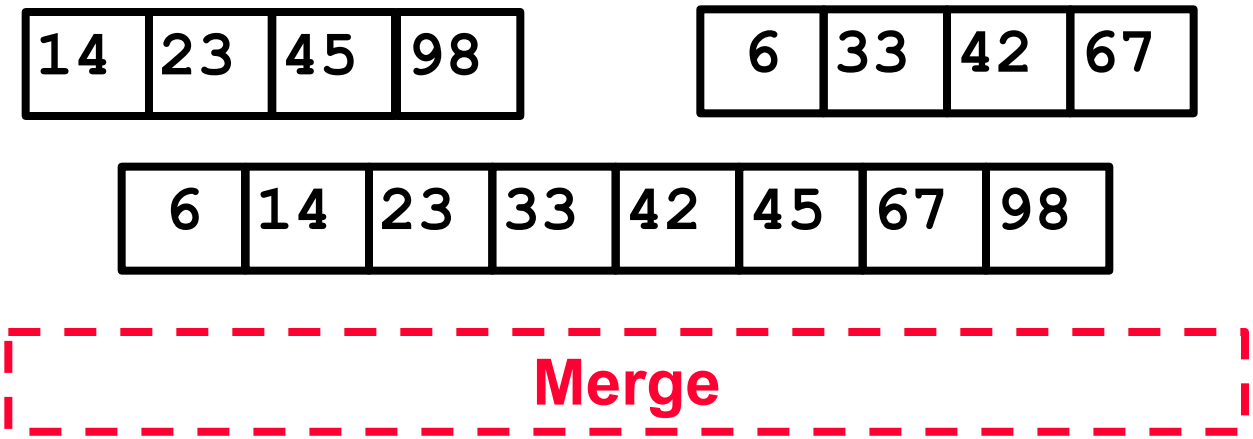

14 23 45 98
6 33 42 67
6 14 23
Merge


14 23 45 98
6 33 42 67
6 14 23 33
Merge

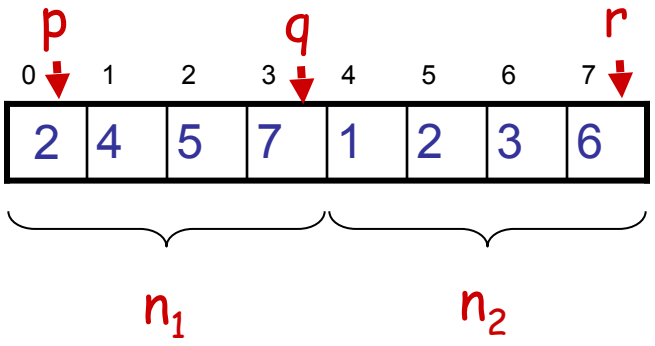

14 23 45 98
6 33 42 67
6 14 23 33 42
Merge


14 23 45 98
6 33 42 67
6 14 23 33 42 45
Merge


14 23 45 98
6 33 42 67
6 14 23 33 42 45 67
Merge

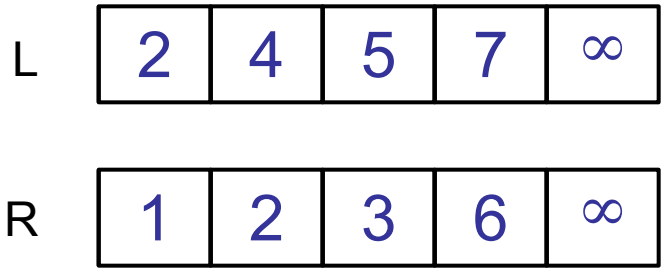

14 23 45 98
6 33 42 67
6 14 23 33 42 45 67 98
Merge

Merge - Pseudocodice

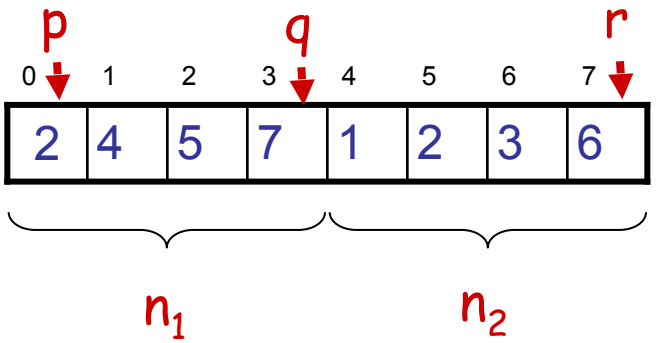


Alg.: MERGE(A, p, q, r)

1. Calcola n_1 e n_2
2. Copia i primi n_1 elementi in
   L[0 . . n_1] e i successivi n_2 in R[0 . . n_2]
3. $L[n_1] = \infty$; $R[n_2] = \infty$
4. i = 0; j = 0;
5. **for** (k = p; k<=r; k++)
6. **if** L[i] ≤ R[j]
7. **then** A[k] = L[i]
8. i =i + 1
9. **else** A[k] = R[j]
10. j = j + 1



Merge - Pseudocodice



Alg.: MERGE(A, p, q, r)

1. Calcola n_1 e n_2
2. Copia i primi n_1 elementi in
   $L[0 \,..\, n_1]$ e i successivi n_2 in $R[0 \,..\, n_2]$
3. $L[n_1] = \infty$; $R[n_2] = \infty$
4. i = 0; j = 0;
5. **for** (k = p; k<=r; k++)
6. **if** L[i] ≤ R[j]
7. **then** A[k] = L[i]
8. i =i + 1
9. **else** A[k] = R[j]
10. j = j + 1

L

2	4	5	7	∞

R

1	2	3	6	∞

A

0	1	2	3	4	5	6	7
2	4	5	7	1	2	3	6

Merge - Costo

Alg.: MERGE(A, p, q, r)

1. Calcola n_1 e n_2
2. Copia i primi n_1 elementi in
   $L[0 \,..\, n_1]$ e i successivi n_2 in $R[0 \,..\, n_2]$
3. $L[n_1] = \infty$; $R[n_2] = \infty$
4. i = 0; j = 0;
5. **for** (k = p; k<=r; k++)
6. **if** $L[\, i \,] \leq R[\, j \,]$
7. **then** $A[k] = L[\, i \,]$
8. i =i + 1
9. **else** $A[k] = R[\, j \,]$
10. j = j + 1

Merge - Costo

Alg.: MERGE(A, p, q, r)

1.	Calcola n_1 e n_2	$\Theta(1)$
2.	Copia i primi n_1 elementi in $L[0 \ldots n_1]$ e i successivi n_2 in $R[0 \ldots n_2]$	$\Theta(n)$
3.	$L[n_1] = \infty$; $R[n_2] = \infty$	$\Theta(1)$
4.	i = 0; j = 0;	$\Theta(1)$
5.	**for** (k = p; k<=r; k++)	
6.	**if** L[i] ≤ R[j]	
7.	**then** A[k] = L[i]	
8.	i =i + 1	
9.	**else** A[k] = R[j]	
10.	j = j + 1	

Merge - Costo

Alg.: MERGE(A, p, q, r)

1. Calcola n_1 e n_2 — $\Theta(1)$
2. Copia i primi n_1 elementi in
   $L[0 \,..\, n_1]$ e i successivi n_2 in $R[0 \,..\, n_2]$ — $\Theta(n)$
3. $L[n_1] = \infty$; $R[n_2] = \infty$ — $\Theta(1)$
4. i = 0; j = 0; — $\Theta(1)$
5. **for** (k = p; k<=r; k++)
6. **if** L[i] ≤ R[j]
7. **then** A[k] = L[i]
8. i =i + 1
9. **else** A[k] = R[j]
10. j = j + 1

Merge - Costo

Alg.: MERGE(A, p, q, r)

1. Calcola n_1 e n_2 — $\Theta(1)$
2. Copia i primi n_1 elementi in L[0 . . n_1] e i successivi n_2 in R[0 . . n_2] — $\Theta(n)$
3. $L[n_1] = \infty$; $R[n_2] = \infty$ — $\Theta(1)$
4. i = 0; j = 0; — $\Theta(1)$
5. **for** (k = p; k<=r; k++)
6. **if** L[i] ≤ R[j]
7. **then** A[k] = L[i]
8. i =i + 1
9. **else** A[k] = R[j]
10. j = j + 1

$\Theta(n_1 + n_2) = \Theta(n)$

Merge - Costo

$\Theta(n)$

Alg.: MERGE(A, p, q, r)

1. Calcola n_1 e n_2 — $\Theta(1)$
2. Copia i primi n_1 elementi in
   L[0 . . n_1] e i successivi n_2 in R[0 . . n_2] — $\Theta(n)$
3. $L[n_1] = \infty$; $R[n_2] = \infty$ — $\Theta(1)$
4. i = 0; j = 0; — $\Theta(1)$
5. **for** (k = p; k<=r; k++)
6. **if** L[i] ≤ R[j]
7. **then** A[k] = L[i]
8. i =i + 1
9. **else** A[k] = R[j]
10. j = j + 1

$\Theta(n_1 + n_2) = \Theta(n)$

Merge - Correttezza

Alg.: MERGE(A, p, q, r)

1. Calcola n_1 e n_2
2. Copia i primi n_1 elementi in
   $L[0 \,..\, n_1]$ e i successivi n_2 in $R[0 \,..\, n_2]$
3. $L[n_1] = \infty$; $R[n_2] = \infty$
4. i = 0; j = 0;
5. **for** (k = p; k<=r; k++)
6. **if** L[i] ≤ R[j]
7. **then** *A*[k] = L[i]
8. i =i + 1
9. **else** *A*[k] = R[j]
10. j = j + 1

Invariante: all'inizio di ogni **for**, A[p..k-1] contiene i k-p elementi più piccoli di L e R, in ordine. L[i] e R[j] sono i più piccoli non ancora copiati in A

Da dimostrare:

1. **Inizializzazione**: invariante vera all'inizio del primo ciclo

2. **Mantenimento**: se vera prima di un ciclo, allora vera prima del prossimo

3. **Terminazione**: vera alla fine del ciclo

Merge - Correttezza

Alg.: MERGE(A, p, q, r)

1. Calcola n_1 e n_2
2. Copia i primi n_1 elementi in
   $L[0 \,..\, n_1]$ e i successivi n_2 in $R[0 \,..\, n_2]$
3. $L[n_1] = \infty$; $R[n_2] = \infty$
4. i = 0; j = 0;
5. **for** (k = p; k<=r; k++)
6. **if** L[i] ≤ R[j]
7. **then** A[k] = L[i]
8. i =i + 1
9. **else** A[k] = R[j]
10. j = j + 1

Invariante: all'inizio di ogni **for**, A[p..k-1] contiene i k-p elementi più piccoli di L e R, in ordine. L[i] e R[j] sono i più piccoli non ancora copiati in A

1. **Inizializzazione**: invariante vera all'inizio del primo ciclo

????

