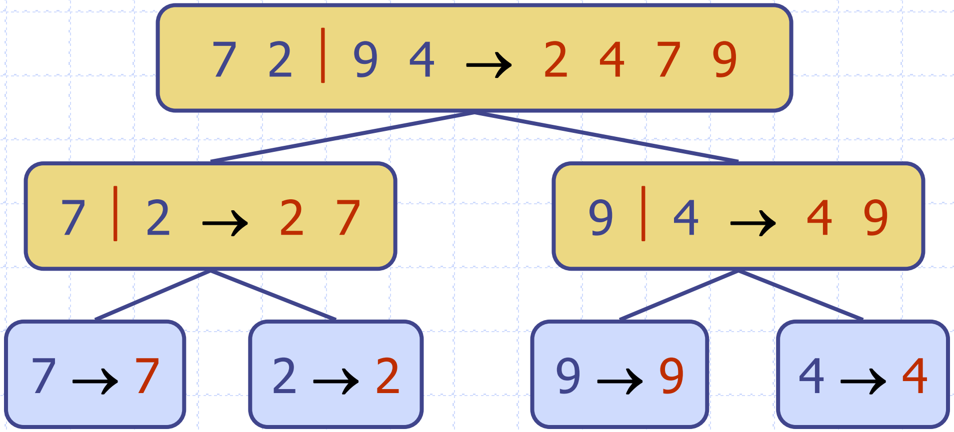
Merge Sort



Intorduction to Merge sort

- On merge sort we apply Divide and Conquer techniques in following steps
- Divide-and conquer is a general algorithm design paradigm:
  - Divide : Given a sequence of n elements (a[1],a[2],..,a[n])
  - Split into two sets a[1],..a[n/2] and a[n/2+1],a..[n]
  - Conquer: Each set is individually sorted
  - Conquer: Resulting sorted sequence are merged to produce a single sorted sequence of n elements.
- Merge-sort is a sorting algorithm based on the divide-and-conquer paradigm
- Like heap-sort
  - It uses a comparator
  - It has $O(n \log n)$ running time
- Unlike heap-sort
  - It does not use an auxiliary priority queue
  - It accesses data in a sequential manner (suitable to sort data on a disk)

Merge-Sort

- If the time for merging operantion is propotional to n,then the computing time for merge sort is described by the recurrence relation
- T(n)= a n=1
  2T(n/2)+cn n>1

When n is a power of 2, n=2k, we can solve this equation by recursive method(succesive substitution or iterative method)

T(n)=2(2T(n/4)+cn/2)+cn

=4T(n/4)+2cn

=4(2T(n/8)=cn/4)+2cn

.

.

.

=2kT(1)+kcn

=an+cnlogn

T(n)=O(nlogn)

Algorithm *mergeSort(low,high)*

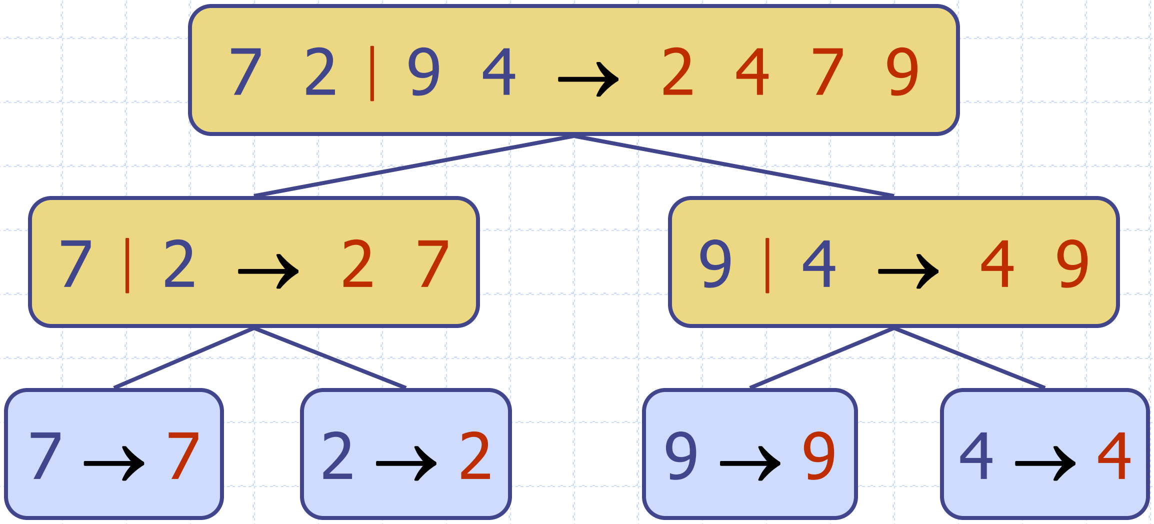
```
{
    if(low<high) then
        mid=(low+high)/2
        mergeSort(low,mid)
        mergeSort(mid+1,high);
        Merge(low,mid,high)
```

