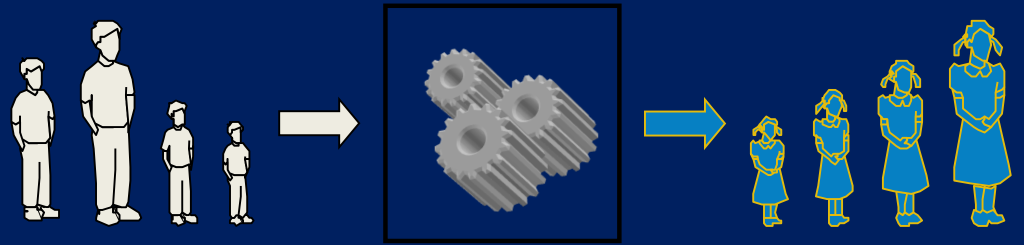
Chapter 5: Divide and Conquer

Merge Sort



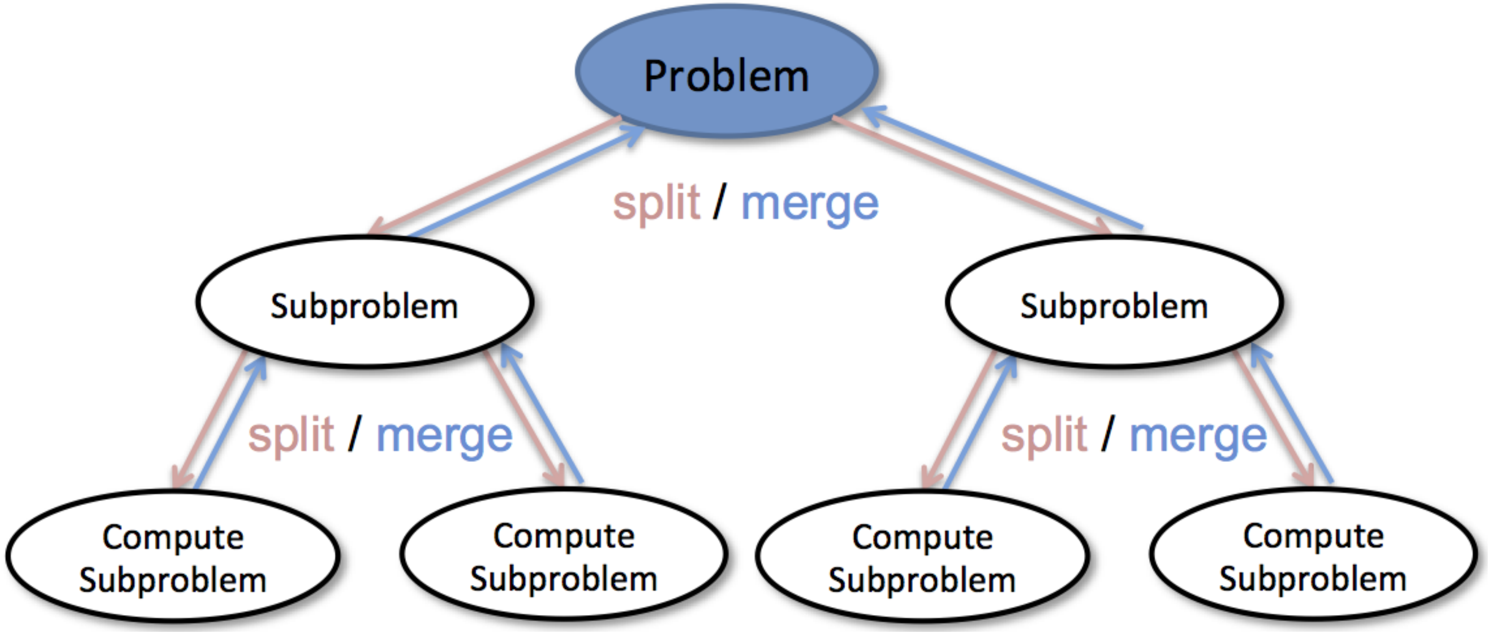
Input Algorithm Output

Prof. Anand Singh Jalal

Department of Computer Engineering & Applications

Divide and Conquer

Divide the problem into a number of sub-problems that are smaller instances of the same problem.
Conquer the sub-problems by solving them recursively.
Combine the solutions to the sub-problems into the solution for the original problem.



