UNIVERSITÀ PARTHENOPE
DiST
Dipartimento di Scienze e Tecnologie

Programmazione II e Laboratorio di P2

Merge Sort

Angelo Ciaramella

Introduzione

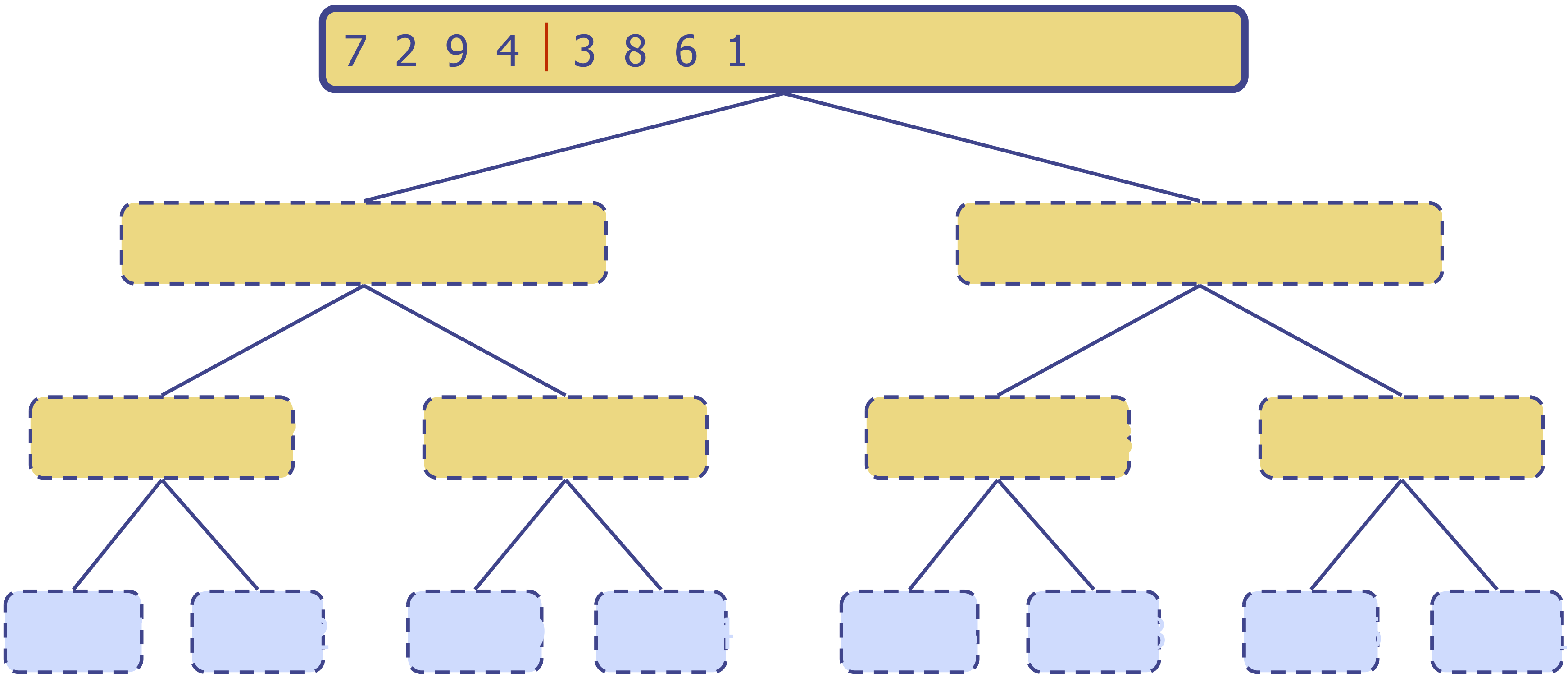
- Inventato da von Neumann nel 1945
- Esempio del paradigma algoritmico del divide et impera
- Richiede spazio ausiliario
  - O(*N*)

- Si divide il vettore dei dati in due parti ordinate separatamente, quindi si fondono le parti per ottenere un vettore ordinato globalmente