Merging Two Sorted Sequences

```
Algorithm merge(low,mid,high)
{
    h:=low:i:=high:j:=mid+1
    while((h<=mid) and(j<=high))do
    {
        if(a[h]<=a[j])
        {b[i]:=a[h];h:=h+1;          }
        else
        {b[i]:=a[j]:j:=j+1;      }
        i:=i+1;
    }
    if(h>mid)
        for k:=j to high do
            {b[i]:=a[k];i:=i+1;
            }
    else
        for k:=h to mid do
            {b[i]:=a[k]:i:=i+1;      }
    for k:=low to high do a[k]:=b[k];
}
```

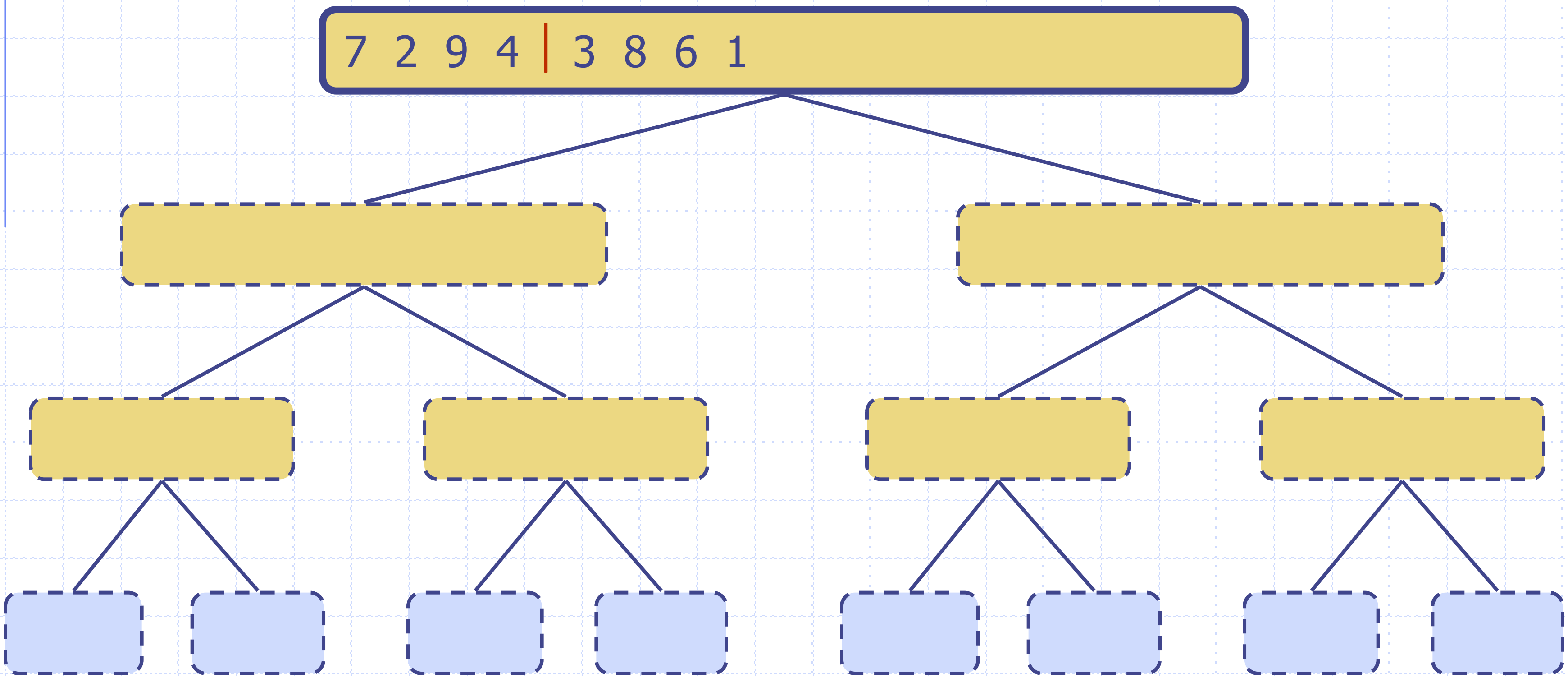
Merge-Sort Tree

- An execution of merge-sort is depicted by a binary tree
  - each node represents a recursive call of merge-sort and stores
    - unsorted sequence before the execution and its partition
    - sorted sequence at the end of the execution
  - the root is the initial call
  - the leaves are calls on subsequences of size 0 or 1