Merge - Correttezza

Alg.: MERGE(A, p, q, r)

1. Calcola n_1 e n_2
2. Copia i primi n_1 elementi in
   $L[0 \,..\, n_1]$ e i successivi n_2 in $R[0 \,..\, n_2]$
3. $L[n_1] = \infty$; $R[n_2] = \infty$
4. i = 0; j = 0;
5. **for** (k = p; k<=r; k++)
6. **if** L[i] ≤ R[j]
7. **then** A[k] = L[i]
8. i =i + 1
9. **else** A[k] = R[j]
10. j = j + 1

Invariante: all'inizio di ogni **for**, A[p..k-1] contiene i k-p elementi più piccoli di L e R, in ordine. L[i] e R[j] sono i più piccoli non ancora copiati in A

1. **Inizializzazione**: invariante vera all'inizio del primo ciclo

All'inizio, k=p, e quindi A[p..k-1] contiene i k-p=0 elementi più piccoli di L e R, in ordine

Merge - Correttezza

Alg.: MERGE(A, p, q, r)

1. Calcola n_1 e n_2
2. Copia i primi n_1 elementi in
   $L[0 \ldots n_1]$ e i successivi n_2 in $R[0 \ldots n_2]$
3. $L[n_1] = \infty$; $R[n_2] = \infty$
4. i = 0; j = 0;
5. **for** (k = p; k<=r; k++)
6. **if** L[i] ≤ R[j]
7. **then** A[k] = L[i]
8. i =i + 1
9. **else** A[k] = R[j]
10. j = j + 1

Invariante: all'inizio di ogni **for**, A[p..k-1] contiene i k-p elementi più piccoli di L e R, in ordine. L[i] e R[j] sono i più piccoli non ancora copiati in A

1. **Inizializzazione**: invariante vera all'inizio del primo ciclo

All'inizio, k=p, e quindi A[p..k-1] contiene i k-p=0 elementi più piccoli di L e R, in ordine

Inoltre, essendo i=j=0, L[j] e R[j] sono i più piccoli elementi di L e R, rispettivamente, non ancora copiati in A

Merge - Correttezza

Alg.: MERGE(A, p, q, r)

1. Calcola n_1 e n_2
2. Copia i primi n_1 elementi in
   $L[0 \ldots n_1]$ e i successivi n_2 in $R[0 \ldots n_2]$
3. $L[n_1] = \infty$; $R[n_2] = \infty$
4. i = 0; j = 0;
5. **for** (k = p; k<=r; k++)
6. **if** L[i] ≤ R[j]
7. **then** *A*[k] = L[i]
8. i =i + 1
9. **else** *A*[k] = R[j]
10. j = j + 1

Invariante: all'inizio di ogni **for**, A[p..k-1] contiene i k-p elementi più piccoli di L e R, in ordine. L[i] e R[j] sono i più piccoli non ancora copiati in A

2. **Mantenimento**: se vera prima di un ciclo, allora vera prima del prossimo

????

Merge - Correttezza

Alg.: MERGE(A, p, q, r)

1. Calcola n_1 e n_2
2. Copia i primi n_1 elementi in
   $L[0 \ldots n_1]$ e i successivi n_2 in $R[0 \ldots n_2]$
3. $L[n_1] = \infty$; $R[n_2] = \infty$
4. i = 0; j = 0;
5. **for** (k = p; k<=r; k++)
6. **if** L[i] ≤ R[j]
7. **then** *A*[k] = L[i]
8. i =i + 1
9. **else** *A*[k] = R[j]
10. j = j + 1

Invariante: all'inizio di ogni **for**, A[p..k-1] contiene i k-p elementi più piccoli di L e R, in ordine. L[i] e R[j] sono i più piccoli non ancora copiati in A

2. **Mantenimento**: se vera prima di un ciclo, allora vera prima del prossimo

Supponiamo che **L[i]≤R[j]**. Quindi, L[i] è il più piccolo elemento non ancora copiato in A

Merge - Correttezza

Alg.: MERGE(A, p, q, r)

1. Calcola n_1 e n_2
2. Copia i primi n_1 elementi in
   $L[0 \,..\, n_1]$ e i successivi n_2 in $R[0 \,..\, n_2]$
3. $L[n_1] = \infty$; $R[n_2] = \infty$
4. i = 0; j = 0;
5. **for** (k = p; k<=r; k++)
6. **if** L[i] ≤ R[j]
7. **then** A[k] = L[i]
8. i =i + 1
9. **else** A[k] = R[j]
10. j = j + 1

Invariante: all'inizio di ogni **for**, A[p..k-1] contiene i k-p elementi più piccoli di L e R, in ordine. L[i] e R[j] sono i più piccoli non ancora copiati in A

2. **Mantenimento**: se vera prima di un ciclo, allora vera prima del prossimo

Supponiamo che **L[i]≤R[j]**. Quindi, L[i] è il più piccolo elemento non ancora copiato in A

Dato che A[p..k-1] contiene i k-p elementi più piccoli, dopo aver copiato L[i] in A[k] (Linea 7), A[p..k] contiene i k-p+1 elementi più piccoli

Merge - Correttezza

Alg.: MERGE(A, p, q, r)

1. Calcola n_1 e n_2
2. Copia i primi n_1 elementi in
   $L[0 \ldots n_1]$ e i successivi n_2 in $R[0 \ldots n_2]$
3. $L[n_1] = \infty$; $R[n_2] = \infty$
4. i = 0; j = 0;
5. **for** (k = p; k<=r; k++)
6. **if** L[i] ≤ R[j]
7. **then** A[k] = L[i]
8. i =i + 1
9. **else** A[k] = R[j]
10. j = j + 1

Invariante: all'inizio di ogni **for**, A[p..k-1] contiene i k-p elementi più piccoli di L e R, in ordine. L[i] e R[j] sono i più piccoli non ancora copiati in A

2. **Mantenimento**: se vera prima di un ciclo, allora vera prima del prossimo

Supponiamo che **L[i]≤R[j]**. Quindi, L[i] è il più piccolo elemento non ancora copiato in A

Dato che A[p..k-1] contiene i k-p elementi più piccoli, dopo aver copiato L[i] in A[k] (Linea 7), A[p..k] contiene i k-p+1 elementi più piccoli

L'incremento di k (nel **for**) e di i (Linea 8) conferma l'invariante del ciclo per la successiva iterazione. Analogo ragionamento se invece L[i] > R[j]

Merge - Correttezza

Alg.: MERGE(A, p, q, r)

1. Calcola n_1 e n_2
2. Copia i primi n_1 elementi in
   $L[0 \,..\, n_1]$ e i successivi n_2 in $R[0 \,..\, n_2]$
3. $L[n_1] = \infty$; $R[n_2] = \infty$
4. i = 0; j = 0;
5. **for** (k = p; k<=r; k++)
6. **if** L[i] ≤ R[j]
7. **then** A[k] = L[i]
8. i =i + 1
9. **else** A[k] = R[j]
10. j = j + 1

Invariante: all'inizio di ogni **for**, A[p..k-1] contiene i k-p elementi più piccoli di L e R, in ordine. L[i] e R[j] sono i più piccoli non ancora copiati in A

3. **Terminazione**: vera alla fine del ciclo

????

Merge - Correttezza

Alg.: MERGE(A, p, q, r)

1. Calcola n_1 e n_2
2. Copia i primi n_1 elementi in
   $L[0 \,..\, n_1]$ e i successivi n_2 in $R[0 \,..\, n_2]$
3. $L[n_1] = \infty$; $R[n_2] = \infty$
4. i = 0; j = 0;
5. **for** (k = p; k<=r; k++)
6. **if** L[i] ≤ R[j]
7. **then** *A*[k] = L[i]
8. i =i + 1
9. **else** *A*[k] = R[j]
10. j = j + 1

Invariante: all'inizio di ogni **for**, A[p..k-1] contiene i k-p elementi più piccoli di L e R, in ordine. L[i] e R[j] sono i più piccoli non ancora copiati in A

3. **Terminazione**: vera alla fine del ciclo

Alla fine, k=r+1. Per l'invariante, A[p..k-1] contiene i k-p=r-p+1 elementi più piccoli di L e R, in ordine

Merge - Correttezza

Alg.: MERGE(A, p, q, r)

1. Calcola n_1 e n_2
2. Copia i primi n_1 elementi in
   $L[0 \,..\, n_1]$ e i successivi n_2 in $R[0 \,..\, n_2]$
3. $L[n_1] = \infty$; $R[n_2] = \infty$
4. i = 0; j = 0;
5. **for** (k = p; k<=r; k++)
6. **if** L[i] ≤ R[j]
7. **then** A[k] = L[i]
8. i =i + 1
9. **else** A[k] = R[j]
10. j = j + 1

Invariante: all'inizio di ogni **for**, A[p..k-1] contiene i k-p elementi più piccoli di L e R, in ordine. L[i] e R[j] sono i più piccoli non ancora copiati in A

3. **Terminazione**: vera alla fine del ciclo

Alla fine, k=r+1. Per l'invariante, A[p..k-1] contiene i k-p=r-p+1 elementi più piccoli di L e R, in ordine

L e R contengono **quanti** elementi?

