# Analisi algoritmi ricorsivi e relazioni di ricorrenza

# Punto della situazione

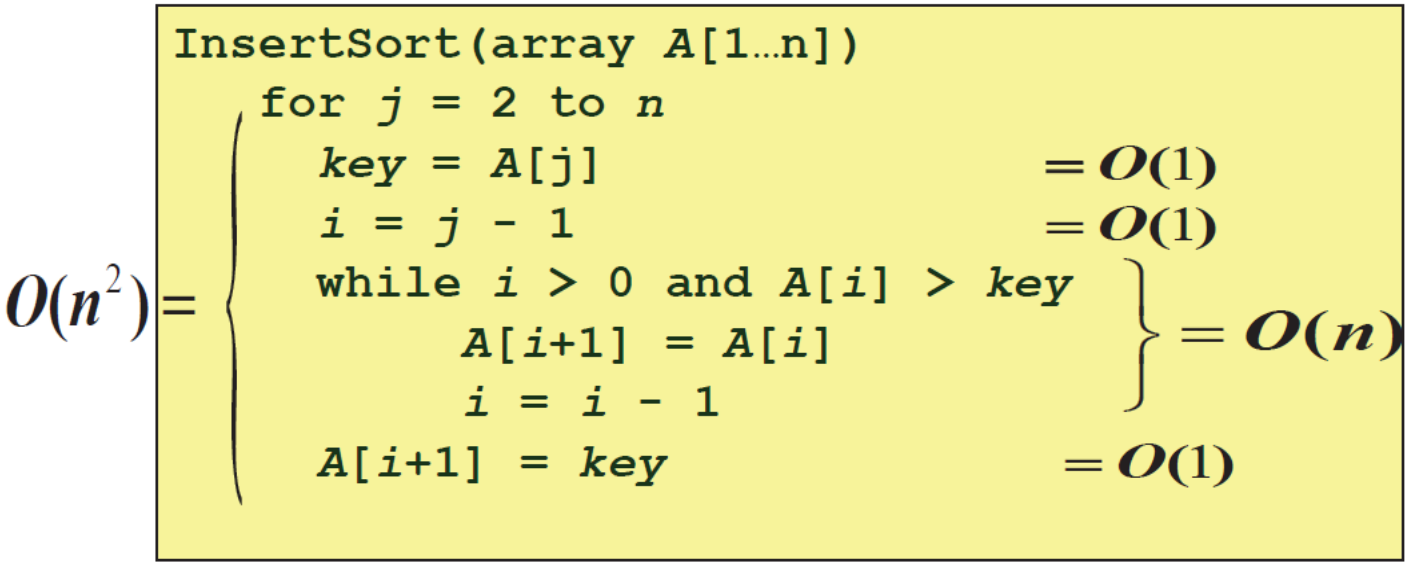
- Finora abbiamo affrontato: il tempo di esecuzione di un algoritmo, l'analisi asintotica con le notazioni asintotiche e la tecnica divide-et-impera che dà luogo ad algoritmi ricorsivi.
- Adesso: come si analizza il tempo di esecuzione degli algoritmi ricorsivi? Introduzione e risoluzione delle relazioni di ricorrenza
- Riprenderemo il MergeSort….
- Cominciamo col fissare le idee sul calcolo del tempo di esecuzione di algoritmi non ricorsivi

# Calcolo della complessità di tempo

- il costo di istruzioni semplici, quali assegnazione, letture/scrittura di variabili é $O(1)$
- il costo di un blocco sequenziale di istruzioni è pari alla somma dei costi delle singole istruzioni
- il costo di istruzioni tipo **if.... then ....else** é pari al tempo per effettuare il test (tipicamente $O(1)$) piú $O(\text{costo della alternativa piú costosa})$
- il costo di loop (**for, while, repeat**) é pari alla somma su tutte le iterazioni del costo di ogni iterazioni
- Usando queste regole si puó ottenere la complessitá di tempo di ogni algoritmo (ad eccezione degli algoritmi ricorsivi, che richiedono una tecnica diversa)

# Esempio: InsertionSort

Algoritmo di ordinamento di A[1…n] ottenuto mantenendo ad ogni iterazione A[1…j-1] ordinato e inserendovi A[j].



# Analisi di InsertionSort

$O(n^2) =$

```
InsertSort(array A[1…n])
   for j = 2 to n
      key = A[j]                          = O(1)
      i = j - 1                           = O(1)
      while i > 0 and A[i] > key
            A[i+1] = A[i]                 } = O(n)
            i = i - 1
      A[i+1] = key                        = O(1)
```

Più precisamente:

Fissato j, il test del `while` è eseguito un numero di volte compreso fra 1 e j. Da cui

$T(n) \leq \sum_{j=2}^{n} j = \frac{n(n+1)}{2} - 1$ e quindi $T(n) = O(n^2)$.

Inoltre $T(n) = \Omega(n)$

# Divide – et - Impera

- Dividi il problema in sottoproblemi
- Risolvi ogni sottoproblema ricorsivamente
- Combina le soluzioni ai sottoproblemi per ottenere la soluzione  al problema

# Algoritmi ricorsivi

Schema di un algoritmo ricorsivo (su un'istanza $\mathcal{I}$ ):

ALGO ($\mathcal{I}$)

If «caso base» then «esegui certe operazioni»

else «esegui delle operazioni fra le quali

ALGO($\mathcal{I}_1$), … , ALGO($\mathcal{I}_k$) »

# Mergesort

Mergesort su una sequenza ***S*** con ***n*** elementi consiste di tre passi:

1. Divide: separa ***S*** in due sequenze ***S***1 e ***S***2, ognuna di circa ***n***/2 elementi;

2. Ricorsione: ricorsivamente ordina ***S***1 e ***S***2

3. Conquer (impera): unisci ***S***1 e ***S***2 in un'unica sequenza *ordinata*

# Merging

- Merging. Combine two pre-sorted lists into a sorted whole.