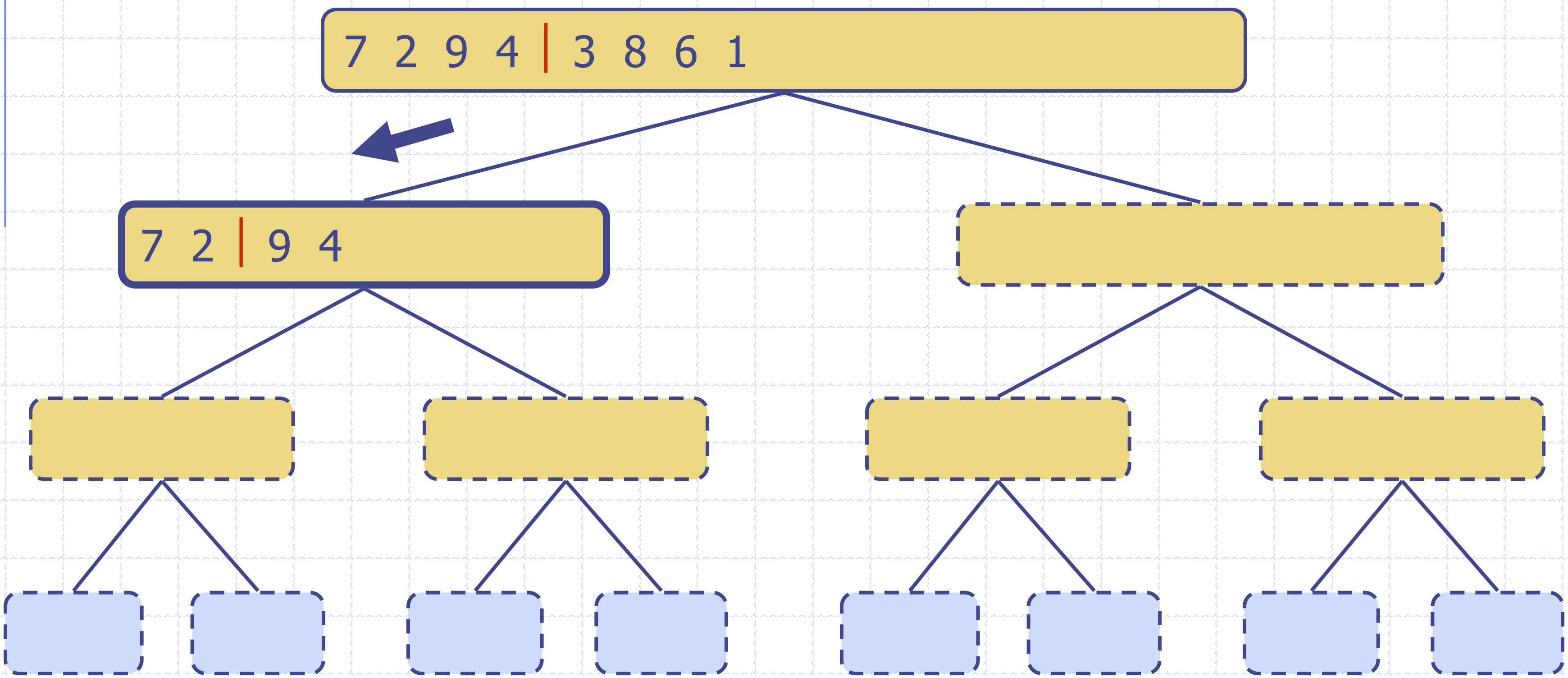
Execution Example

- Partition



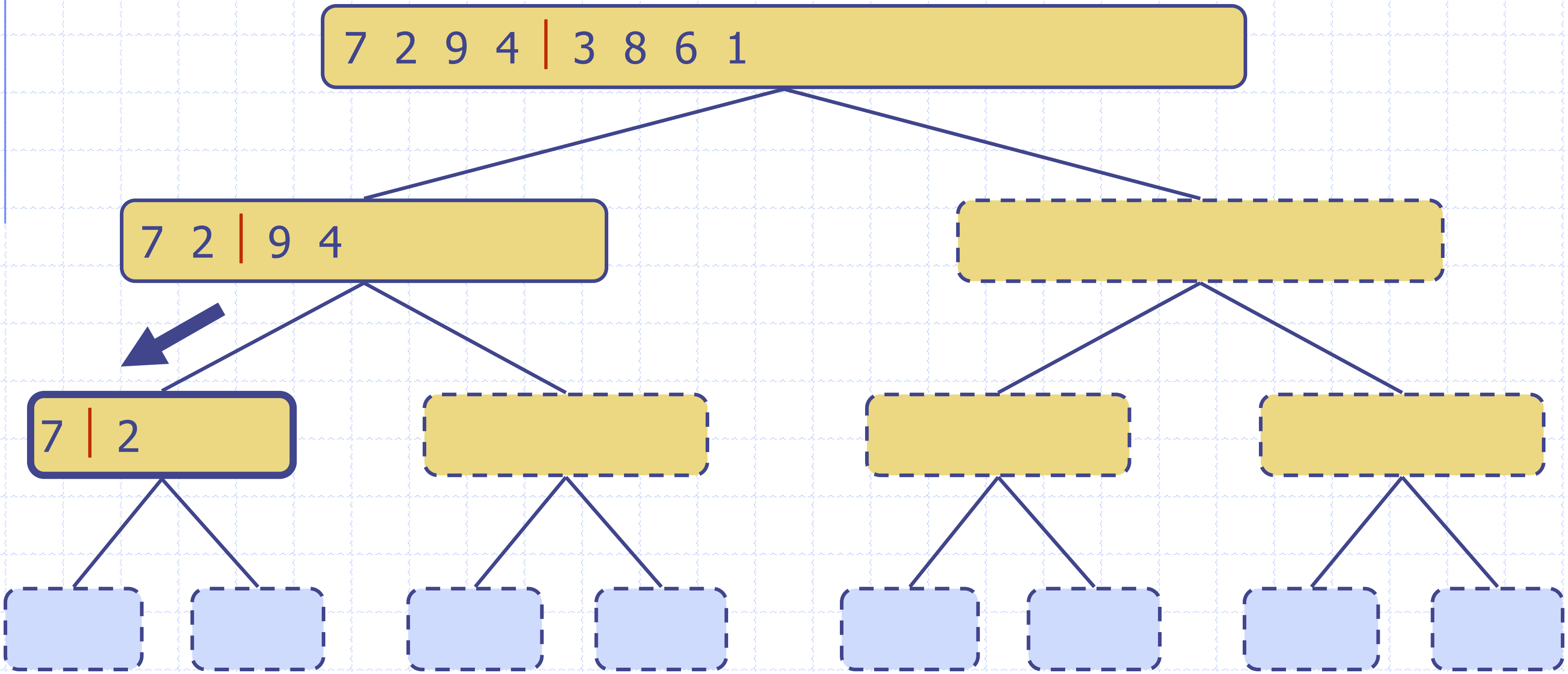
Execution Example (cont.)