Merge - Correttezza

Alg.: MERGE(A, p, q, r)

1. Calcola n_1 e n_2
2. Copia i primi n_1 elementi in
   $L[0 \,..\, n_1]$ e i successivi n_2 in $R[0 \,..\, n_2]$
3. $L[n_1] = \infty$; $R[n_2] = \infty$
4. i = 0; j = 0;
5. **for** (k = p; k<=r; k++)
6. **if** L[i] ≤ R[j]
7. **then** A[k] = L[i]
8. i =i + 1
9. **else** A[k] = R[j]
10. j = j + 1

Invariante: all'inizio di ogni **for**, A[p..k-1] contiene i k-p elementi più piccoli di L e R, in ordine. L[i] e R[j] sono i più piccoli non ancora copiati in A

3. **Terminazione**: vera alla fine del ciclo

Alla fine, k=r+1. Per l'invariante, A[p..k-1] contiene i k-p=r-p+1 elementi più piccoli di L e R, in ordine

L e R contengono n1 + n2 + 2=r-p+3 elementi

Merge - Correttezza

Alg.: MERGE(A, p, q, r)

1. Calcola n_1 e n_2
2. Copia i primi n_1 elementi in
   $L[0 \,..\, n_1]$ e i successivi n_2 in $R[0 \,..\, n_2]$
3. $L[n_1] = \infty$; $R[n_2] = \infty$
4. i = 0; j = 0;
5. **for** (k = p; k<=r; k++)
6. **if** L[i] ≤ R[j]
7. **then** A[k] = L[i]
8. i =i + 1
9. **else** A[k] = R[j]
10. j = j + 1

Invariante: all'inizio di ogni **for**, A[p..k-1] contiene i k-p elementi più piccoli di L e R, in ordine. L[i] e R[j] sono i più piccoli non ancora copiati in A

3. **Terminazione**: vera alla fine del ciclo

Alla fine, k=r+1. Per l'invariante, A[p..k-1] contiene i k-p=r-p+1 elementi più piccoli di L e R, in ordine

L e R contengono n1 + n2 + 2=r-p+3 elementi

Tutti gli elementi di L e R sono stati copiati, tranne i più grandi

Merge - Correttezza

ariante: all'inizio di ogni **for**,
k-1] contiene i k-p elementi più
li di L e R, in ordine. L[i] e R[j]
i più piccoli non ancora copiati in A

inazione: vera alla fine del ciclo

Alla fine, k=r+1. Per l'invariante, A[p..k-1] contiene i k-p=r-p+1 elementi più piccoli di L e R, in ordine

L e R contengono n1 + n2 + 2=r-p+3 elementi

Tutti gli elementi di L e R sono stati copiati, tranne i più grandi

MergeSort - Costo

Alg.: MERGE-SORT(A, p, r)

if p < r

then $q = \lfloor (p + r)/2 \rfloor$

MERGE-SORT(A, p, q)

MERGE-SORT(A, q + 1, r)

MERGE(A, p, q, r)

Come impostiamo il calcolo?

MergeSort - Costo

Alg.: MERGE-SORT(A, p, r)

if p < r

then $q = \lfloor (p + r)/2 \rfloor$

MERGE-SORT(A, p, q)

MERGE-SORT(A, q + 1, r)

MERGE(A, p, q, r)

Esempio di Divide-et-Impera….

MergeSort - Costo

Esempio di Divide-et-Impera….

Alg.: MERGE-SORT(*A*, p, r)

if p < r

then q = ⌊(p + r)/2⌋

MERGE-SORT(*A*, p, q)

MERGE-SORT(*A*, q + 1, r)

MERGE(*A*, p, q, r)

$$T(n) = \begin{cases} \Theta(1) & \text{se } n \leq c \\ aT(\frac{n}{b}) + D(n) + C(n) & \text{se } n > c \end{cases}$$

MergeSort - Costo

Alg.: MERGE-SORT(*A*, p, r)

```
if p < r
   then q = ⌊(p + r)/2⌋
        MERGE-SORT(A, p, q)
        MERGE-SORT(A, q + 1, r)
        MERGE(A, p, q, r)
```

Esempio di Divide-et-Impera….

$$T(n) = \begin{cases} \Theta(1) & \text{se } n \leq c \\ aT(\frac{n}{b}) + D(n) + C(n) & \text{se } n > c \end{cases}$$

D(n) = ????

MergeSort - Costo

Alg.: MERGE-SORT(*A*, p, r)

if p < r

  then q = ⌊(p + r)/2⌋

    MERGE-SORT(*A*, p, q)

    MERGE-SORT(*A*, q + 1, r)

    MERGE(*A*, p, q, r)

Esempio di Divide-et-Impera….

$$T(n) = \begin{cases} \Theta(1) & \text{se } n \leq c \\ aT(\frac{n}{b}) + D(n) + C(n) & \text{se } n > c \end{cases}$$

D(n) = Θ(1)

C(n) = Θ(????)

MergeSort - Costo

Esempio di Divide-et-Impera….

Alg.: MERGE-SORT(*A*, p, r)

if p < r

  then q = ⌊(p + r)/2⌋

    MERGE-SORT(*A*, p, q)

    MERGE-SORT(*A*, q + 1, r)

    MERGE(*A*, p, q, r)

$$T(n) = \begin{cases} \Theta(1) & \text{se } n \leq c \\ aT(\frac{n}{b}) + D(n) + C(n) & \text{se } n > c \end{cases}$$

D(n) = Θ(1)

C(n) = Θ(n)

MergeSort - Costo

Alg.: MERGE-SORT(*A*, p, r)

```
if p < r
   then q = ⌊(p + r)/2⌋
        MERGE-SORT(A, p, q)
        MERGE-SORT(A, q + 1, r)
        MERGE(A, p, q, r)
```

Esempio di Divide-et-Impera….

$$T(n) = \begin{cases} \Theta(1) & \text{se } n \leq c \\ aT(\frac{n}{b}) + D(n) + C(n) & \text{se } n > c \end{cases}$$

D(n) = Θ(1)

C(n) = Θ(n)

Impera(n) → cosa sono *a* e *b*?

MergeSort - Costo

Alg.: MERGE-SORT(*A*, p, r)

if p < r

  then q = ⌊(p + r)/2⌋

    MERGE-SORT(*A*, p, q)

    MERGE-SORT(*A*, q + 1, r)

    MERGE(*A*, p, q, r)

Esempio di Divide-et-Impera….

$$T(n) = \begin{cases} \Theta(1) & \text{se } n \leq c \\ aT(\frac{n}{b}) + D(n) + C(n) & \text{se } n > c \end{cases}$$

D(n) = Θ(1)

C(n) = Θ(n)

Impera(n) → *a* = b = 2

MergeSort - Costo

Esempio di Divide-et-Impera….

Alg.: MERGE-SORT(*A*, p, r)

```
if p < r
   then q = ⌊(p + r)/2⌋
        MERGE-SORT(A, p, q)
        MERGE-SORT(A, q + 1, r)
        MERGE(A, p, q, r)
```

$$T(n) = \begin{cases} \Theta(1) & \text{se } n \leq c \\ aT(\frac{n}{b}) + D(n) + C(n) & \text{se } n > c \end{cases}$$

D(n) = Θ(1)

C(n) = Θ(n)

Impera(n) → *a* = b = 2

$$T(n) = \begin{cases} ???? & \text{se } n \leq ? \\ ???? & \text{se } n > ? \end{cases}$$

MergeSort - Costo

Esempio di Divide-et-Impera….

Alg.: MERGE-SORT(*A*, p, r)

```
if p < r
   then q = ⌊(p + r)/2⌋
        MERGE-SORT(A, p, q)
        MERGE-SORT(A, q + 1, r)
        MERGE(A, p, q, r)
```

$$T(n) = \begin{cases} \Theta(1) & \text{se } n \leq c \\ aT(\frac{n}{b}) + D(n) + C(n) & \text{se } n > c \end{cases}$$

D(n) = Θ(1)

C(n) = Θ(n)

Impera(n) → *a* = b = 2

$$T(n) = \begin{cases} \Theta(1) & \text{se } n = 1 \\ ???? & \text{se } n > ? \end{cases}$$

MergeSort - Costo

Esempio di Divide-et-Impera….