- How to merge efficiently?
  - Linear number of comparisons. $T_{MERGE}(n) = \Theta(n)$
  - Use temporary array.

# Mergesort

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

Supponendo che:
la sequenza sia data come un array A[p, … ,r] con n= r-p+1 elementi,
MERGE(A, p, q, r) «fonda» correttamente le sequenze A[p,…, q] e A[q+1, …, r]

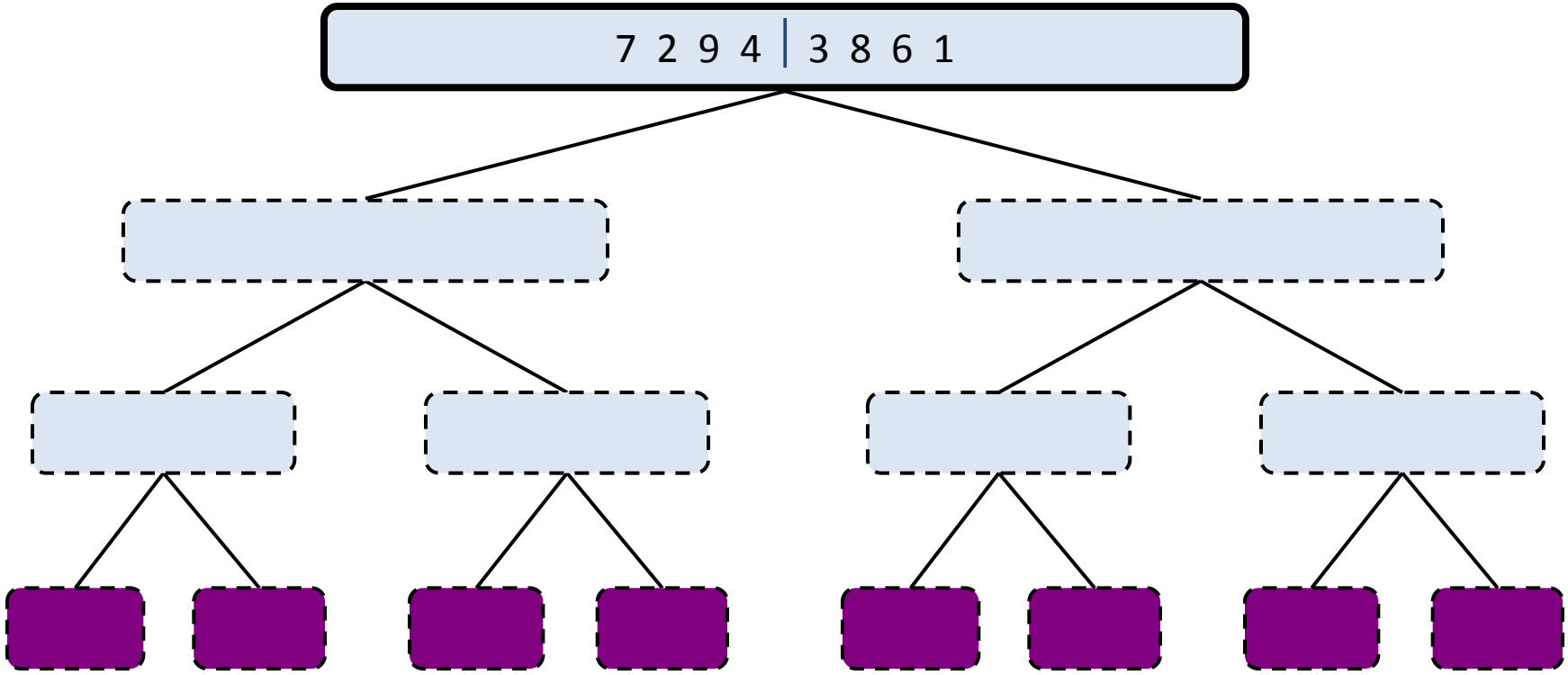
# Memento

- Ricordare che in un algoritmo ricorsivo non si può procedere oltre una chiamata ricorsiva se questa non è stata completata del tutto.

- Quindi nel `MERGE-SORT`, l'esecuzione della linea 4 comincia una volta finita l'esecuzione della linea 3, cioè completata la chiamata ricorsiva `MERGE-SORT(A,p,q)` con tutte le sue sotto-chiamate.