- Recursive call, partition



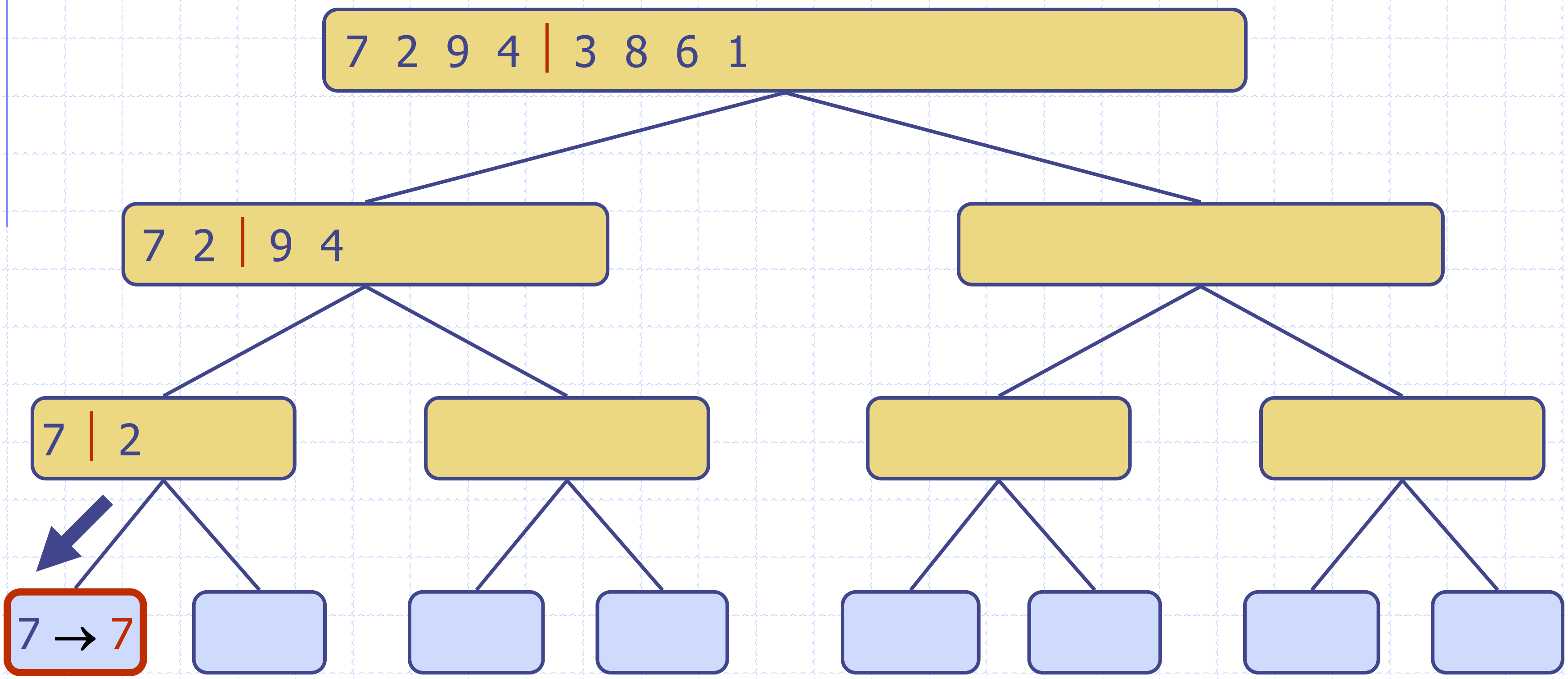
Execution Example (cont.)