Complessità

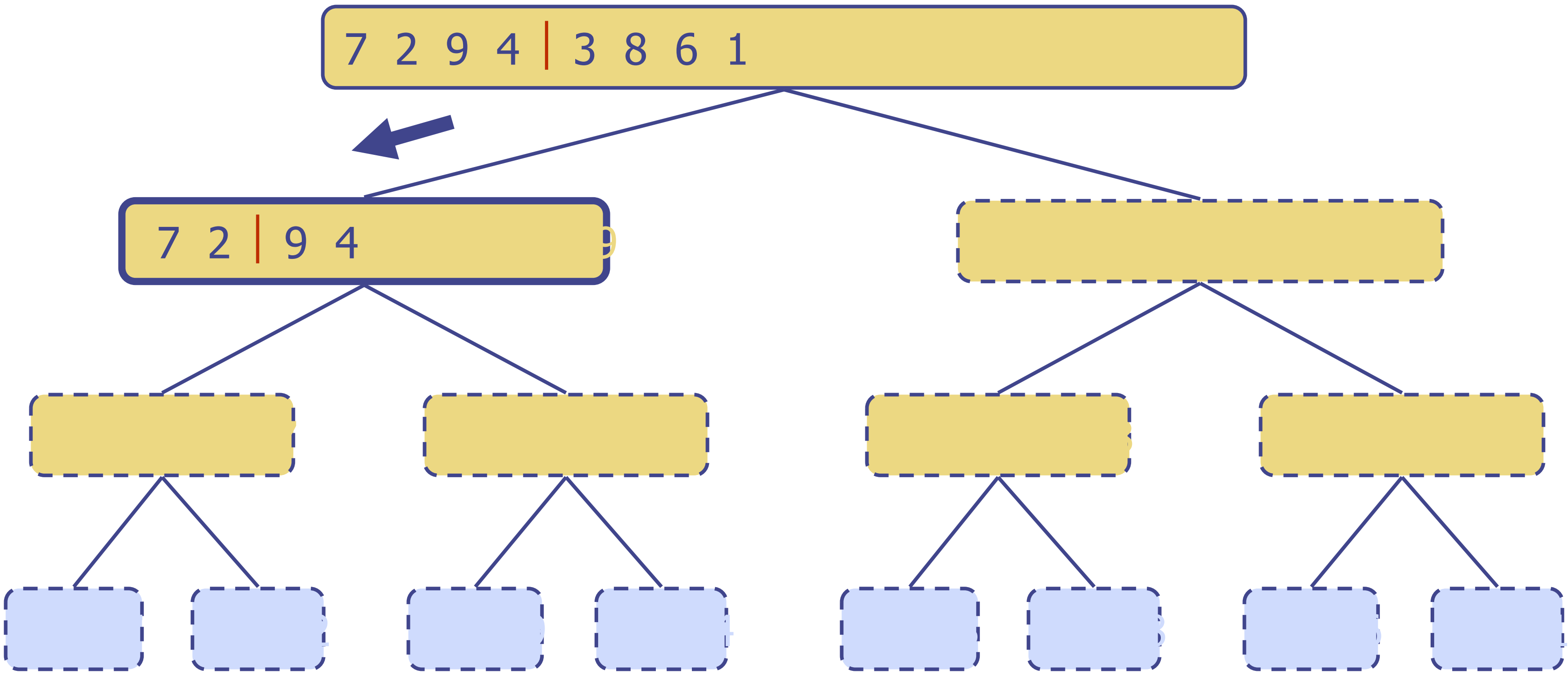
Algorithm	Time	Notes
selection-sort	$O(n^2)$	▪ slow ▪ in-place ▪ for small data sets (< 1K)
insertion-sort	$O(n^2)$	▪ slow ▪ in-place ▪ for small data sets (< 1K)
heap-sort	$O(n \log n)$	▪ fast ▪ in-place ▪ for large data sets (1K — 1M)
merge-sort	$O(n \log n)$	▪ fast ▪ sequential data access ▪ for huge data sets (> 1M)