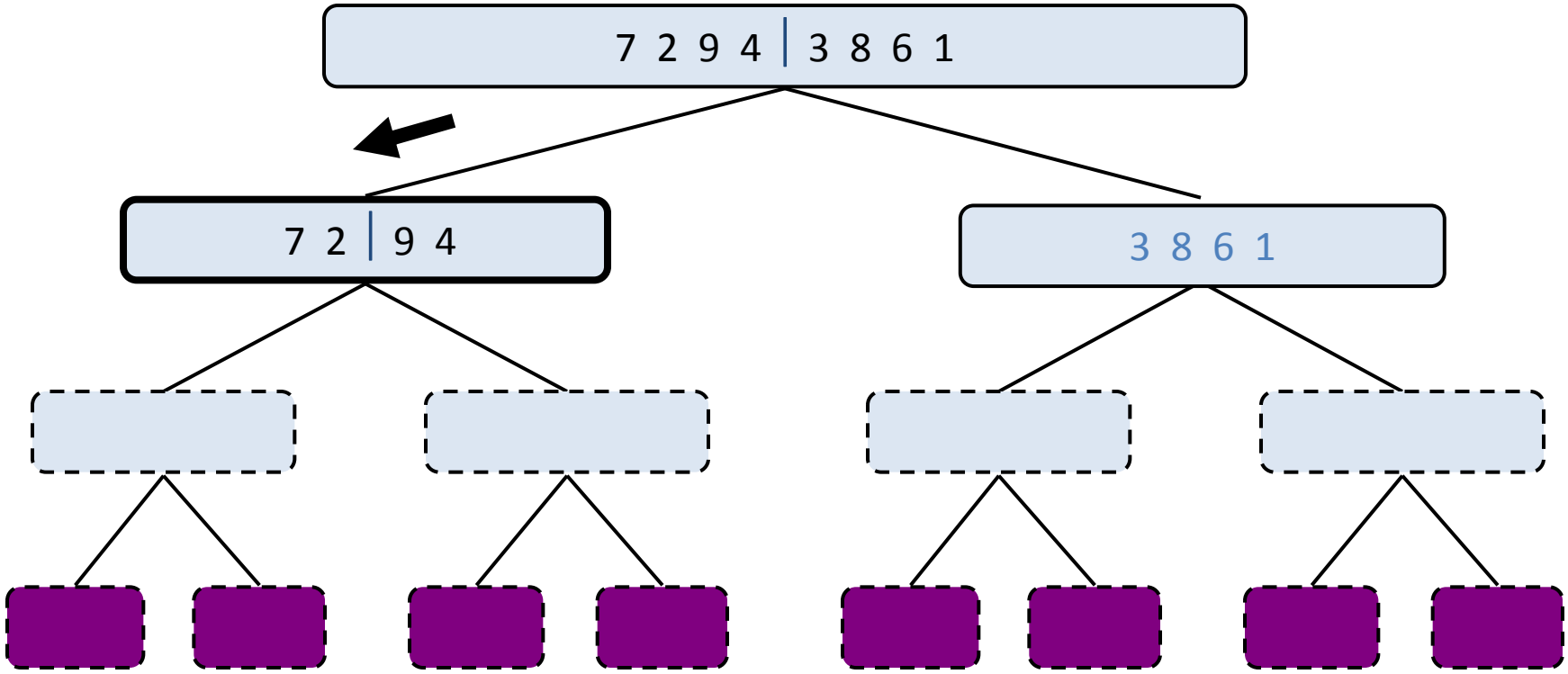
# Esempio di esecuzione di MergeSort

- Divide



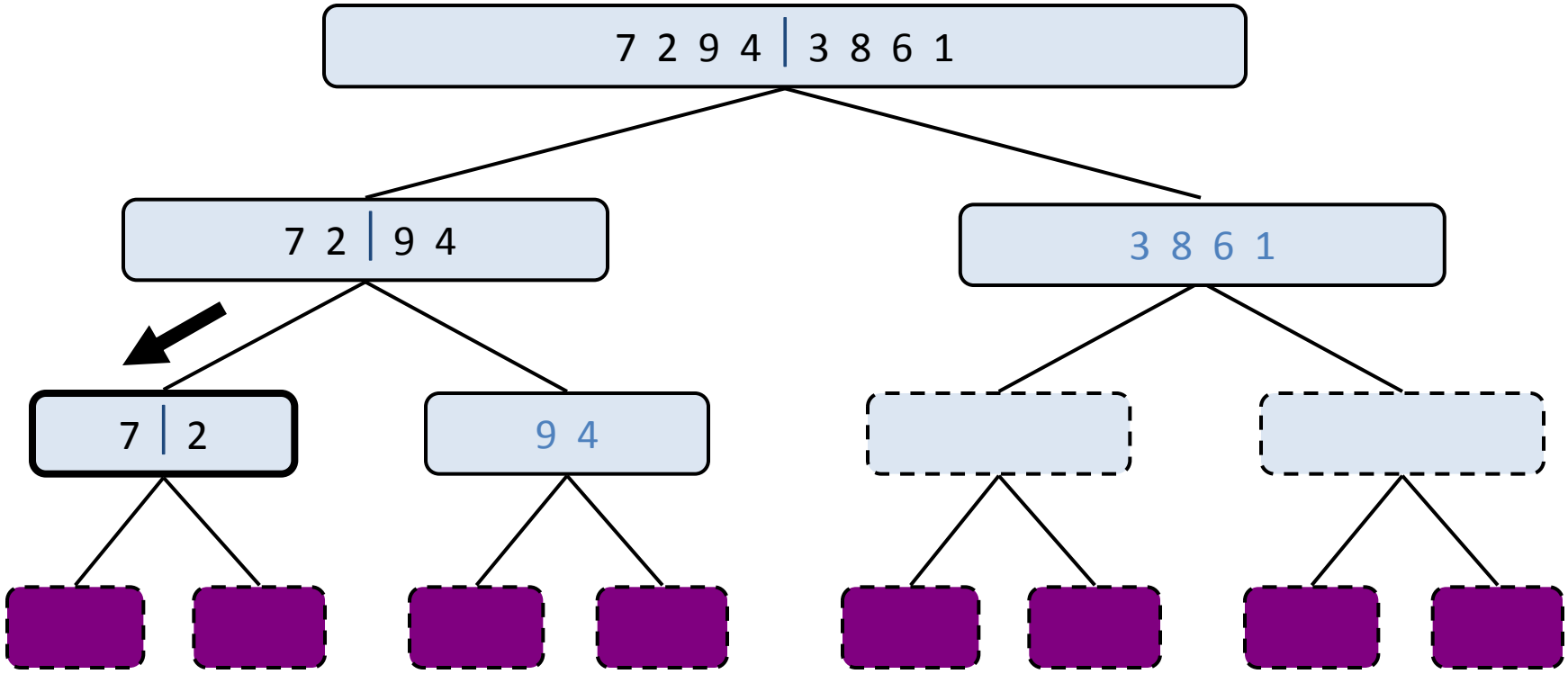
# Esempio di esecuzione (cont.)