- Recursive call, partition



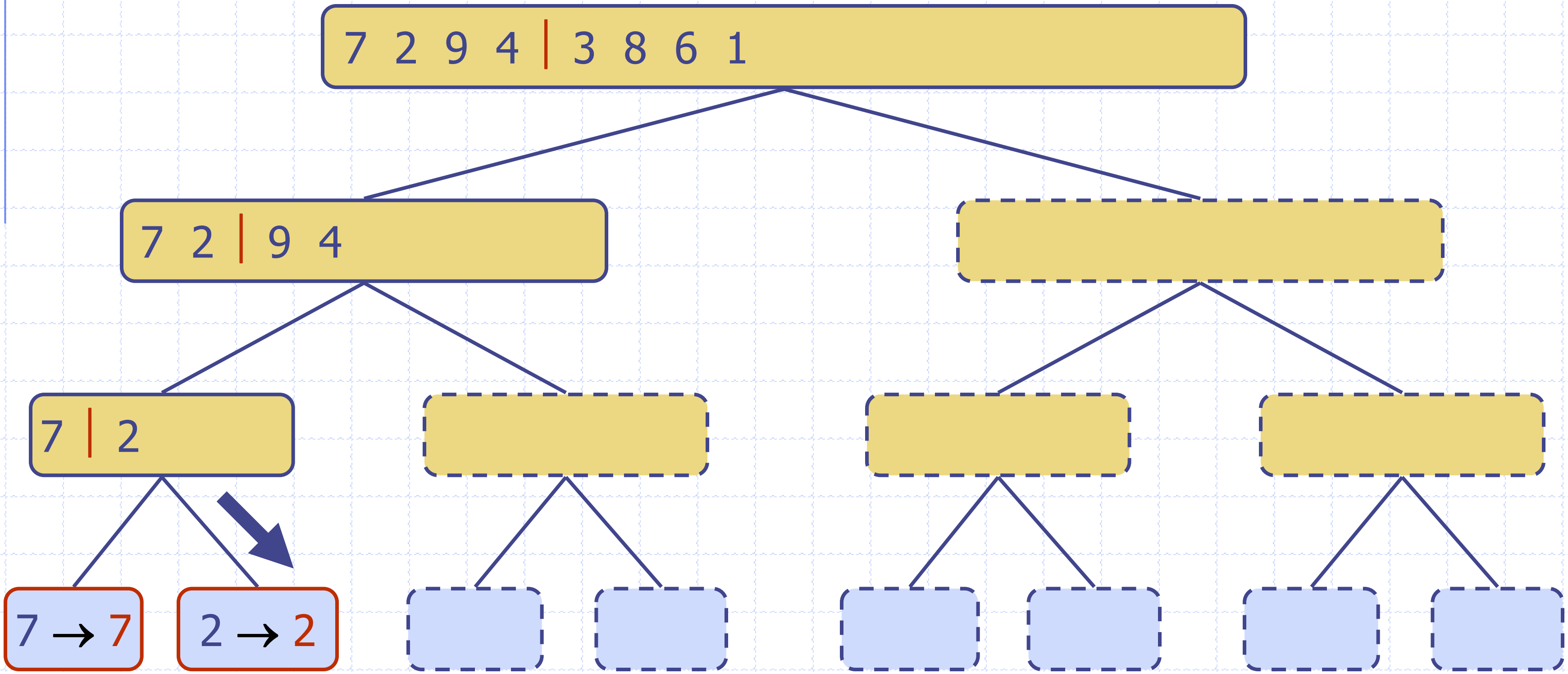
Execution Example (cont.)

- Recursive call, base case



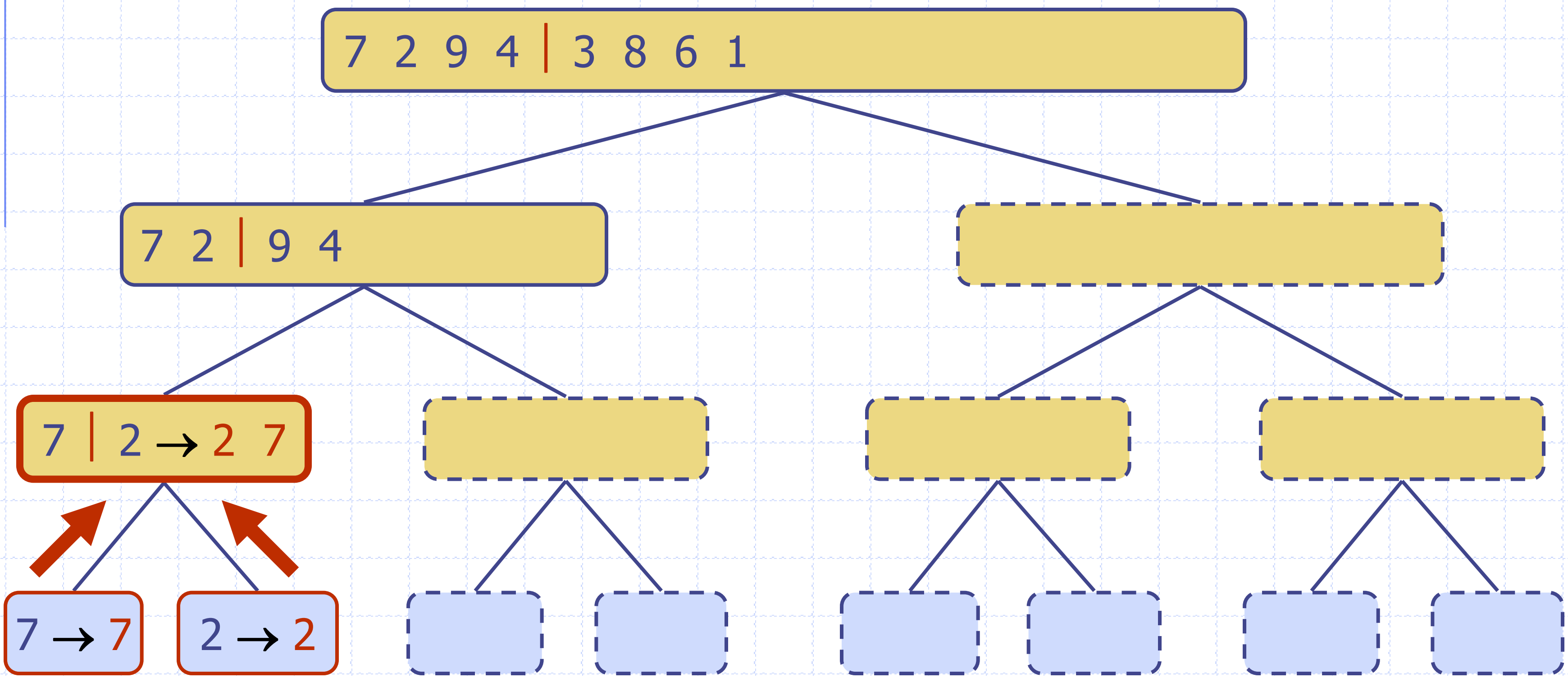
Execution Example (cont.)