Esempio di esecuzione

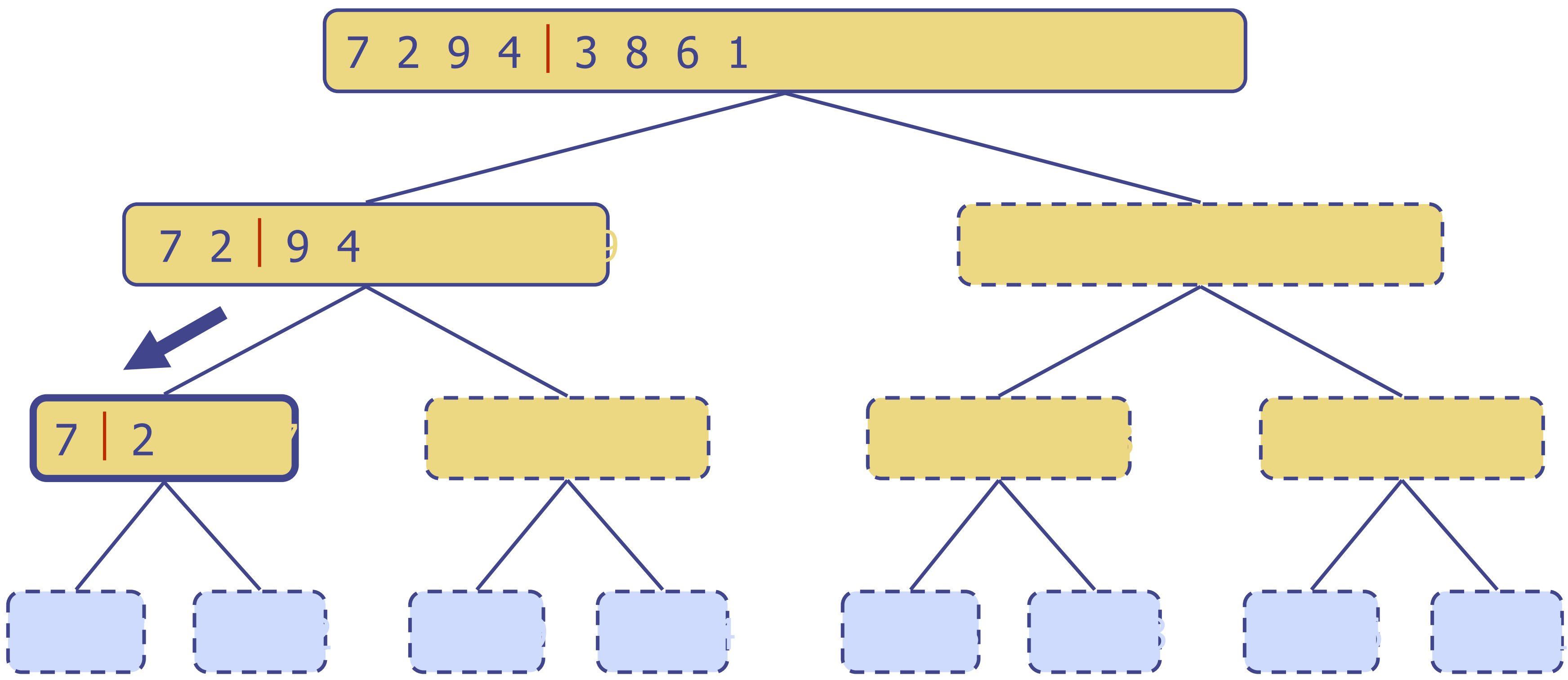


Mergesort

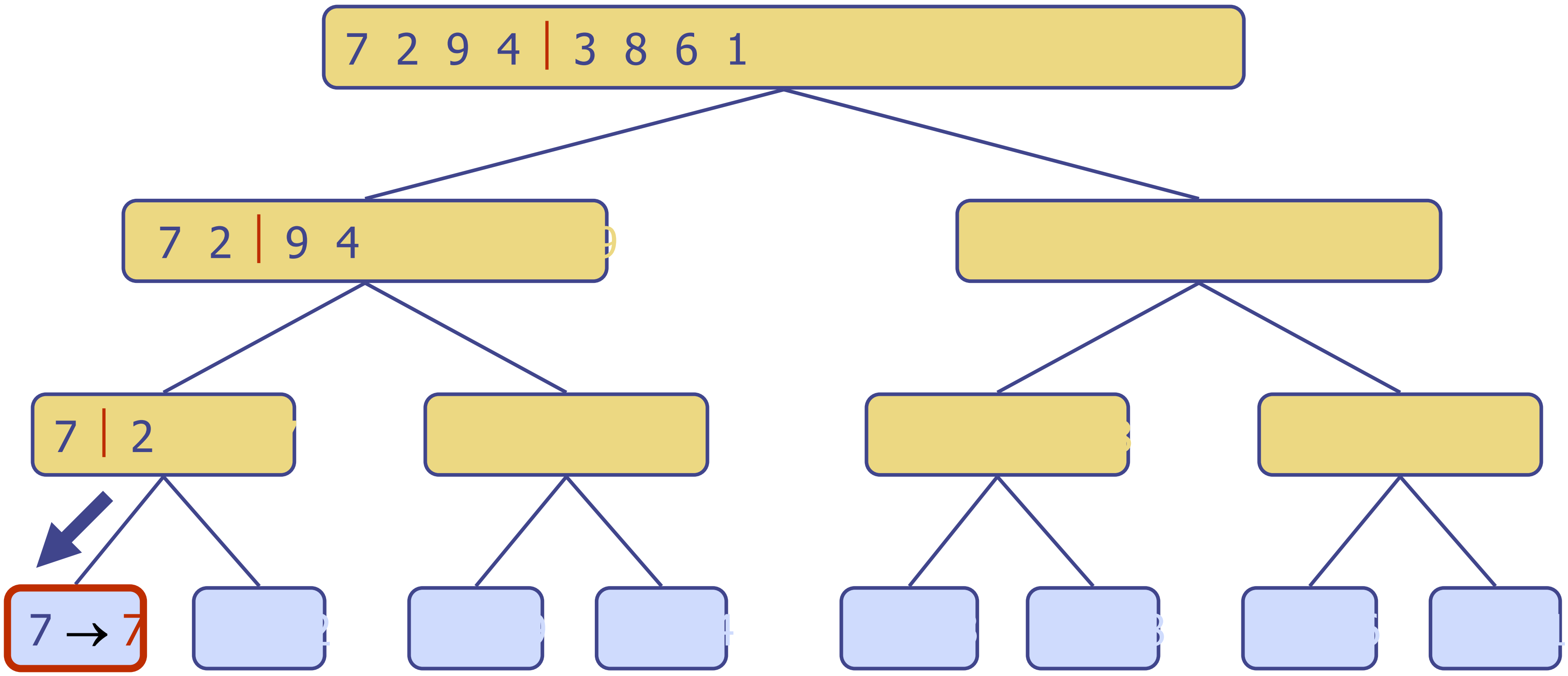
Esempio di esecuzione



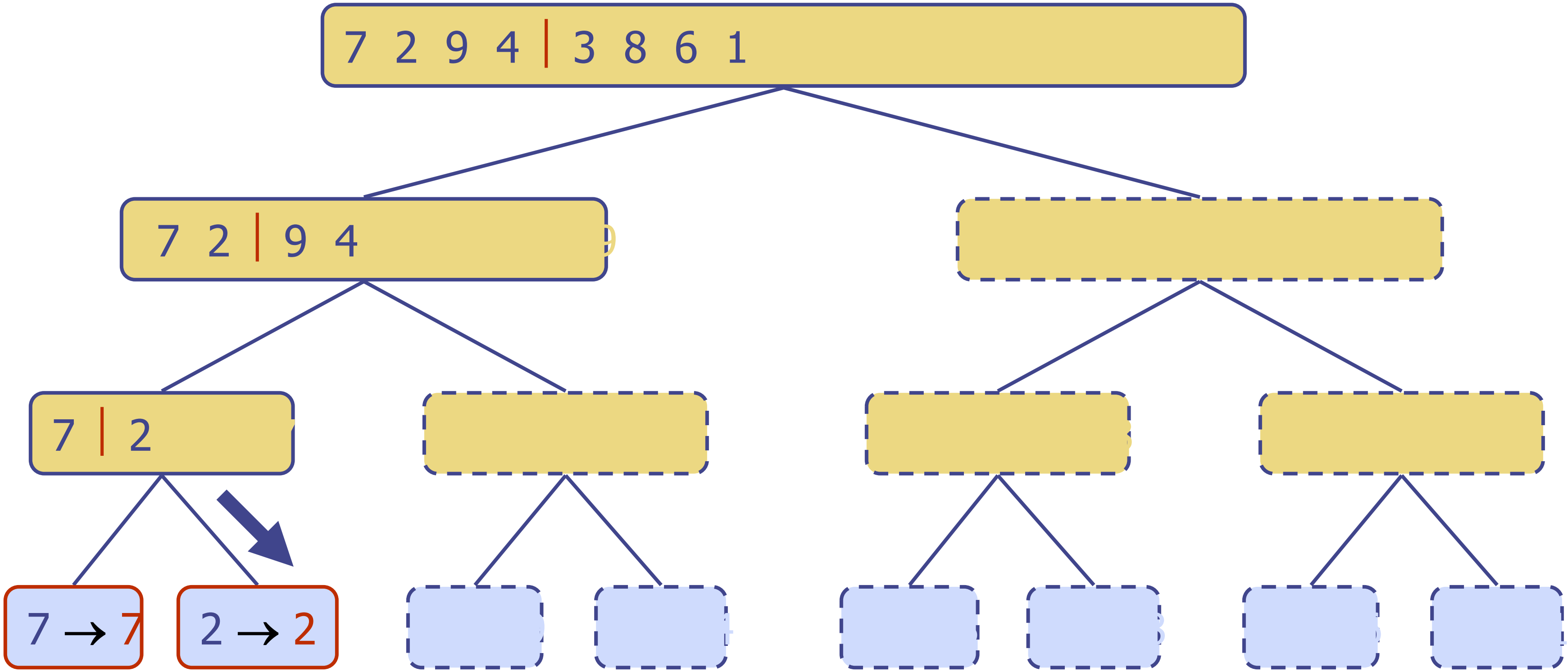
Esempio di esecuzione



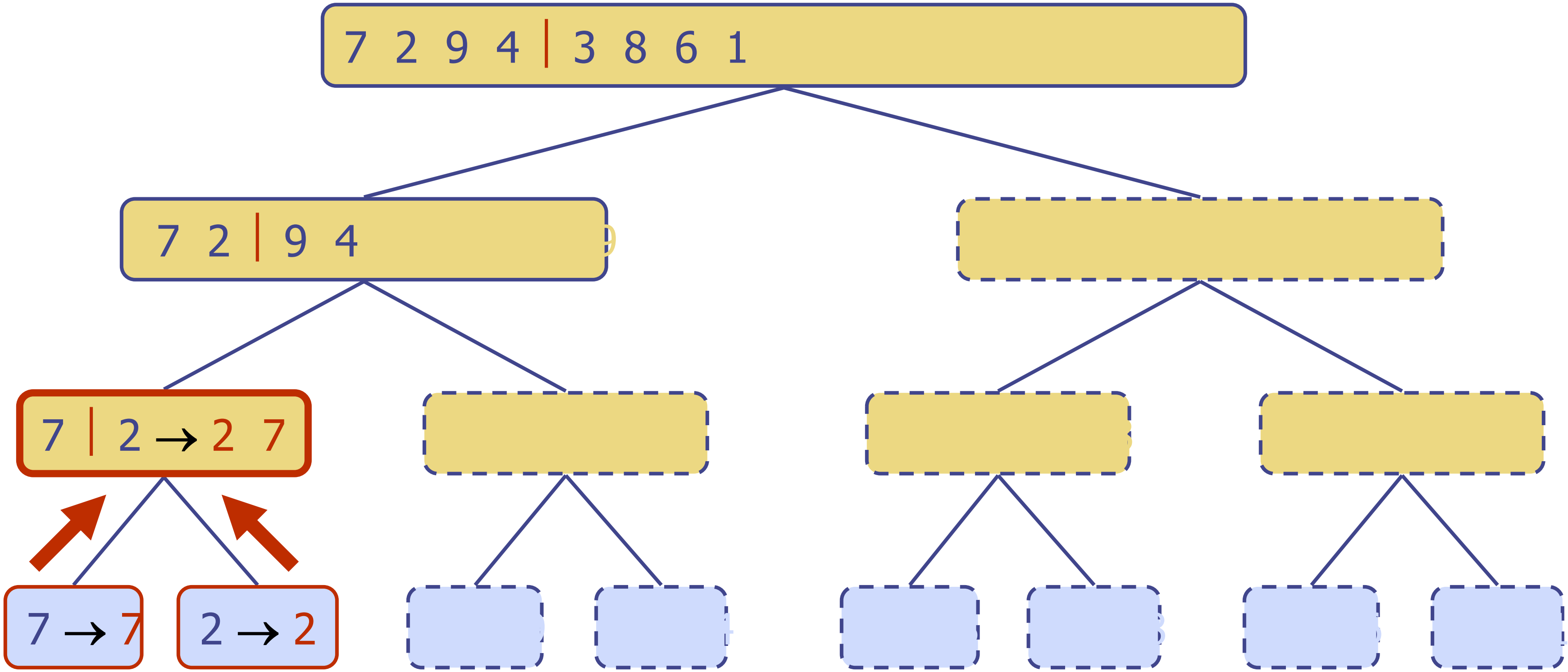
Esempio di esecuzione



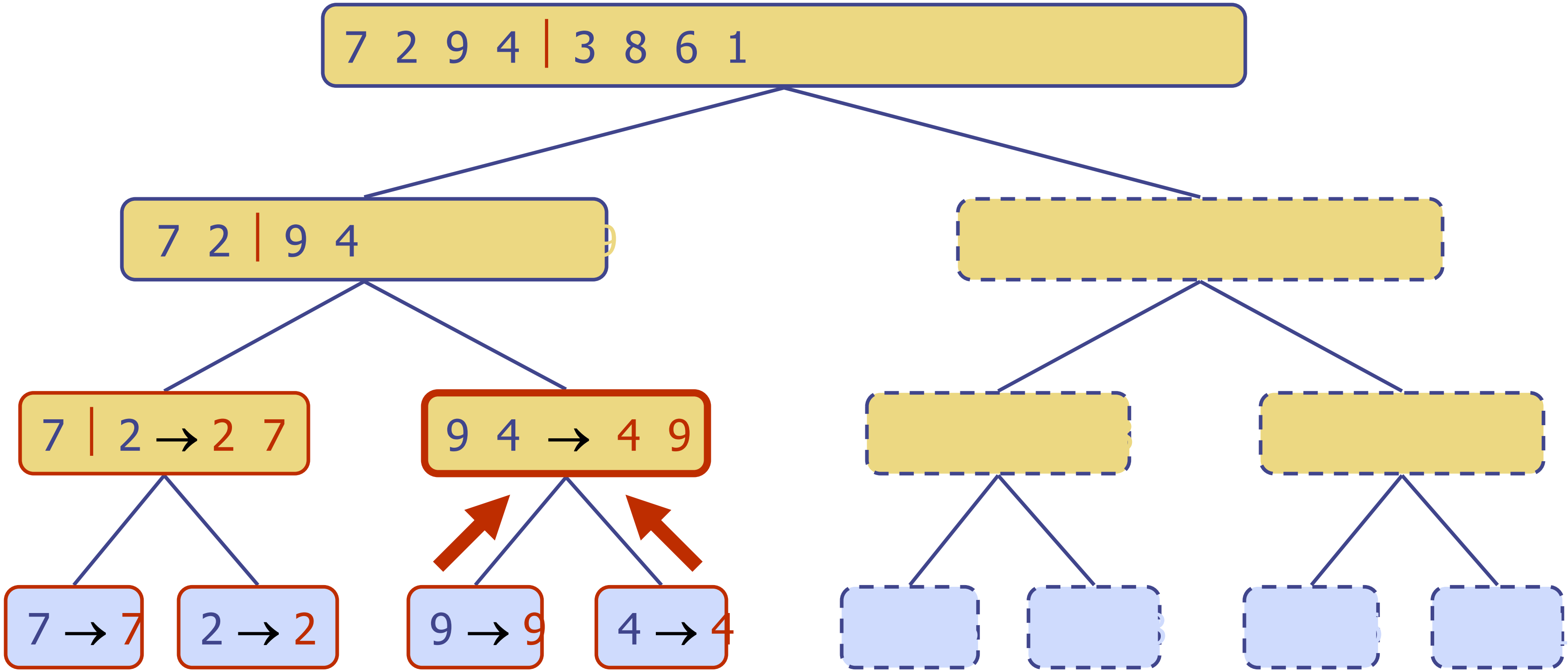
Esempio di esecuzione



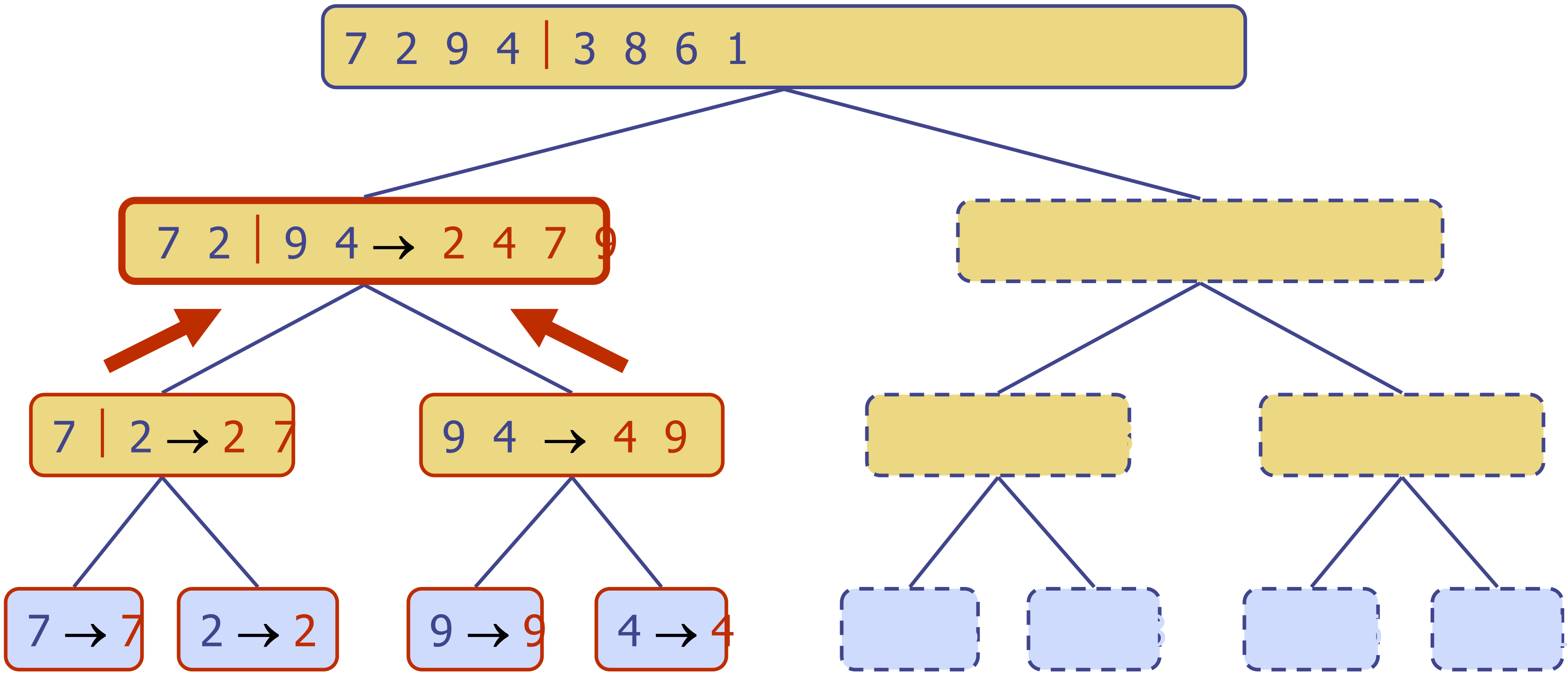
Esempio di esecuzione