- Chiamata ricorsiva, divide



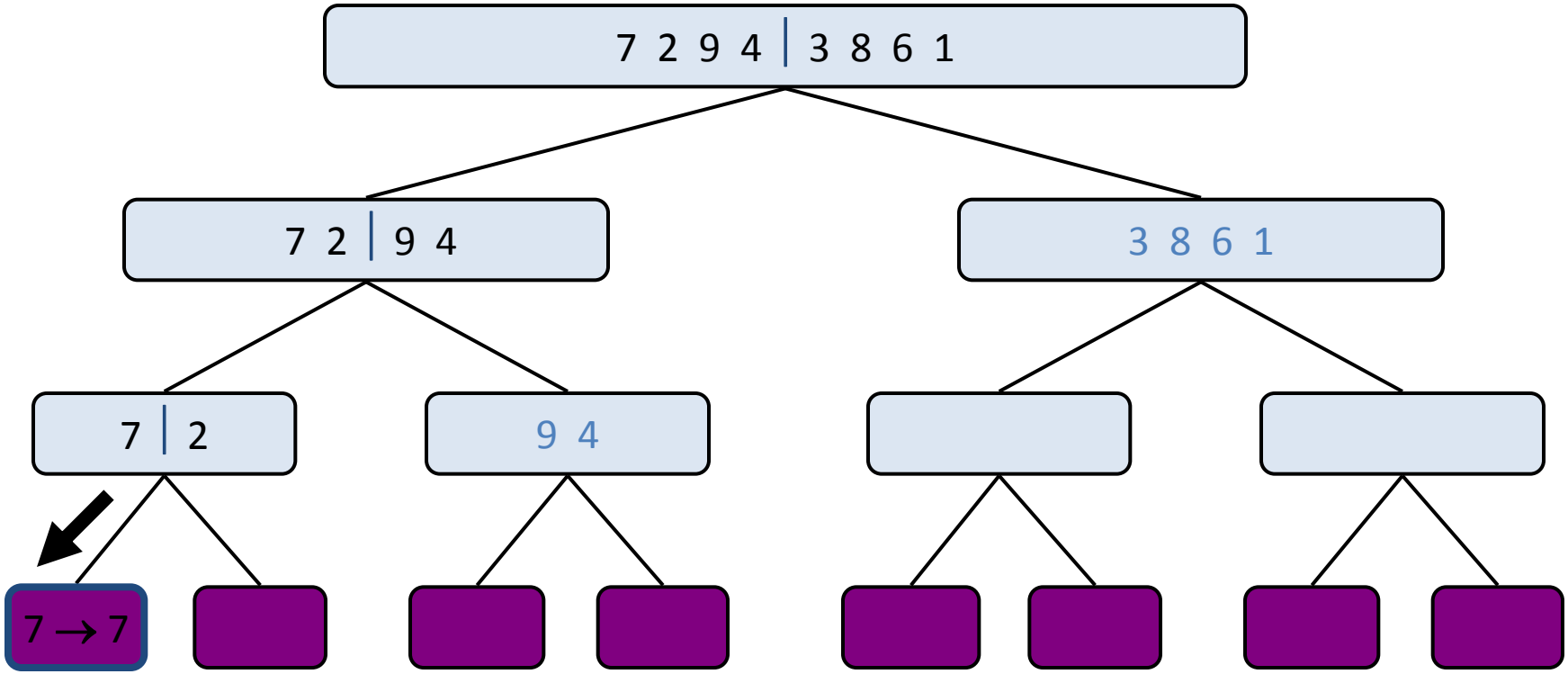
# Esempio di esecuzione (cont.)

- Chiamata ricorsiva, divide



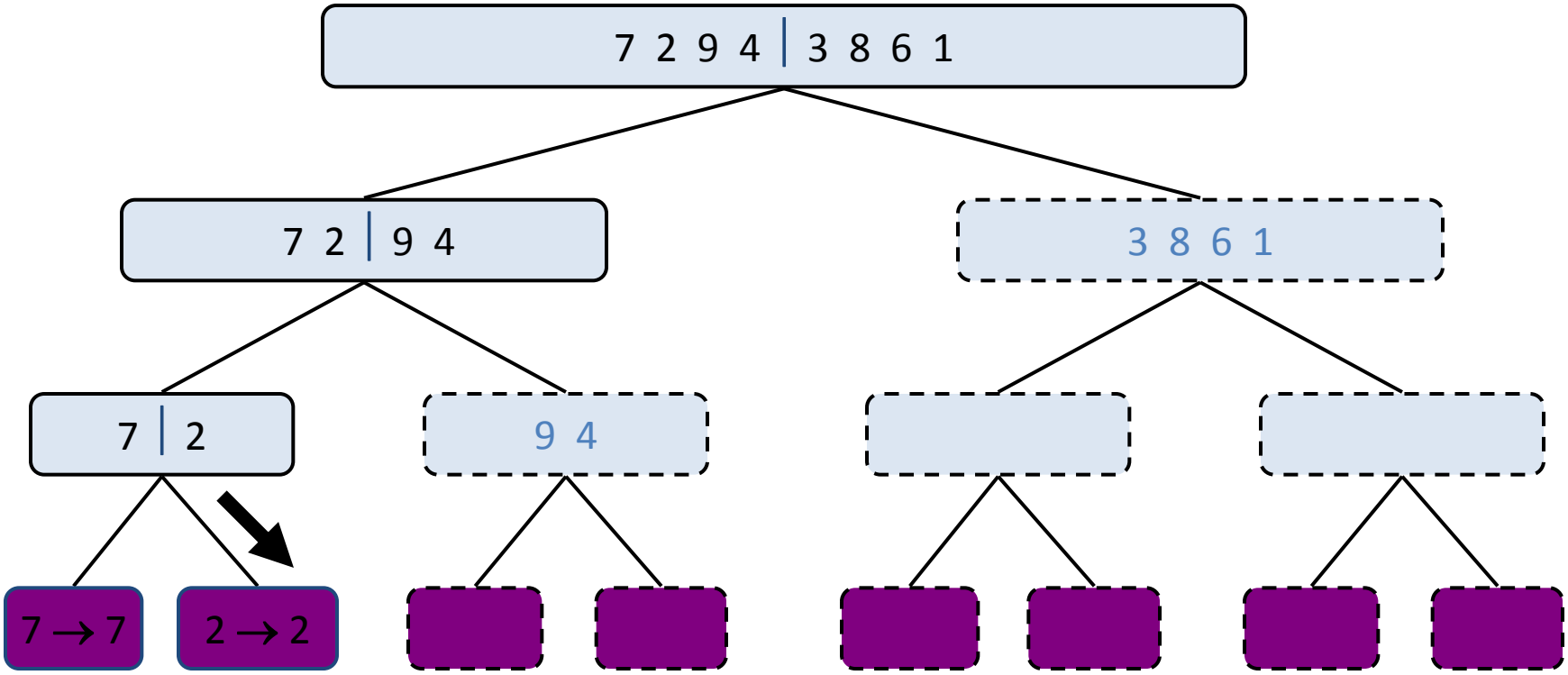
# Esempio di esecuzione (cont.)