Alg.: MERGE-SORT(*A*, p, r)

if p < r

  then q = ⌊(p + r)/2⌋

    MERGE-SORT(*A*, p, q)

    MERGE-SORT(*A*, q + 1, r)

    MERGE(*A*, p, q, r)

$$T(n) = \begin{cases} \Theta(1) & \text{se } n \leq c \\ aT(\frac{n}{b}) + D(n) + C(n) & \text{se } n > c \end{cases}$$

D(n) = Θ(1)

C(n) = Θ(n)

Impera(n) → *a* = b = 2

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

MergeSort - Costo

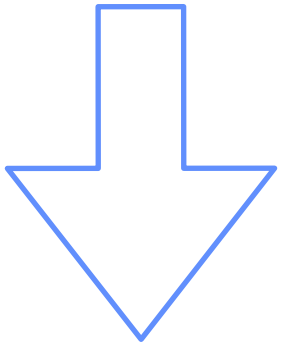
$$T(n) = \begin{cases} \Theta(1) & \text{se } n = 1 \\ 2T(\frac{n}{2}) + \Theta(n) & \text{se } n > 1 \end{cases}$$

Ci sono vari modi per risolvere questa equazione:

- Svolgendola (iterativo)
- Master Theorem
- Albero

MergeSort - Costo con Metodo Iterativo

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



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

MergeSort - Costo con Metodo Iterativo

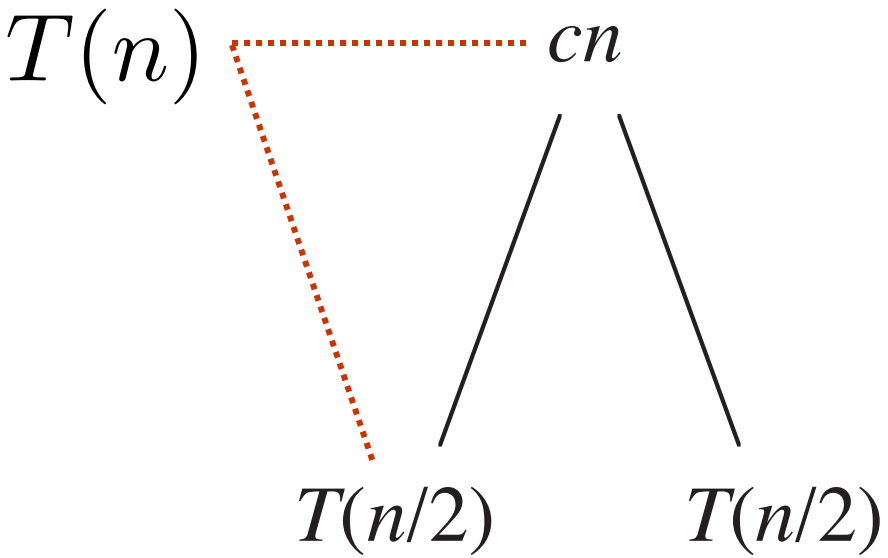
$$T(n) = \begin{cases} c & \text{se } n = 1 \\ 2T(\frac{n}{2}) + cn & \text{se } n > 1 \end{cases}$$

MergeSort - Costo con Albero

$T(n)$

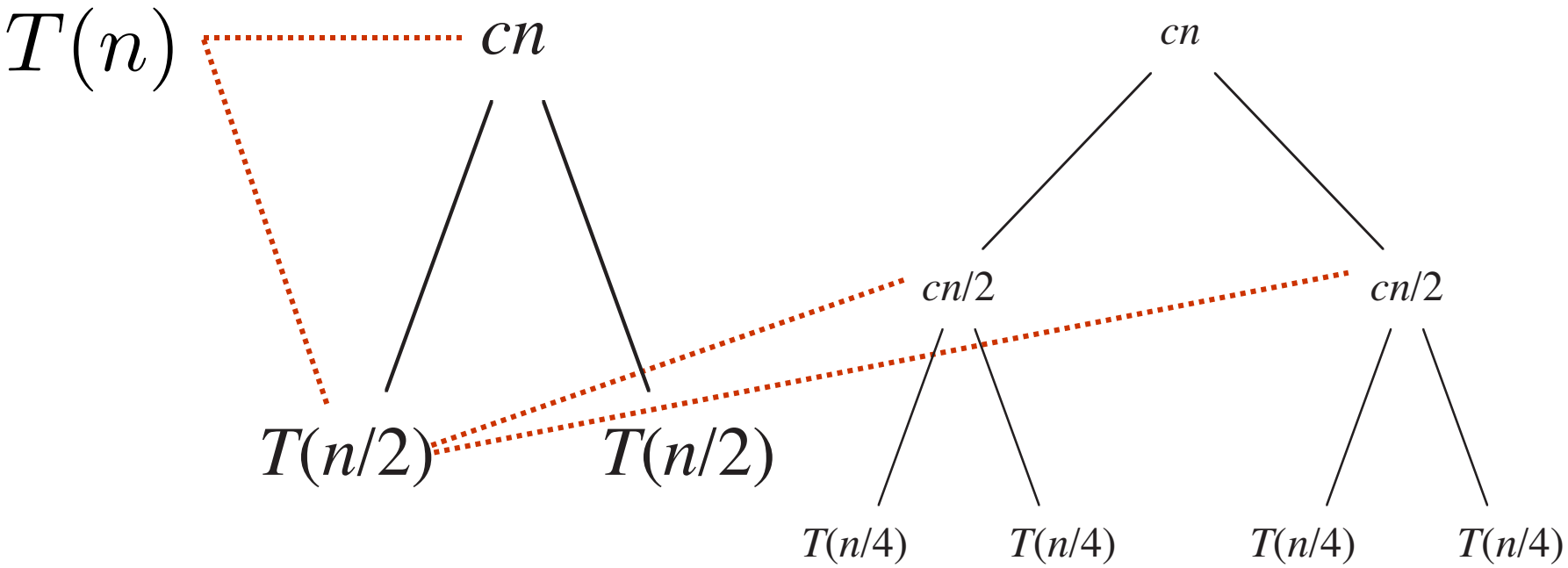
$$T(n) = \begin{cases} c & \text{se } n = 1 \\ 2T(\frac{n}{2}) + cn & \text{se } n > 1 \end{cases}$$

MergeSort - Costo con Albero



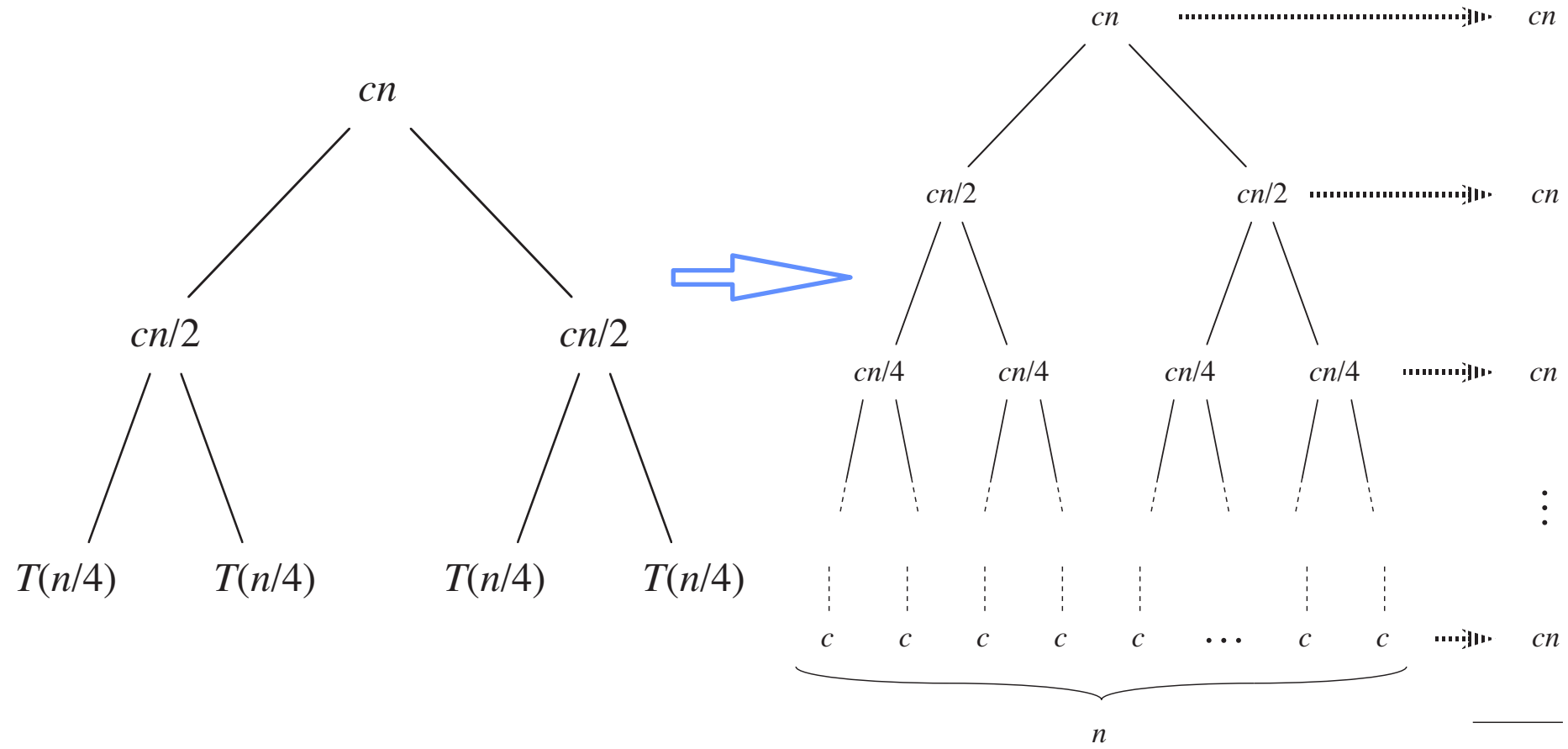
$$T(n) = \begin{cases} c & \text{se } n = 1 \\ 2T(\frac{n}{2}) + cn & \text{se } n > 1 \end{cases}$$