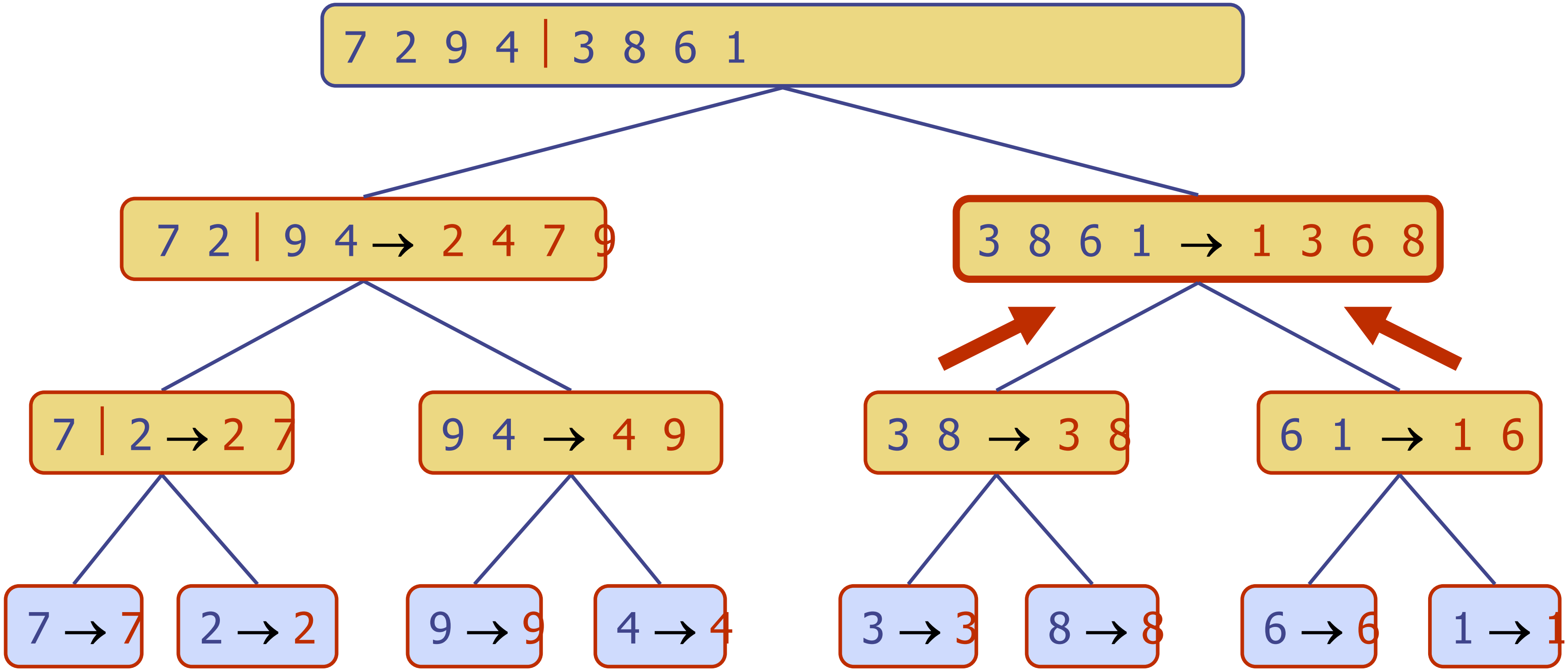
Esempio di esecuzione



Esempio di esecuzione

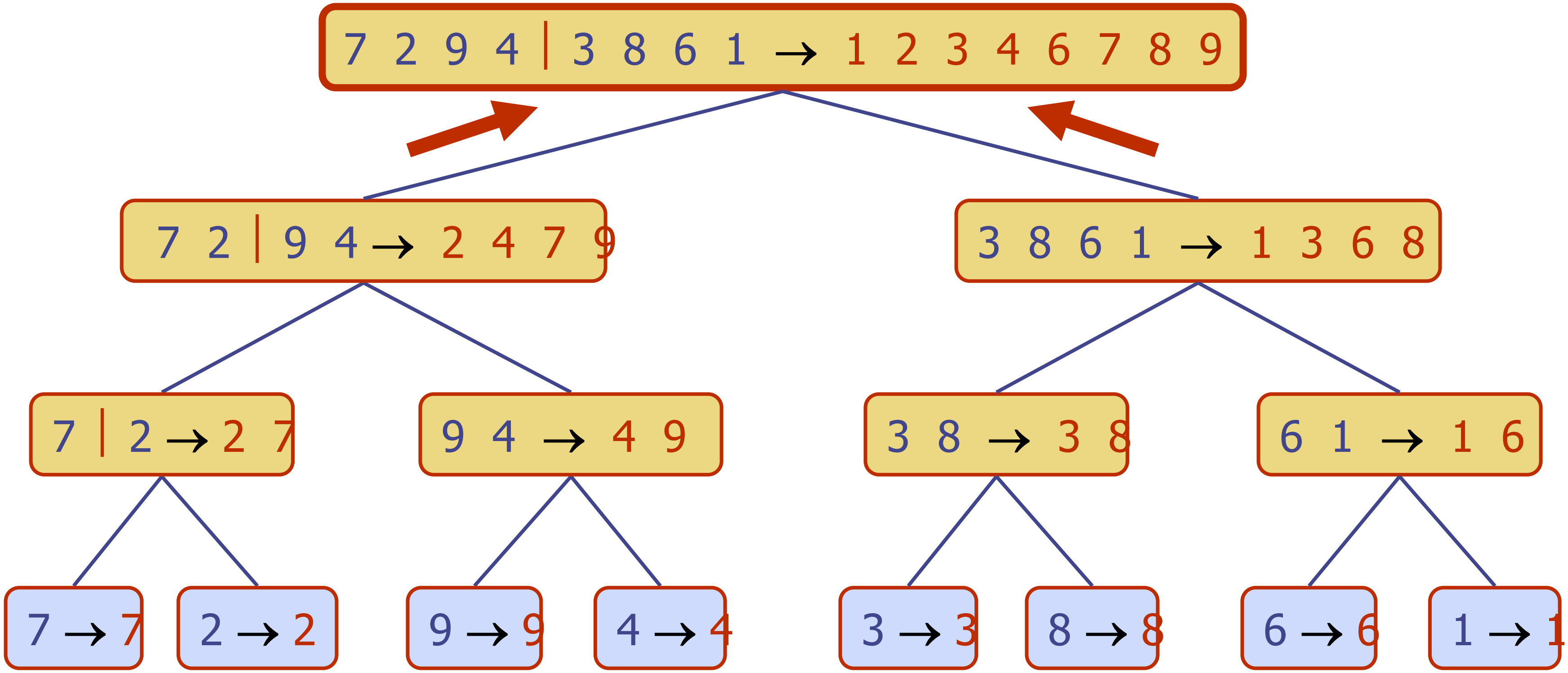


Esempio di esecuzione



Mergesort

Esempio di esecuzione



Algoritmo

```
Algorithm mergeSort(S, C)
        Input sequence S with n
              elements, comparator C
        Output sequence S sorted
        according to C
    if S.size() > 1
        (S1, S2) ← partition(S, n/2)
        mergeSort(S1, C)