- Chiamata ricorsiva: caso base



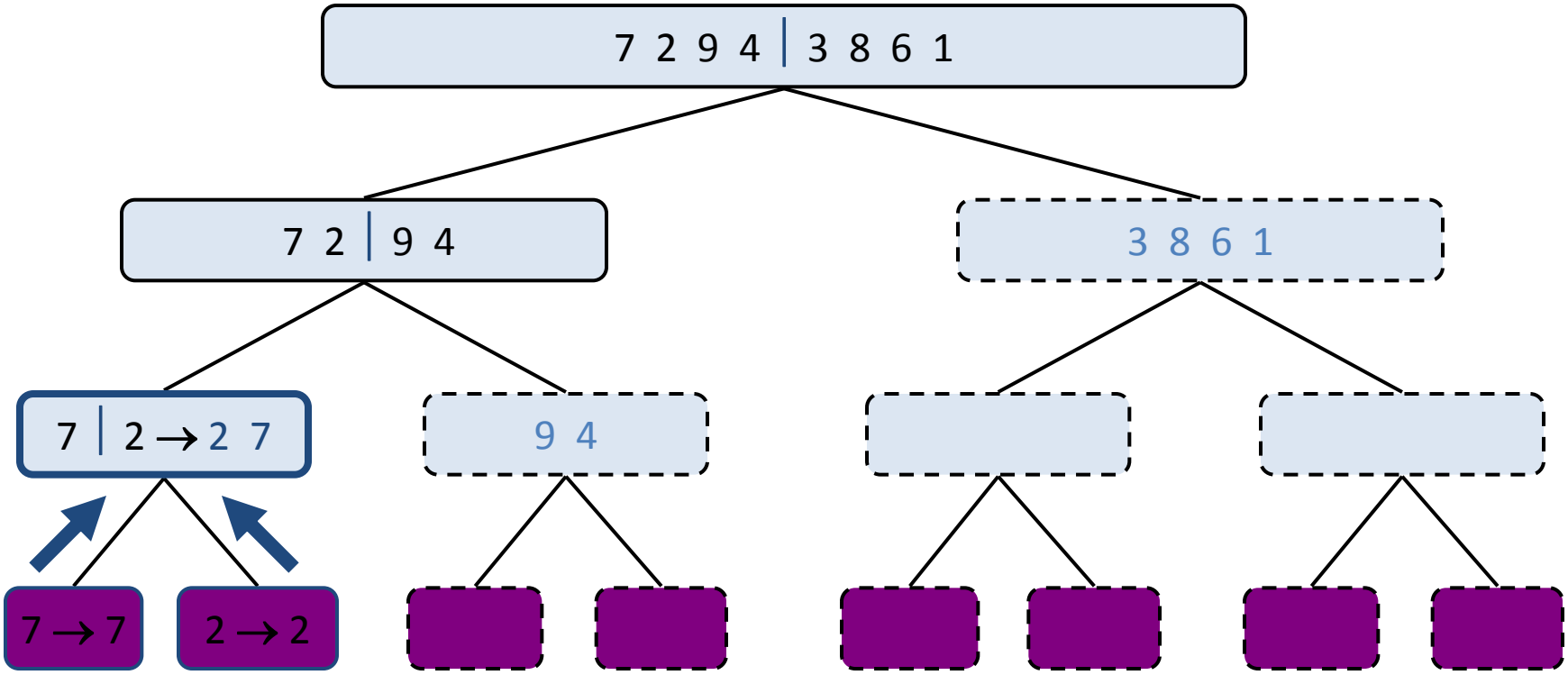
# Esempio di esecuzione (cont.)