- Recursive call, base case



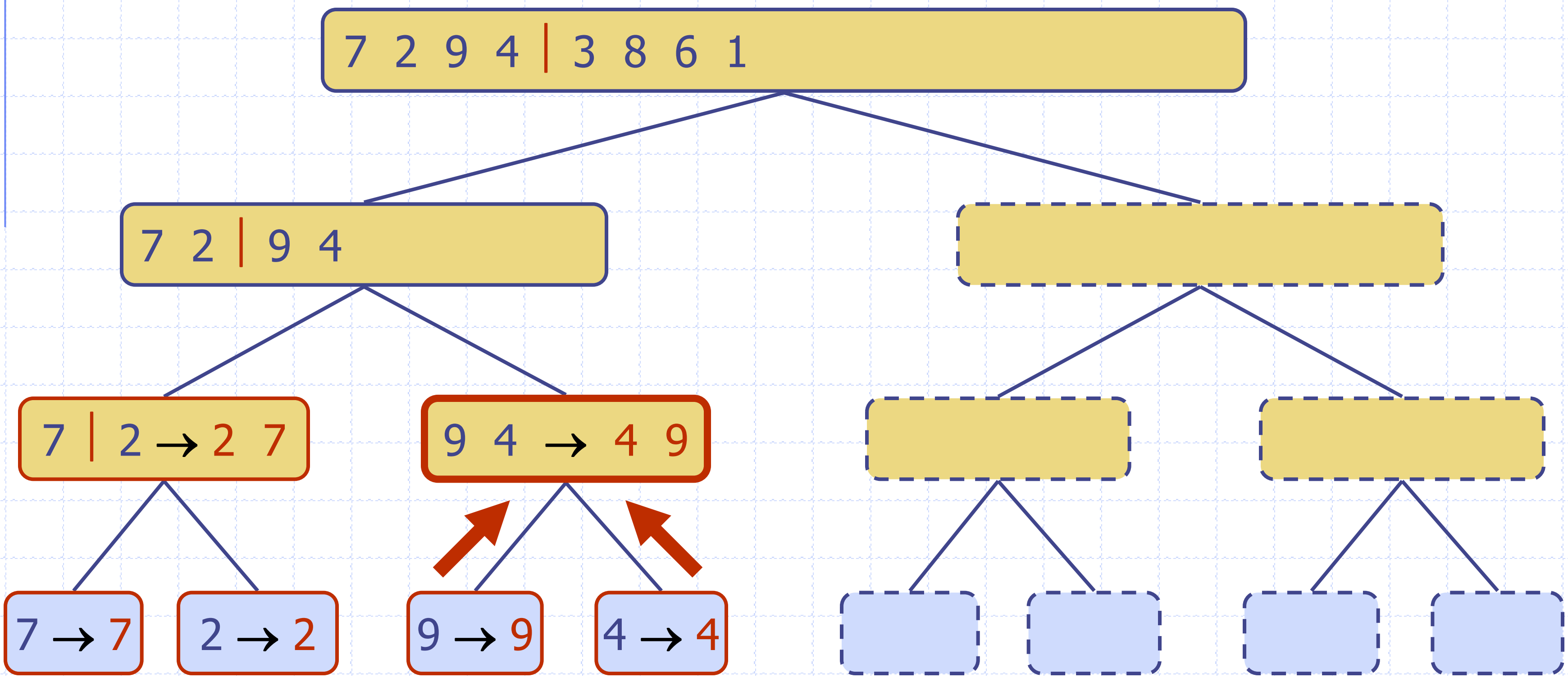
Execution Example (cont.)