MergeSort - Costo con Albero



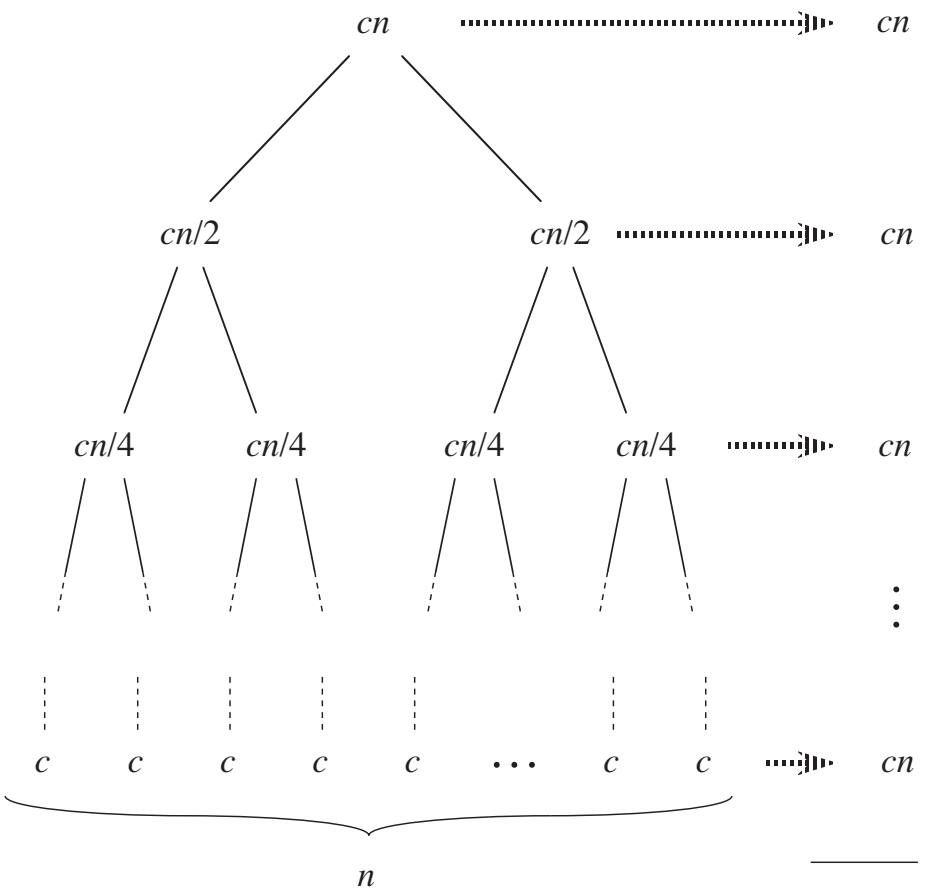
$$T(n) = \begin{cases} c & \text{se } n = 1 \\ 2T(\frac{n}{2}) + cn & \text{se } n > 1 \end{cases}$$

MergeSort - Costo con Albero



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

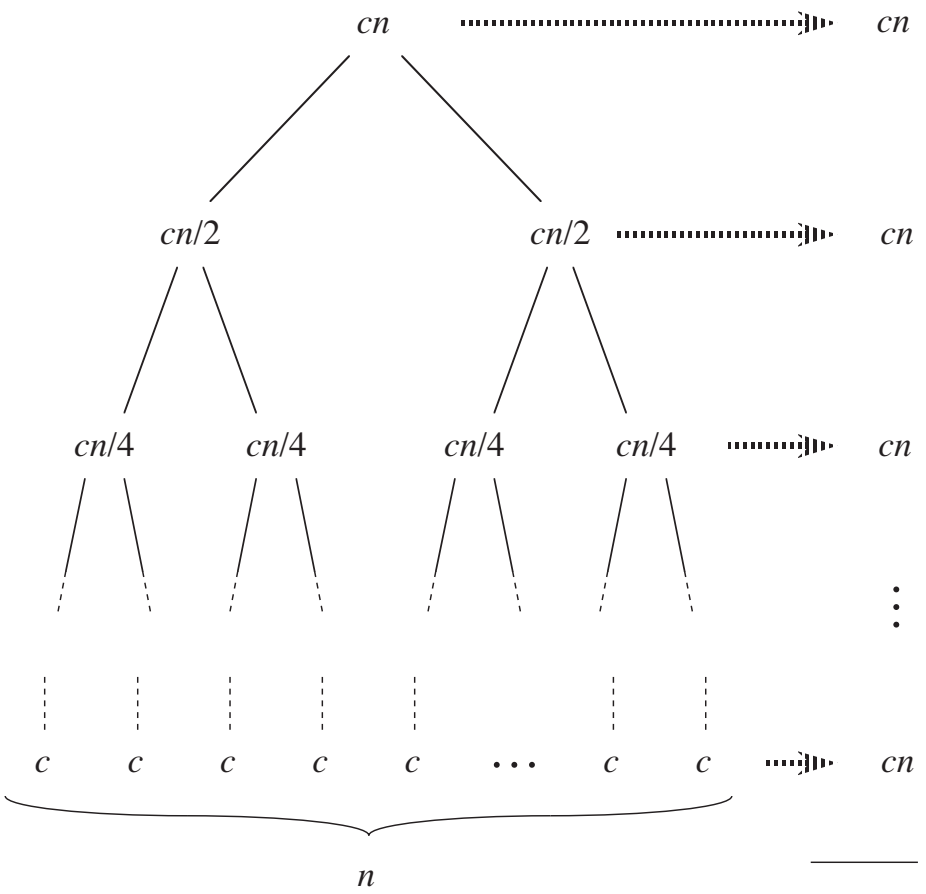
MergeSort - Costo con Albero



Numero di livelli?

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

MergeSort - Costo con Albero

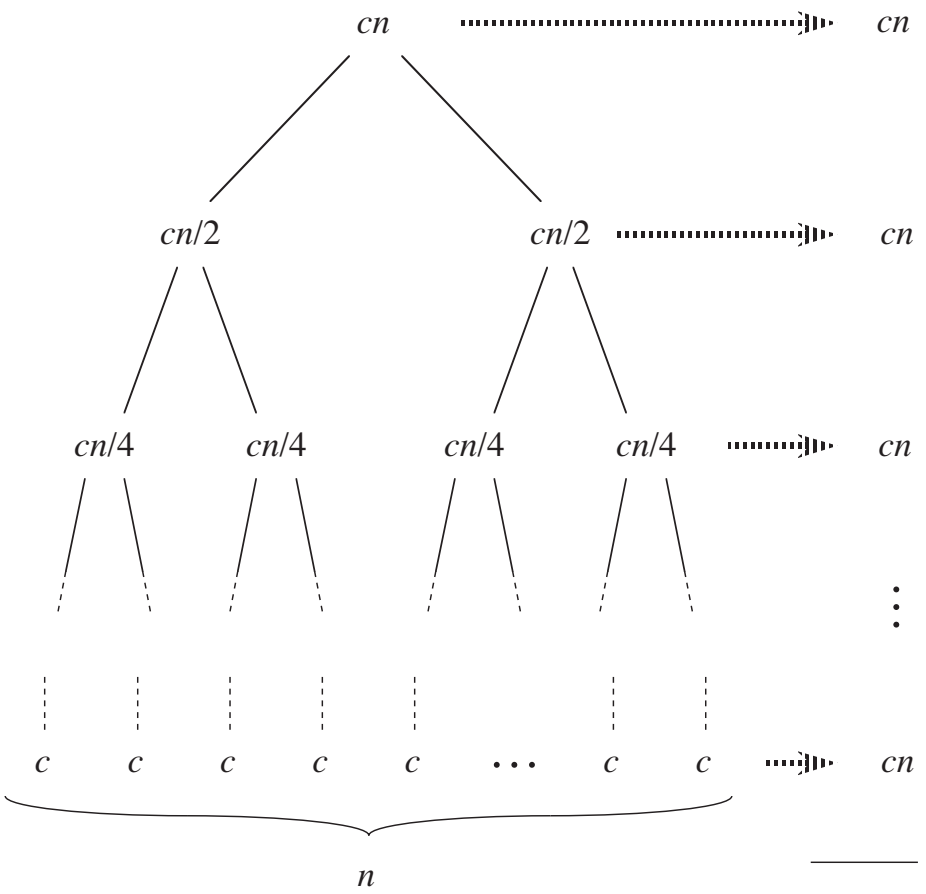


Numero di livelli?

logn + 1

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

MergeSort - Costo con Albero



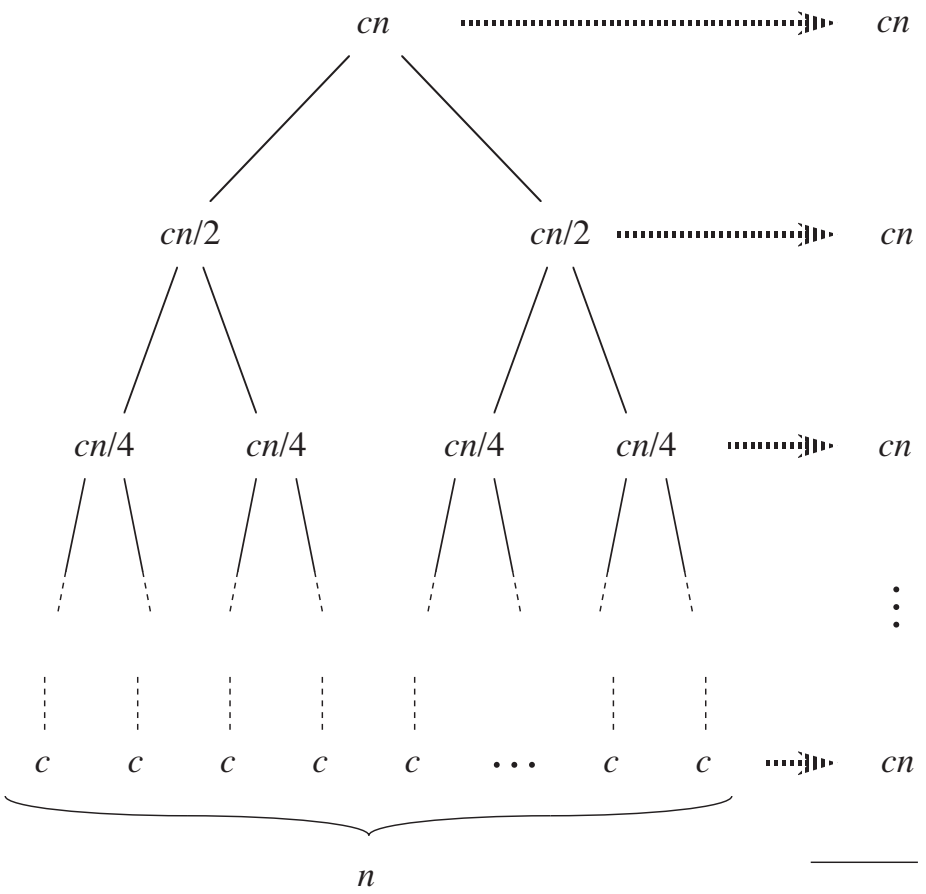
Numero di livelli?

logn + 1

Quindi il costo totale è di….?