- Chiamata ricorsiva: caso base



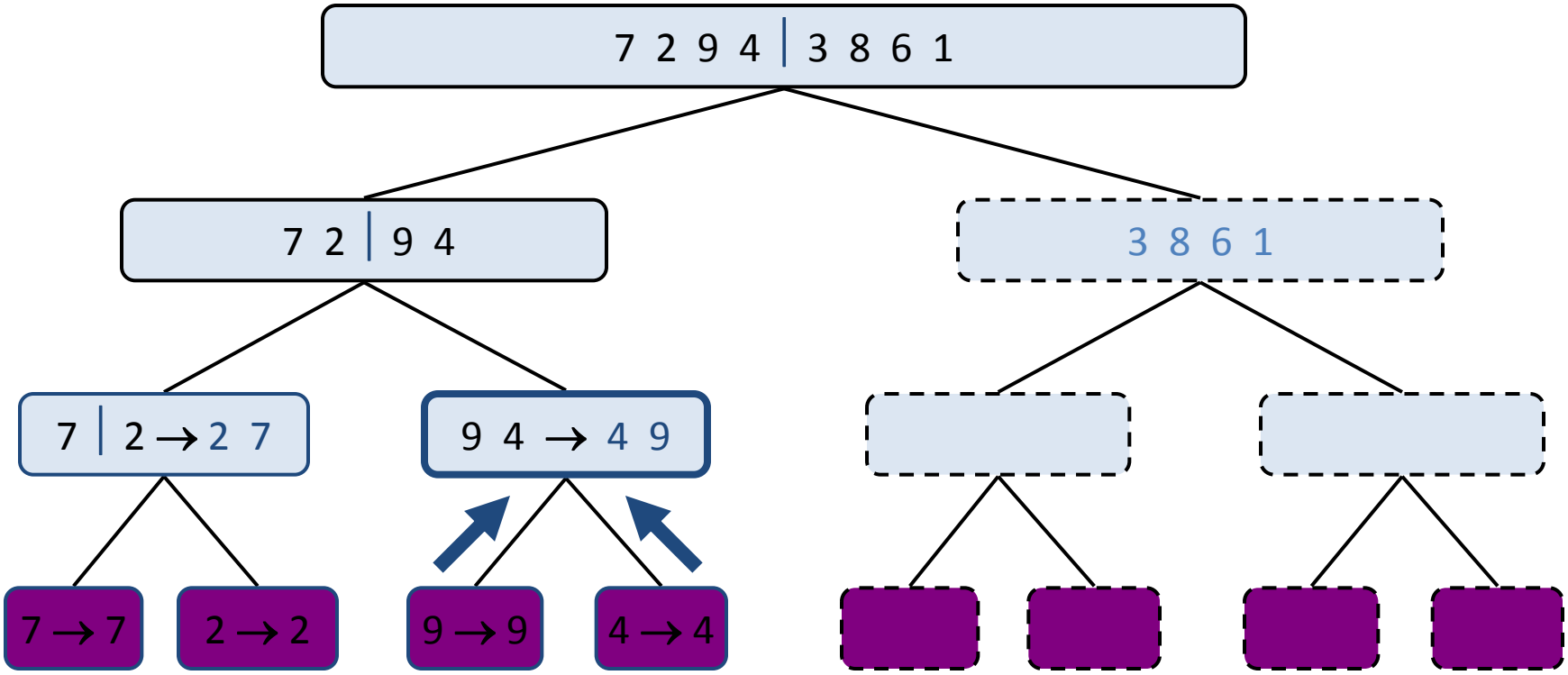
# Esempio di esecuzione (cont.)

- Merge



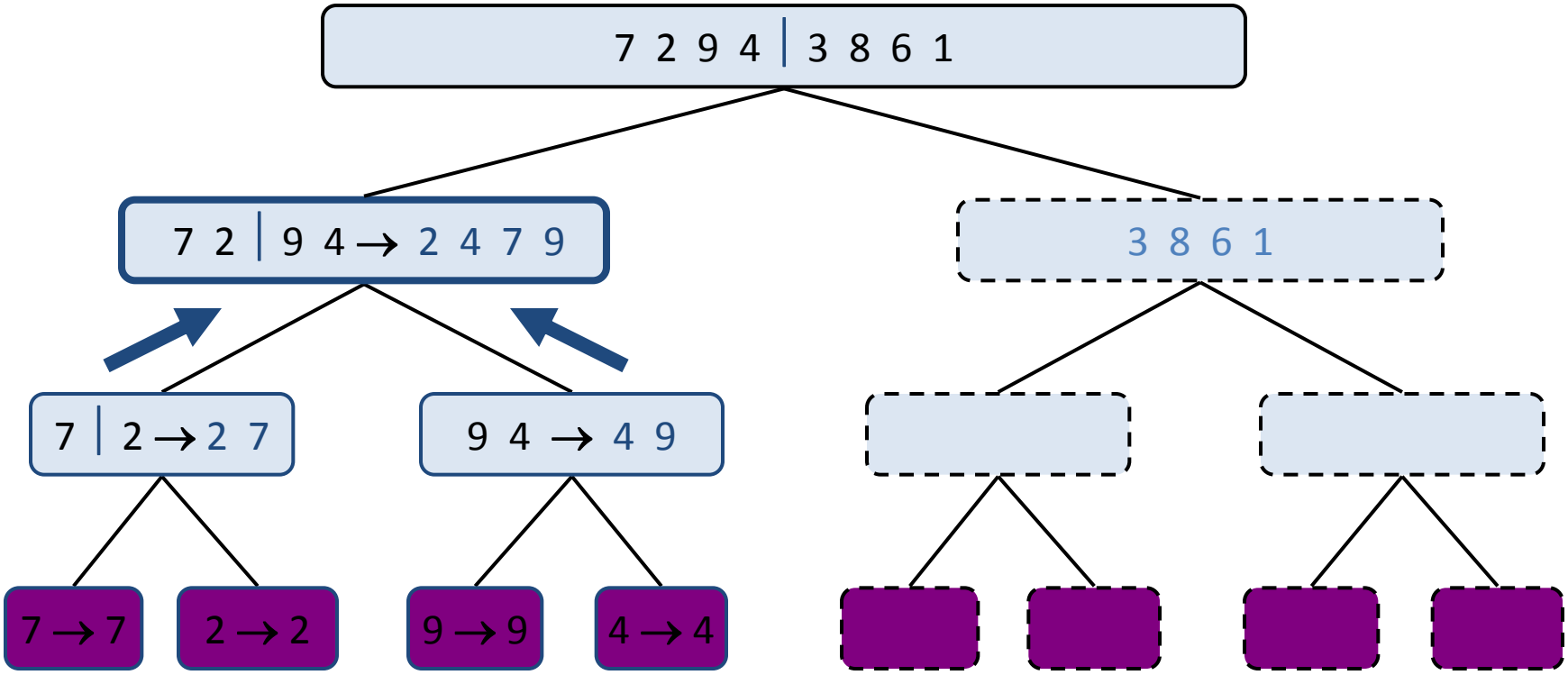
# Esempio di esecuzione (cont.)