        mergeSort(S2, C)
        S ← merge(S1, S2)
```

Approccio divide-et-impera

Algoritmo

```
class MergeSort
{
public:
   MergeSort( int ); // constructor initializes
vector
   void sort(); // sort vector using merge sort
   void displayElements() const; // display vector
elements
private:
   int size; // vector size
   vector< int > data; // vector of ints
   void sortSubVector( int, int ); // sort subvector
   void merge( int, int, int, int ); // merge two
sorted vectors
   void displaySubVector( int, int ) const; //
display subvector
};
```

Mergesort

Algoritmo

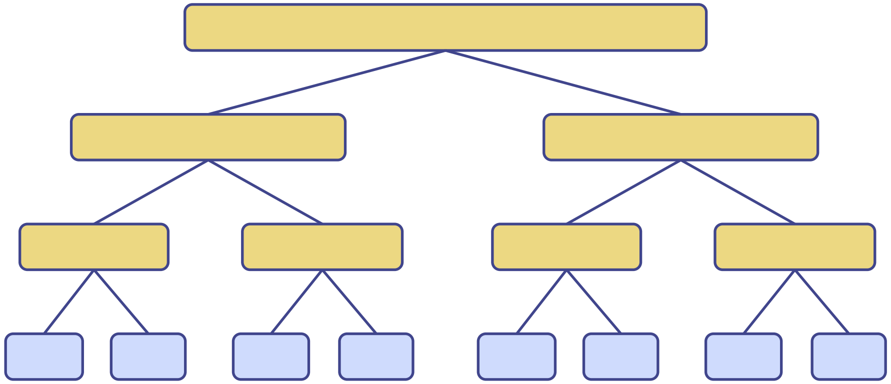
```
Algorithm merge(A, B)
        Input sequences A and B with
                 n/2 elements each
        Output sorted sequence of A ∪ B

    S ← empty sequence
    while ¬A.empty()  ∧ ¬B.empty()
        if A.front() < B.front()
                S.addBack(A.front()); A.eraseFront();
        else
                 S.addBack(B.front()); B.eraseFront();
    while ¬A.empty()
        S.addBack(A.front()); A.eraseFront();
    while ¬B.empty()
        S.addBack(B.front()); B.eraseFront();
    return S
```

Approccio divide-et-impera

Analisi dell'algoritmo

depth	#seqs	size
0	1	n
1	2	$n/2$
i	2^i	$n/2^i$
…	…	…



Complessità

Analisi dell'algoritmo

Ricorrenza

$$t(n) = \begin{cases} b & \text{if } n \leq 1 \\ 2t(n/2) + cn & \text{otherwise.} \end{cases}$$

$$\begin{aligned} t(n) &= 2(2t(n/2^2) + (cn/2)) + cn \\ &= 2^2 t(n/2^2) + 2(cn/2) + cn = 2^2 t(n/2^2) + 2cn. \end{aligned}$$

$$t(n) = 2^i t(n/2^i) + icn.$$

$n/2^i = 1$, i.e., $i = \log n$

$$\begin{aligned} t(n) &= 2^{\log n} t(n/2^{\log n}) + (\log n)cn \\ &= nt(1) + cn \log n \\ &= nb + cn \log n. \end{aligned}$$