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

MergeSort - Costo con Albero



Numero di livelli?

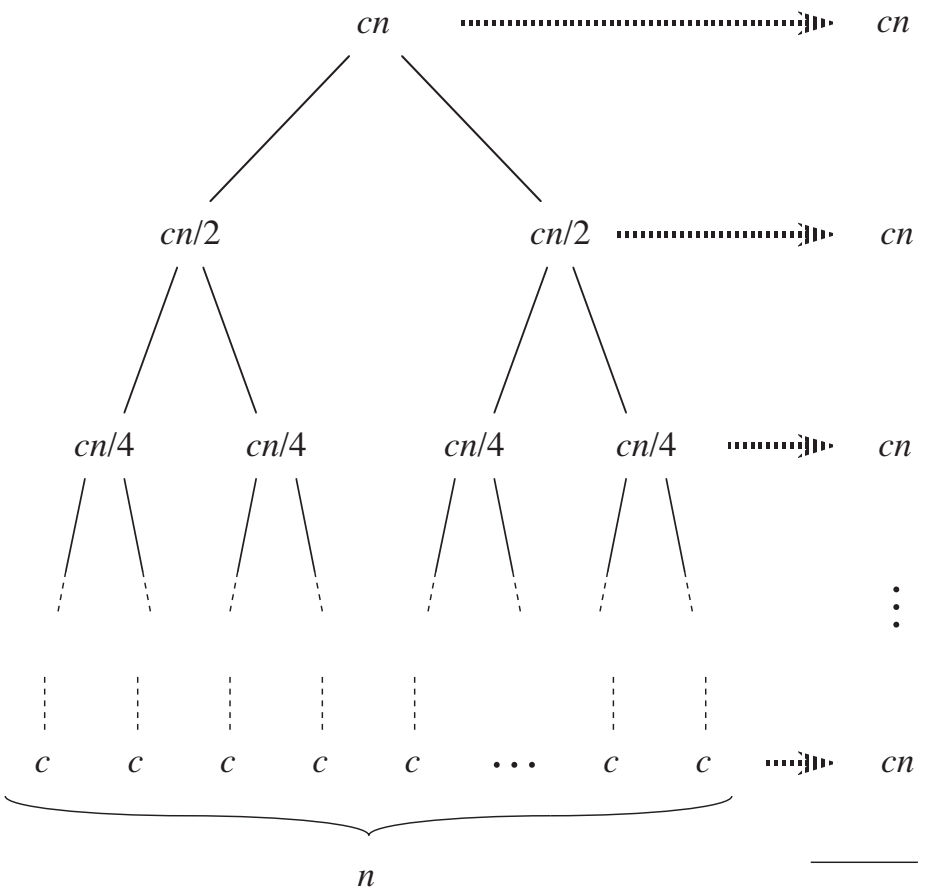
logn + 1

Quindi il costo totale è di….?

cnlogn + cn

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

MergeSort - Costo con Albero



Numero di livelli?

logn + 1

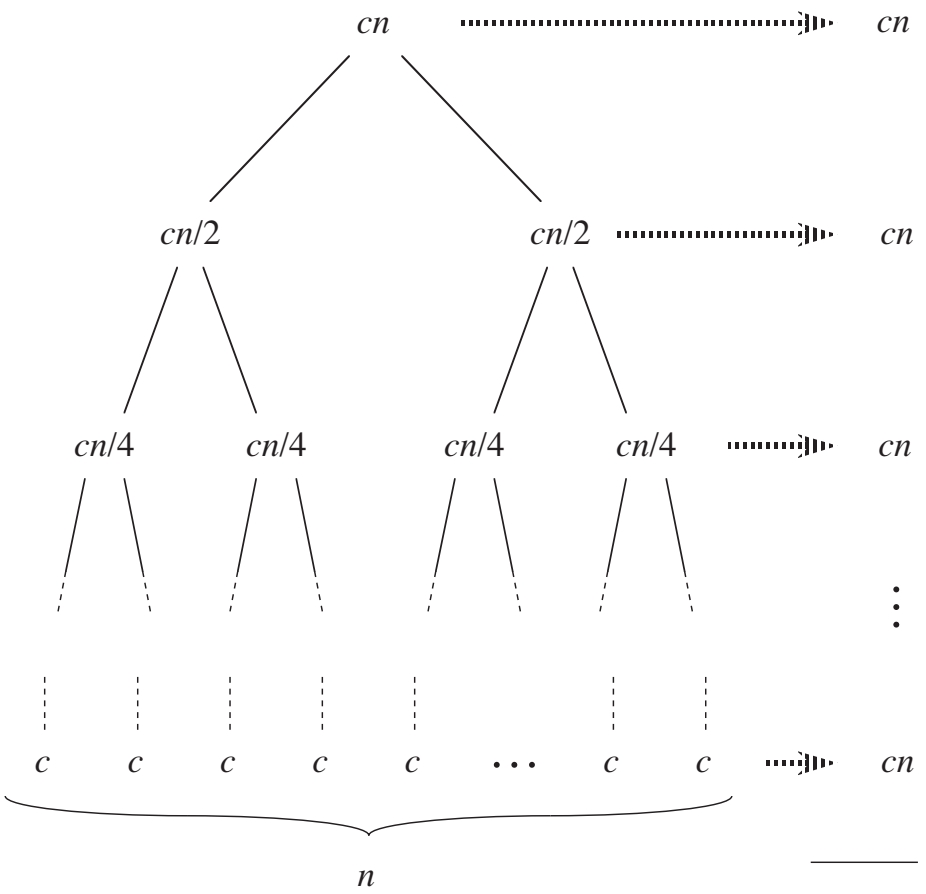
Quindi il costo totale è di….?

cnlogn + cn

Θ(?)

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

MergeSort - Costo con Albero



Numero di livelli?

logn + 1

Quindi il costo totale è di….?

cnlogn + cn

Θ(nlogn)

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

Mergesort in place?

Alg.: MERGE(A, p, q, r)

1. Calcola n_1 e n_2
2. Copia i primi n_1 elementi in
   $L[0 \,..\, n_1]$ e i successivi n_2 in $R[0 \,..\, n_2]$
3. $L[n_1] = \infty$; $R[n_2] = \infty$
4. i = 0; j = 0;
5. **for** (k = p; k<=r; k++)
6. **if** L[i] ≤ R[j]
7. **then** *A*[k] = L[i]
8. i =i + 1
9. **else** *A*[k] = R[j]
10. j = j + 1

Questa versione del MergeSort (implementazioni che trovate in pratica) non è in place (vengono usati array di appoggio)

Necessario?

Mergesort in place?

Alg.: MERGE(A, p, q, r)

1. Calcola n_1 e n_2
2. Copia i primi n_1 elementi in
   $L[0 \,..\, n_1]$ e i successivi n_2 in $R[0 \,..\, n_2]$
3. $L[n_1] = \infty$; $R[n_2] = \infty$
4. i = 0; j = 0;
5. **for** (k = p; k<=r; k++)
6. **if** L[i] ≤ R[j]
7. **then** A[k] = L[i]
8. i =i + 1
9. **else** A[k] = R[j]
10. j = j + 1

Questa versione del MergeSort (implementazioni che trovate in pratica) non è in place (vengono usati array di appoggio)

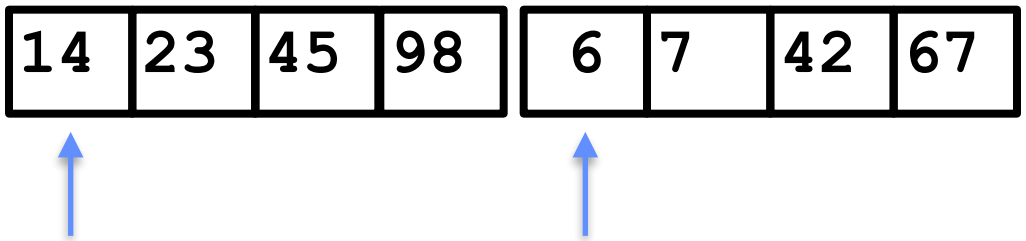
Necessario?

Ni!!!!

Mergesort in place?

Nì!!!!

La parte che cambierebbe in realtà è il **Merge()**, che deve fare tutto in-place….

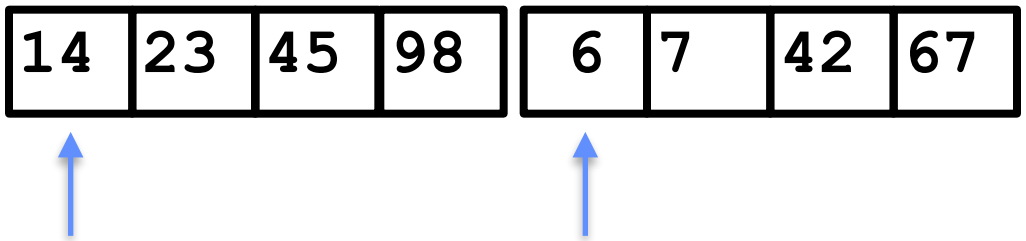


Mergesort in place?

Nì!!!!