- Chiamata ricorsiva, …, caso base, merge



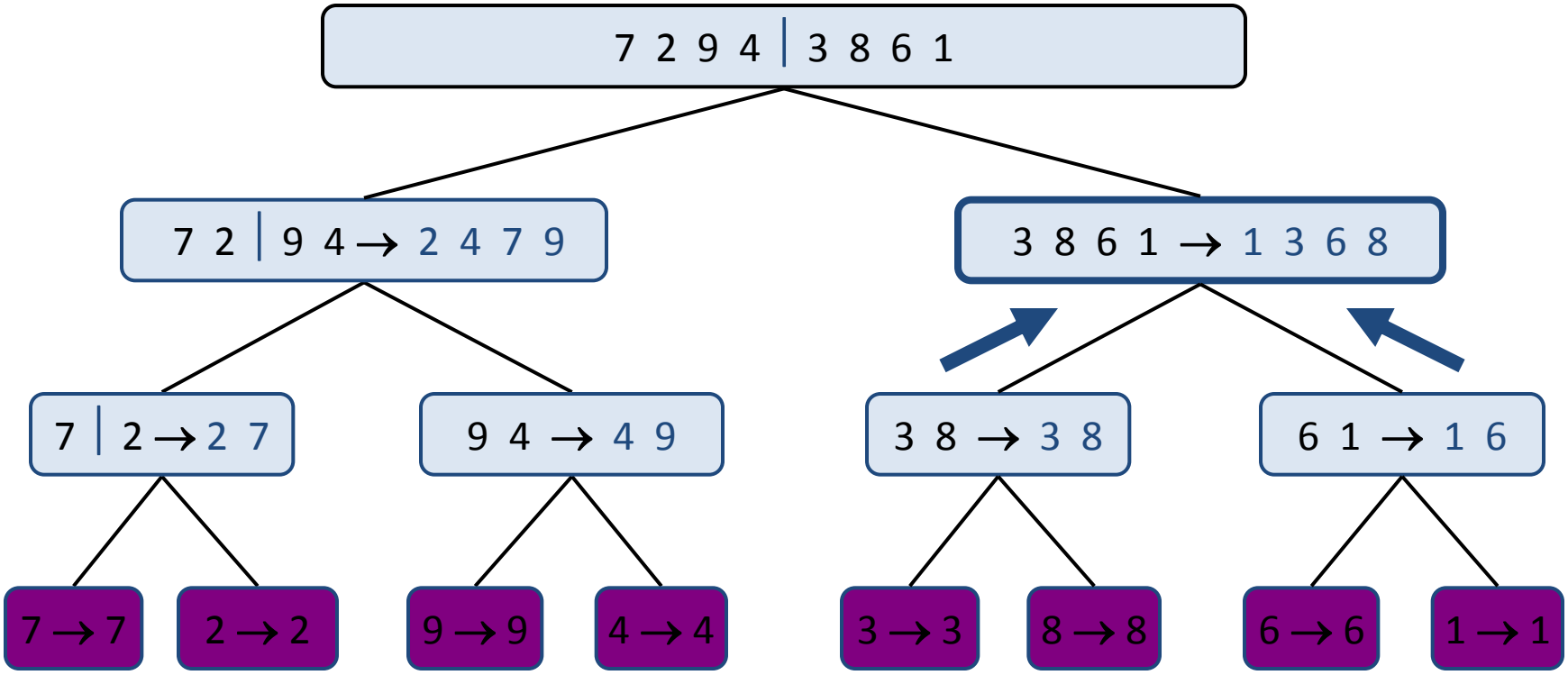
# Esempio di esecuzione (cont.)