- Merge



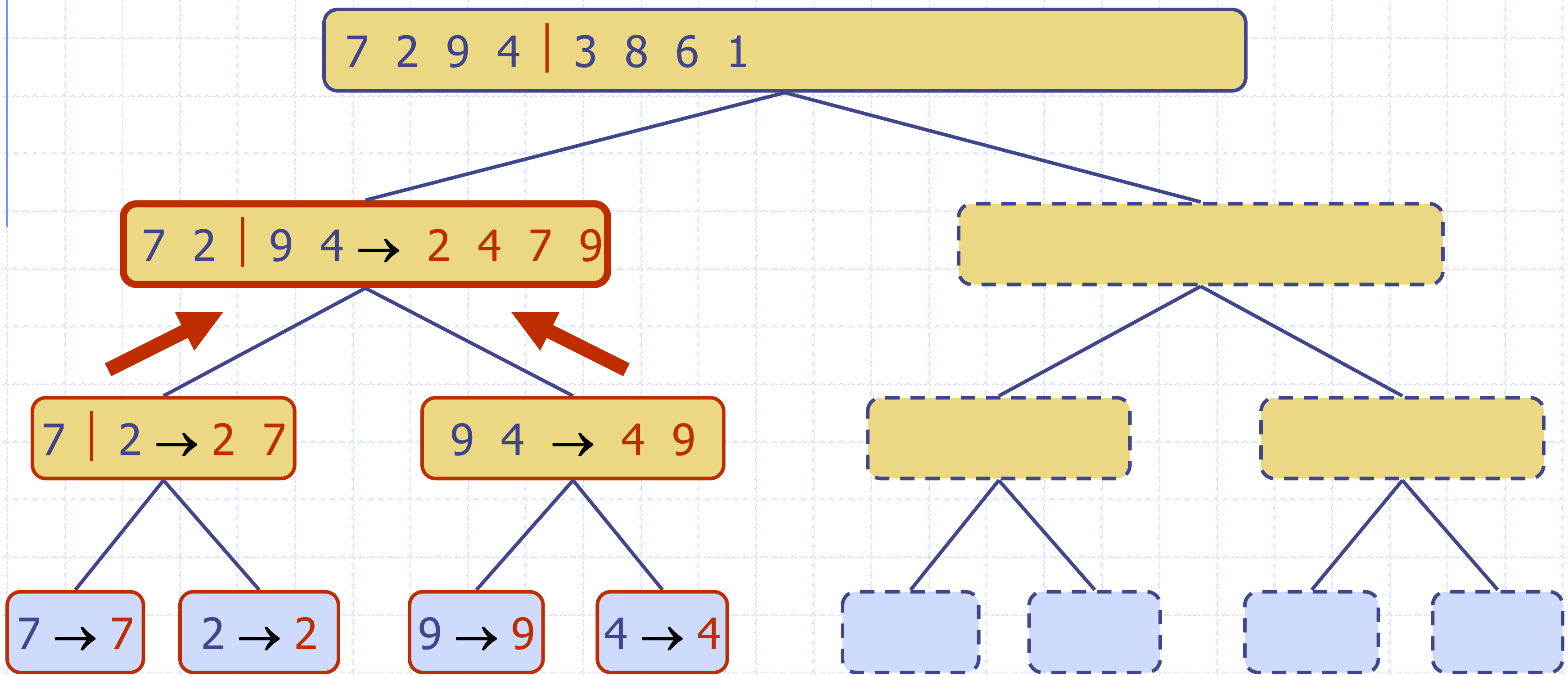
Execution Example (cont.)

- Recursive call, …, base case, merge



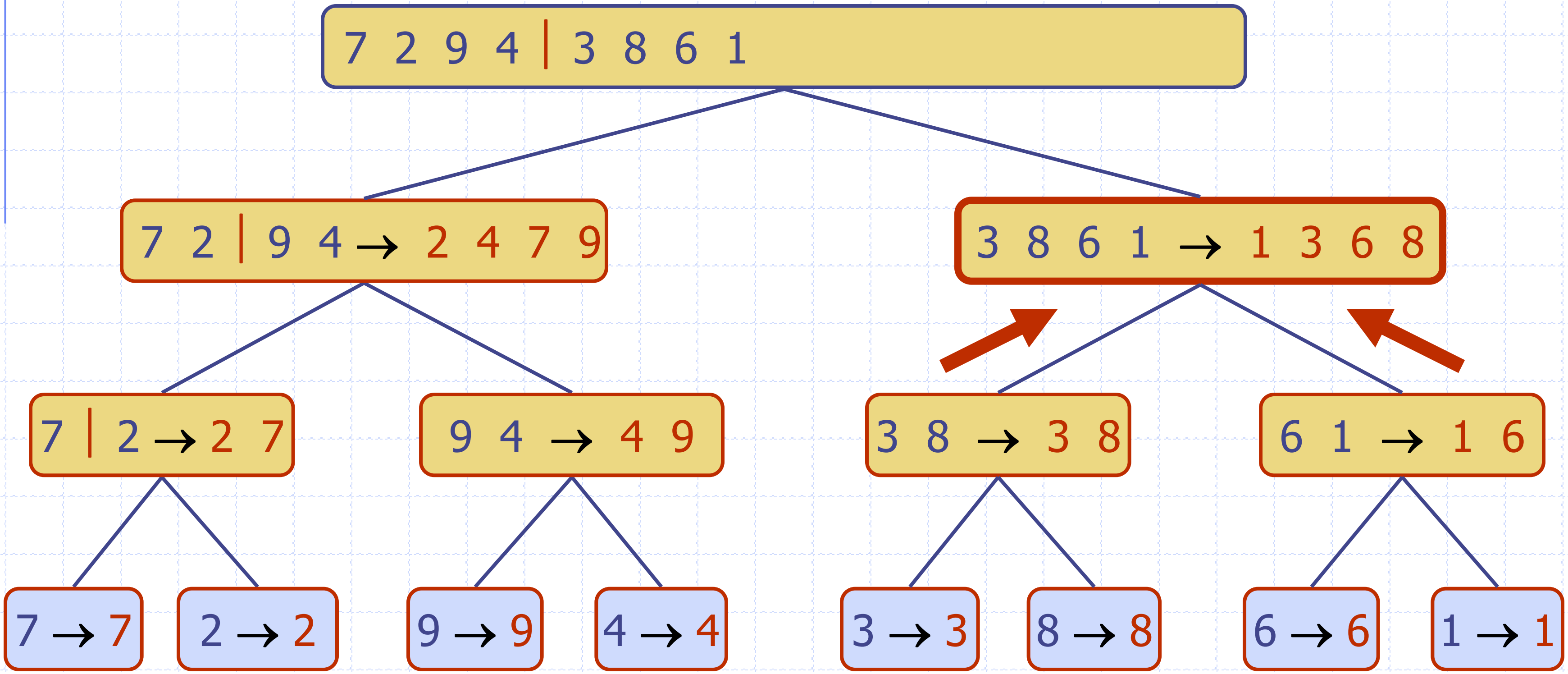
Execution Example (cont.)