La parte che cambierebbe in realtà è il **Merge()**, che deve fare tutto in-place….

6 < 14



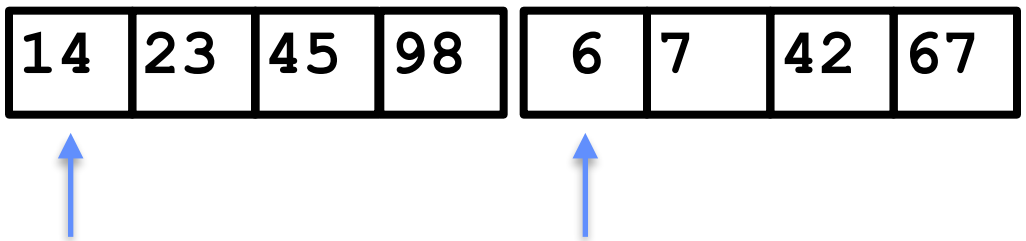
Mergesort in place?

Nì!!!!

La parte che cambierebbe in realtà è il **Merge()**, che deve fare tutto in-place….

Non devo distruggere l'ordinamento parziale delle due sotto sequenze!!!!

6 < 14



Cosa devo fare?

Mergesort in place?

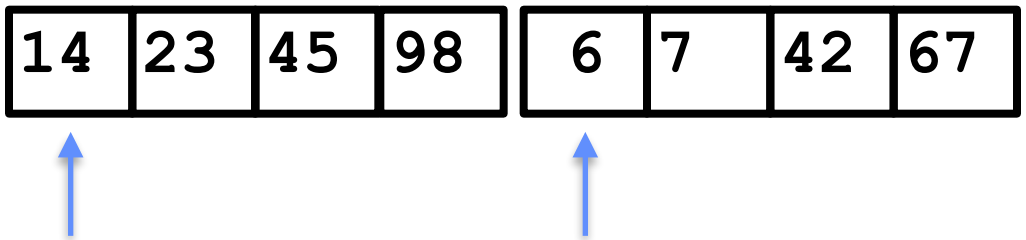
Nì!!!!

La parte che cambierebbe in realtà è il **Merge()**, che deve fare tutto in-place….

Non devo distruggere l'ordinamento parziale delle due sotto sequenze!!!!

Quindi posso semplicemente fare lo scambio?

6 < 14



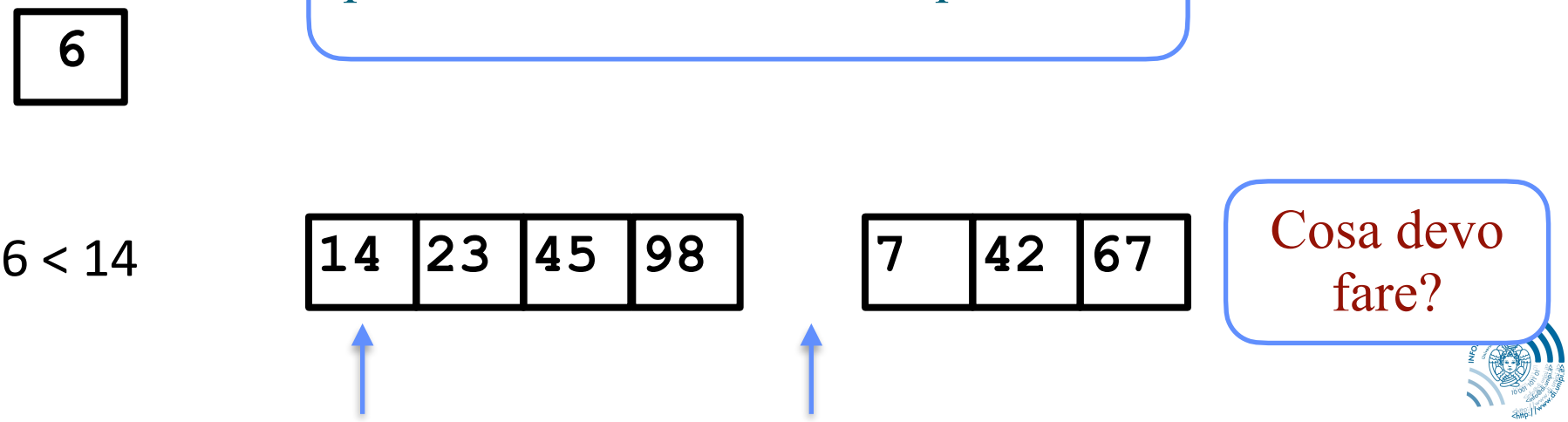
Cosa devo fare?

Mergesort in place?

Nì!!!!

La parte che cambierebbe in realtà è il **Merge()**, che deve fare tutto in-place….

Non devo distruggere l'ordinamento parziale delle due sotto sequenze!!!!

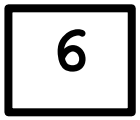


Mergesort in place?

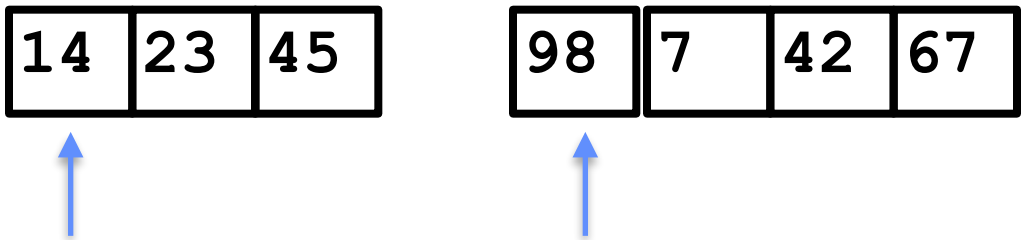
Nì!!!!

La parte che cambierebbe in realtà è il **Merge()**, che deve fare tutto in-place….

Non devo distruggere l'ordinamento parziale delle due sotto sequenze!!!!



6 < 14



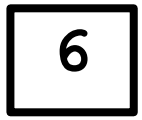
Cosa devo fare?

Mergesort in place?

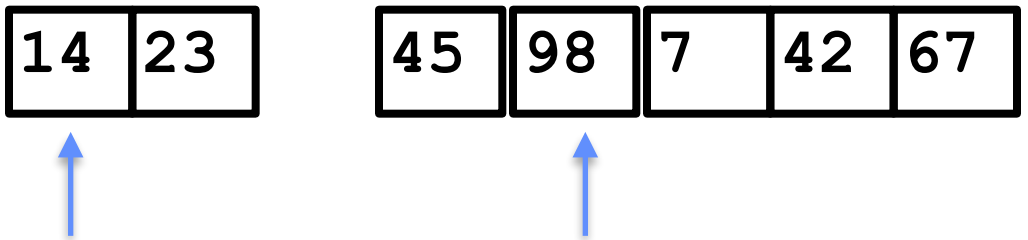
Nì!!!!

La parte che cambierebbe in realtà è il **Merge()**, che deve fare tutto in-place….

Non devo distruggere l'ordinamento parziale delle due sotto sequenze!!!!



6 < 14



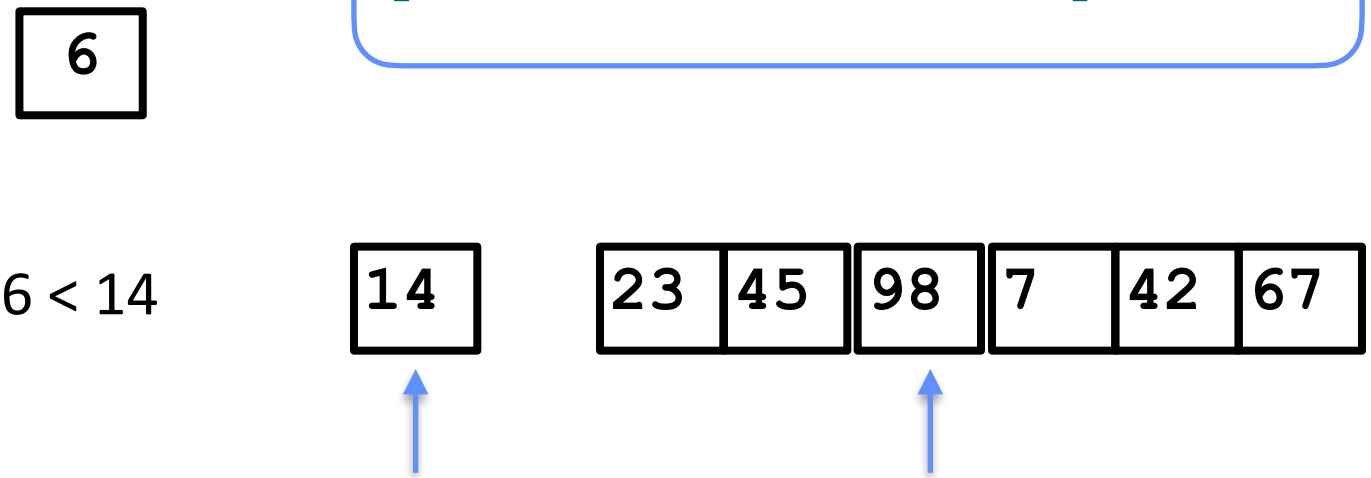
Cosa devo fare?

Mergesort in place?

Nì!!!!

La parte che cambierebbe in realtà è il **Merge()**, che deve fare tutto in-place….

Non devo distruggere l'ordinamento parziale delle due sotto sequenze!!!!



Cosa devo fare?

Mergesort in place?