- Merge



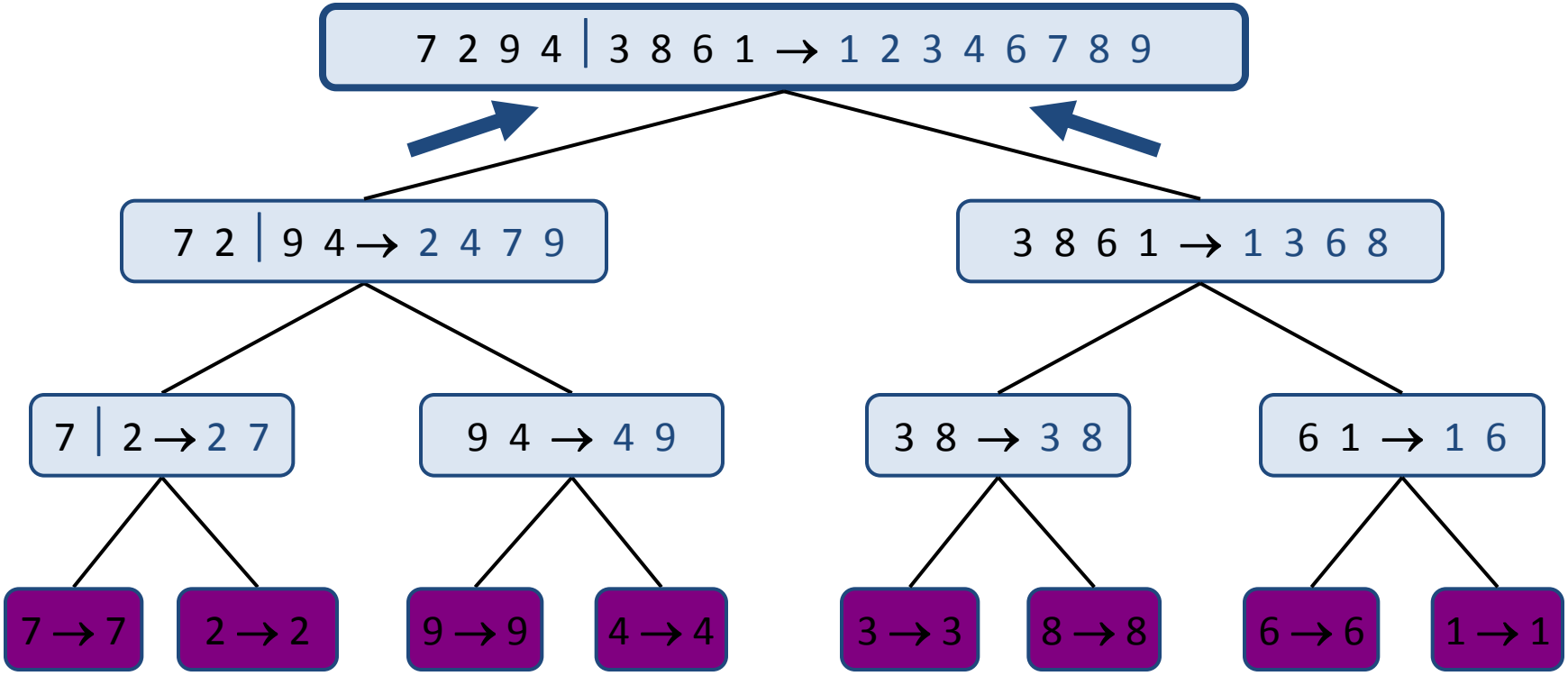
# Esempio di esecuzione (cont.)

- Chiamata ricorsiva, …, merge, merge



# Esempio di esecuzione (fine)

- Ultimo Merge



# Tempo di esecuzione di Mergesort

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

Sia T(n) il tempo di esecuzione di Mergesort su un array di taglia n.

**T(n) = ?**

Come si calcola il tempo di esecuzione di un algoritmo ricorsivo?

E in particolare:

Come si calcola il tempo di esecuzione di un algoritmo Divide et Impera?

# Tempo di esecuzione di Mergesort

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

Sia **T(n)** il tempo di esecuzione di Mergesort su un array di taglia n.

**T(n) = ?**

C'è 1 confronto nella linea 1: $\Theta(1)$, e poi eventualmente un assegnamento $\Theta(1)$ e l'esecuzione del MERGE: $O(n)$;
per un totale di al più: $\Theta(1) + \Theta(1) + O(n) = O(n)$.
E poi **2** esecuzioni di MERGESORT su due array di circa **n/2** elementi, cioè?

$$T(n) \leq \Theta(1) + \Theta(1) + O(n) + 2\,T(n/2)$$

# A Recurrence Relation for Mergesort

Def. T(n) = number of comparisons to mergesort an input of size n.

Mergesort recurrence.

$$T(n) \le \begin{cases} \Theta(1) & \text{if } n = 1 \\ \underbrace{T(\lceil n/2 \rceil)}_{\text{solve left half}} + \underbrace{T(\lfloor n/2 \rfloor)}_{\text{solve right half}} + \underbrace{\Theta(n)}_{\text{merging}} & \text{otherwise} \end{cases}$$

Solution. $T(n) = \Theta(n \log_2 n)$.

Assorted proofs.

We will describe several ways to prove this recurrence.

# A Recurrence Relation for Binary Search

Def. T(n) = number of comparisons to run Binary Search on an input of size n.

Binary Search recurrence.

$$T(n) \le \begin{cases} \Theta(1) & \text{if } n = 1 \\ \underbrace{T(\lfloor n/2 \rfloor)}_{\text{solve left or right half}} + \underbrace{\Theta(1)}_{\text{comparison}} & \text{otherwise} \end{cases}$$

Solution. T(n) = $O(\log_2 n)$

(T(n) constant in the best case).

# Relazioni di ricorrenza

Una relazione di ricorrenza per una funzione T(n) è un'equazione o una disequazione che esprime T(n) rispetto a valori di T su variabili più piccole, completata dal valore di T nel caso base (o nei casi base).

Esempio:

$$T(n)=\begin{cases} 5 & \text{se } n=1 \\ T(n/2)+3 & \text{altrimenti} \end{cases}$$

T(8) = ???
T(8) = T(4)+3
T(4) = T(2)+3
T(2) = T(1)+3 = 5+3=8
T(4) = 8+3=11
T(8) = 11+3=14

E se n = 10?

T(n) = ?  T(n) = 5 + 3 $\log_2 n$ per n potenza di 2

In generale ci basterà T(n) = $\Theta(\log n)$

# Relazioni di ricorrenza per algoritmi ricorsivi

$$T(n) = \begin{cases} c & \text{se } n = n_0 \\ aT(f(n)) + g(n) & \text{altrimenti} \end{cases}$$

- $n_0$ =base ricorsione, $c$ =tempo di esecuzione per la base
- $a$ =numero di volte che le chiamate ricorsive sono effettuate
- $f(n)$ =taglia dei problemi risolti nelle chiamate ricorsive
- $g(n)$ =tutto il tempo di calcolo non incluso nelle chiamate ricorsive

# Relazioni di ricorrenza per algoritmi Divide-et-Impera

- Dividi il problema di taglia $n$ in $a$ sotto-problemi di taglia $n/b$
- Ricorsione sui sottoproblemi
- Combinazione delle soluzioni

$T(n)$= tempo di esecuzione su input di taglia $n$

$T(n) = D(n) + a\,T(n/b) + C(n)$