- Merge



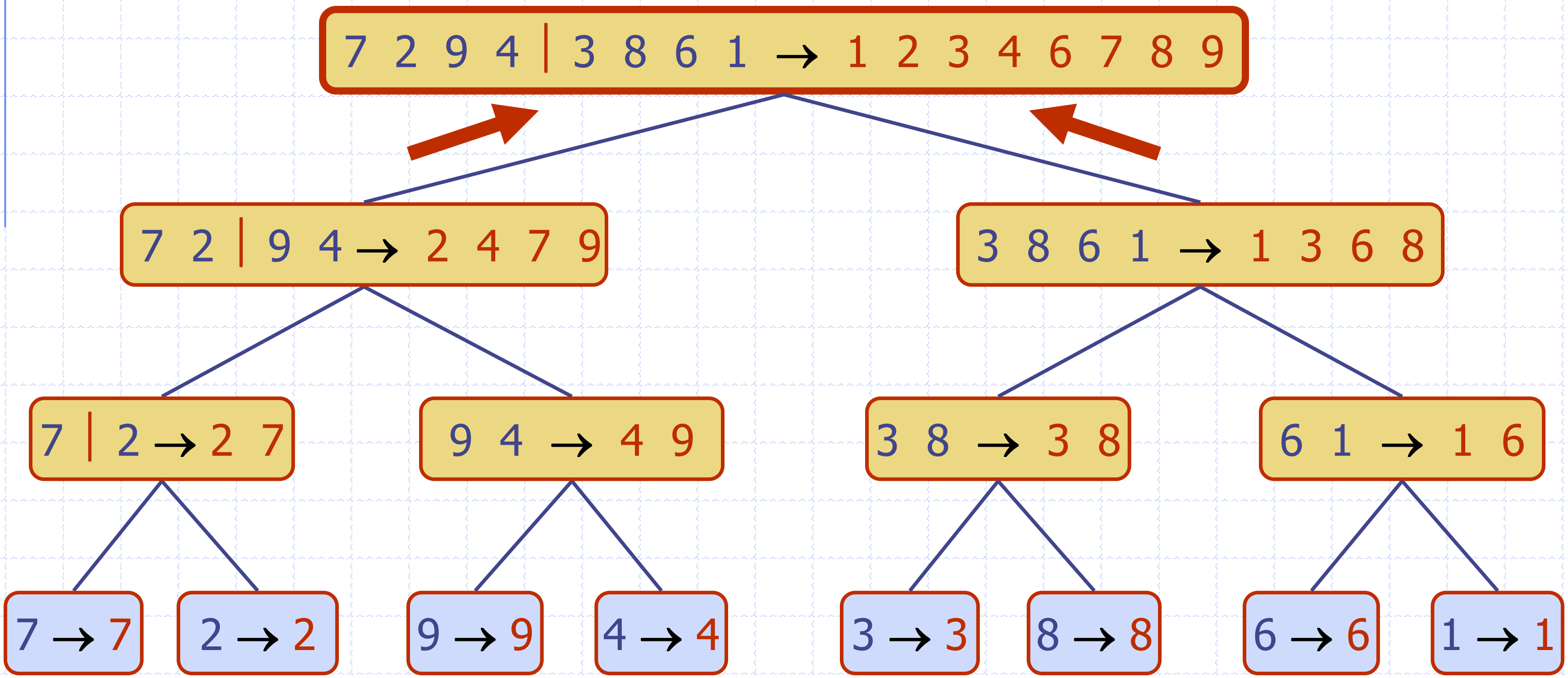
Execution Example (cont.)

- Recursive call, ..., merge, merge



Execution Example (cont.)

- Merge



Summary of Sorting Algorithms

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)

Application of Merge Sort

- Tape drive
- Disk drive
- Online sorting

Scope of Merge Sort

- Parallel processing
- Optimizing merge sort

Assignment

- Q.1)Prove that efficiency of merge sort is O(nlogn).
- Q.2)Explain merge sort with example.
- Q.3)Compare merge sort with Quick sort & Heap sort.