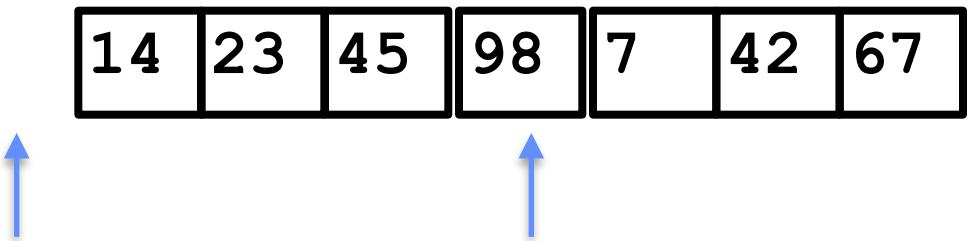
Nì!!!!

La parte che cambierebbe in realtà è il **Merge()**, che deve fare tutto in-place….

Non devo distruggere l'ordinamento parziale delle due sotto sequenze!!!!

6

6 < 14



Cosa devo fare?

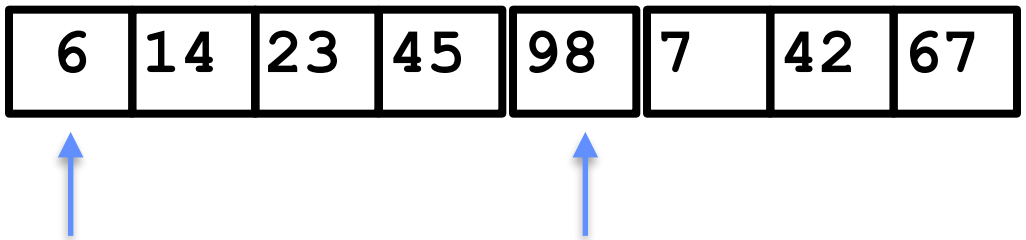
Mergesort in place?

Nì!!!!

La parte che cambierebbe in realtà è il **Merge()**, che deve fare tutto in-place….

Non devo distruggere l'ordinamento parziale delle due sotto sequenze!!!!

6 < 14



Cosa devo fare?

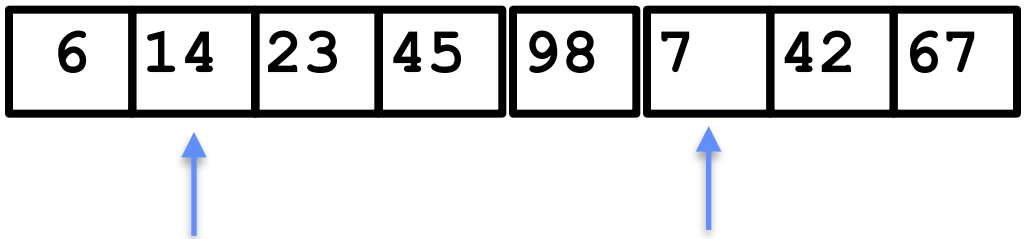
Mergesort in place?

Nì!!!!

La parte che cambierebbe in realtà è il **Merge()**, che deve fare tutto in-place….

Non devo distruggere l'ordinamento parziale delle due sotto sequenze!!!!

6 < 14



Cosa devo fare?

Mergesort in place?

Nì!!!!

La parte che cambierebbe in realtà è il **Merge()**, che deve fare tutto in-place….

Non devo distruggere l'ordinamento parziale delle due sotto sequenze!!!!

7

6 < 14

6	14	23	45	98

42	67

Cosa devo fare?

Mergesort in place?

Nì!!!!

La parte che cambierebbe in realtà è il **Merge()**, che deve fare tutto in-place….

7

Non devo distruggere l'ordinamento parziale delle due sotto sequenze!!!!

6 < 14

6	14	23	45

98	42	67

Cosa devo fare?

Mergesort in place?

Nì!!!!

La parte che cambierebbe in realtà è il **Merge()**, che deve fare tutto in-place….

Non devo distruggere l'ordinamento parziale delle due sotto sequenze!!!!

7

6 < 14

6	14	23

45	98	42	67

Cosa devo fare?

Mergesort in place?

Nì!!!!

La parte che cambierebbe in realtà è il **Merge()**, che deve fare tutto in-place….

7

Non devo distruggere l'ordinamento parziale delle due sotto sequenze!!!!

6 < 14

6	14

23	45	98	42	67

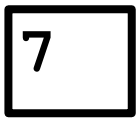
Cosa devo fare?

Mergesort in place?

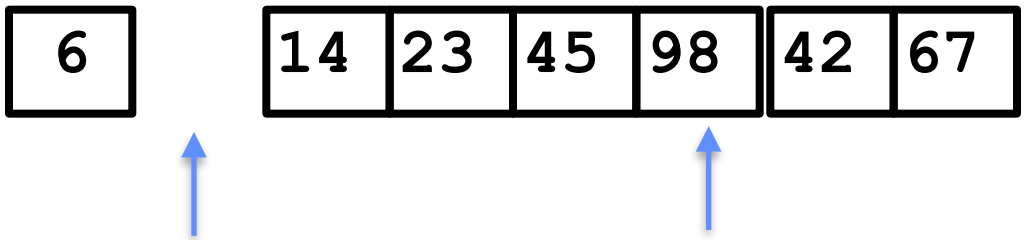
Nì!!!!

La parte che cambierebbe in realtà è il **Merge()**, che deve fare tutto in-place….

Non devo distruggere l'ordinamento parziale delle due sotto sequenze!!!!



6 < 14



Cosa devo fare?

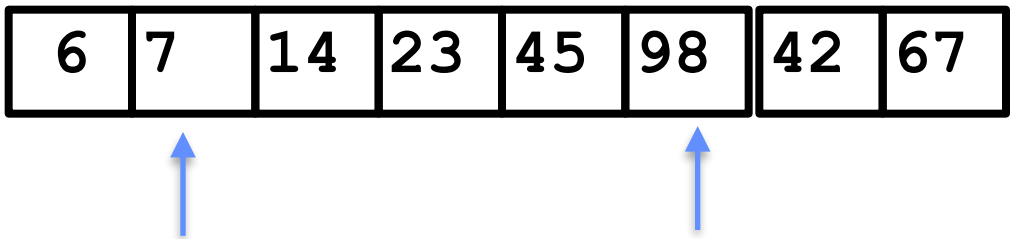
Mergesort in place?

Nì!!!!

La parte che cambierebbe in realtà è il **Merge()**, che deve fare tutto in-place….

Non devo distruggere l'ordinamento parziale delle due sotto sequenze!!!!

6 < 14



Cosa devo fare?

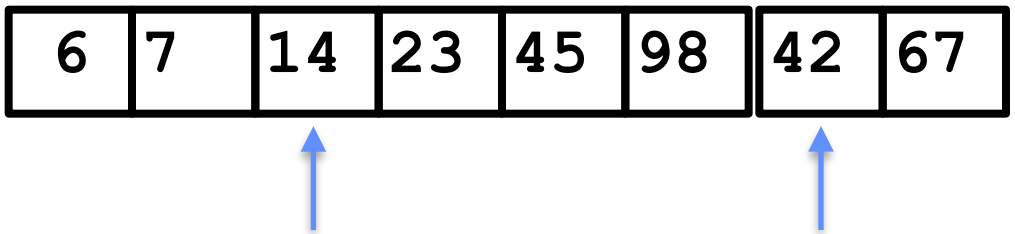
Mergesort in place?

Nì!!!!

La parte che cambierebbe in realtà è il **Merge()**, che deve fare tutto in-place….

Non devo distruggere l'ordinamento parziale delle due sotto sequenze!!!!

6 < 14



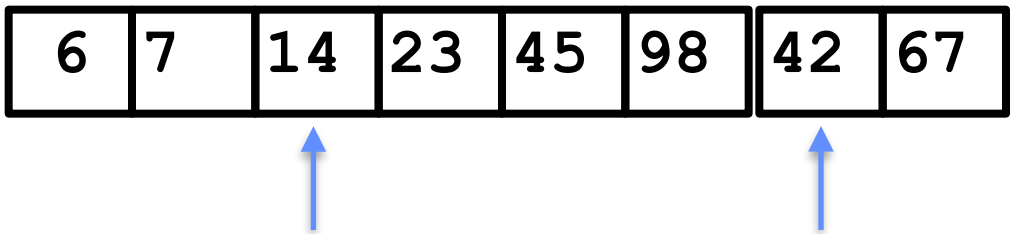
Cosa devo fare?

Mergesort in place?

Nì!!!!

Nel caso peggiore quindi quanto costa (sia n il numero totale di elementi)?

$6 < 14$



Cosa devo fare?

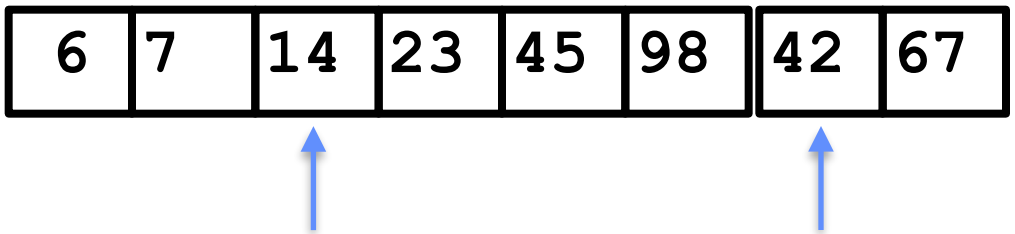
Mergesort in place?

Nì!!!!

Nel caso peggiore quindi quanto costa (sia n il numero totale di elementi)?

$O(n^2)$

$6 < 14$



Cosa devo fare?

Sommario

Algoritmo	Tempo Migliore	Tempo Peggiore	Note
selection-sort	$O(n^2)$	$O(n^2)$	• lento • in-place • per piccoli insiemi (< 1K)
insertion-sort	$O(n)$	$O(n^2)$	• lento • in-place • per piccoli insiemi (< 1K)
merge-sort	$O(n \log n)$	$O(n \log n)$	• veloce • per grandi insiemi (> 1M)