Divide and Conquer

- Sorting Algorithms
  - Bubble sort
  - Selection sort
  - Insertion Sort
  - **Quick Sort**
  - **Merge Sort**

Divide-Conquer Approach

Divide and Conquer: Merge Sort

- **DIVIDE :** Divide the unsorted list into two sub lists of about half the size.
- **CONQUER :** Sort each of the two sub lists recursively. If they are small enough just solve them in a straight forward manner.
- **COMBINE :** Merge the two-sorted sub lists back into one sorted list.

Divide and Conquer: Merge Sort ...

- MergeSort is a recursive sorting procedure that uses at most **O(n lg(n))** comparisons.
- To sort an array of **n** elements, we perform the following steps in sequence:
  - If **n < 2** then the array is already sorted.
  - Otherwise, **n > 1**, and we perform the following three steps in sequence:
    1. **Sort** the **left half** of the the array using MergeSort.
    2. **Sort** the **right half** of the the array using MergeSort.
    3. **Merge** the sorted left and right halves.

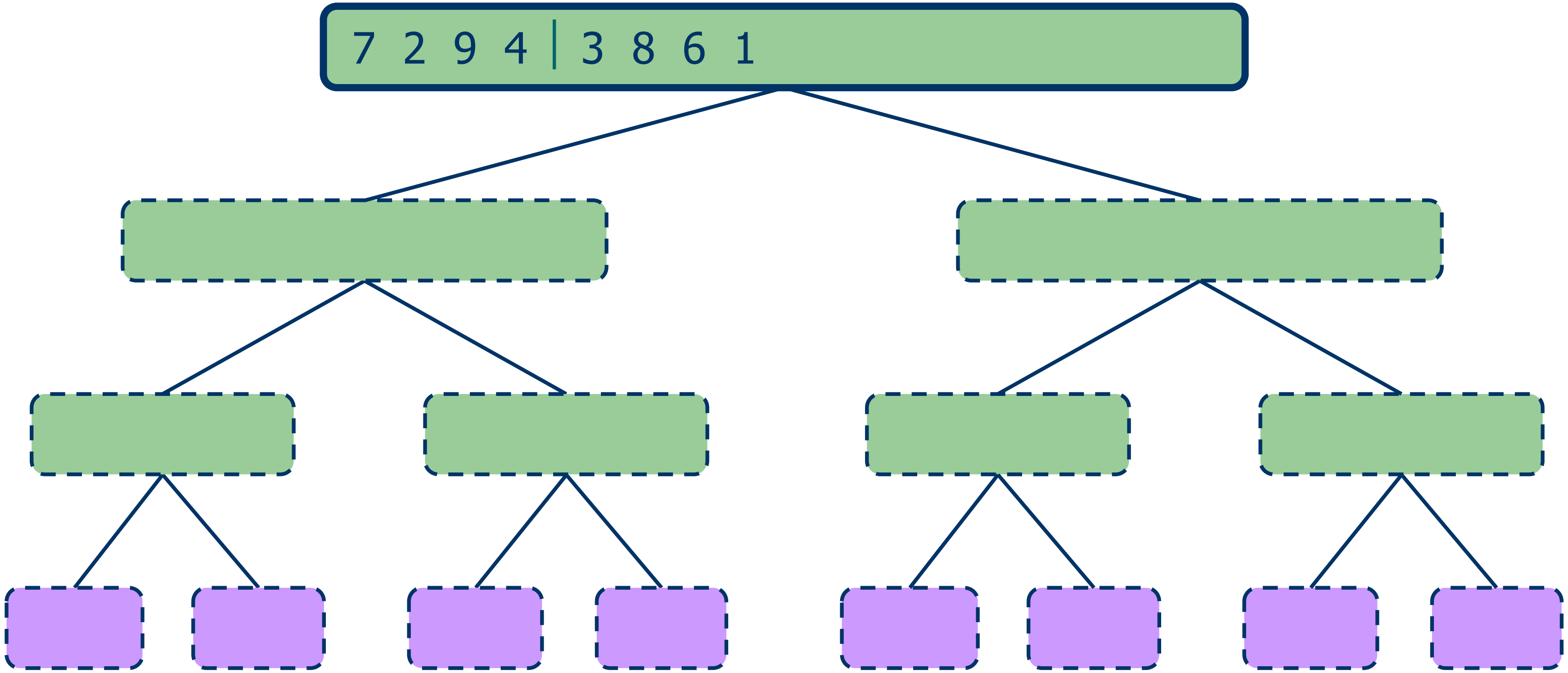
Divide and Conquer: Merge Sort …

8	2	9	4	5	3	1	6

- Divide it in two at the midpoint
- Conquer each side in turn (by recursively sorting)
- Merge two halves together

Execution Example

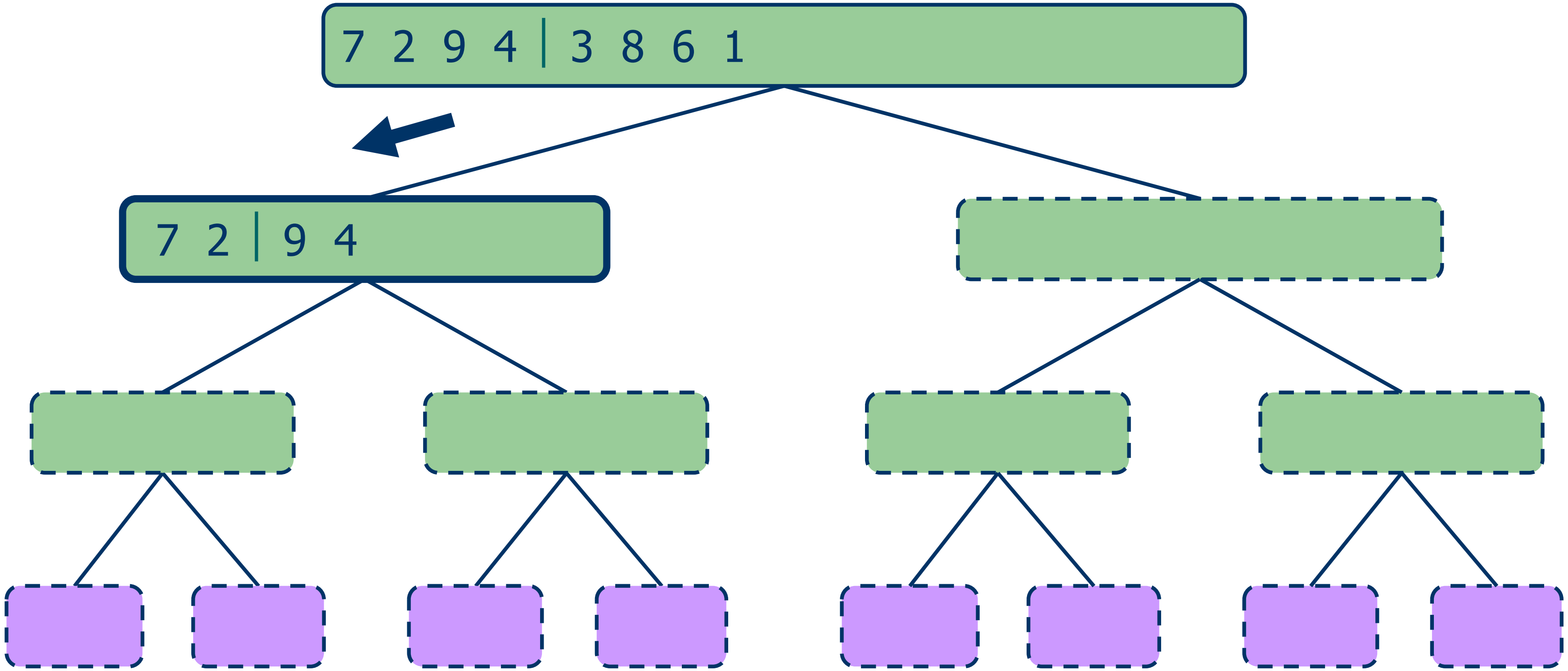
- Partition



Divide and Conquer: Merge Sort ...

Execution Example (cont.)

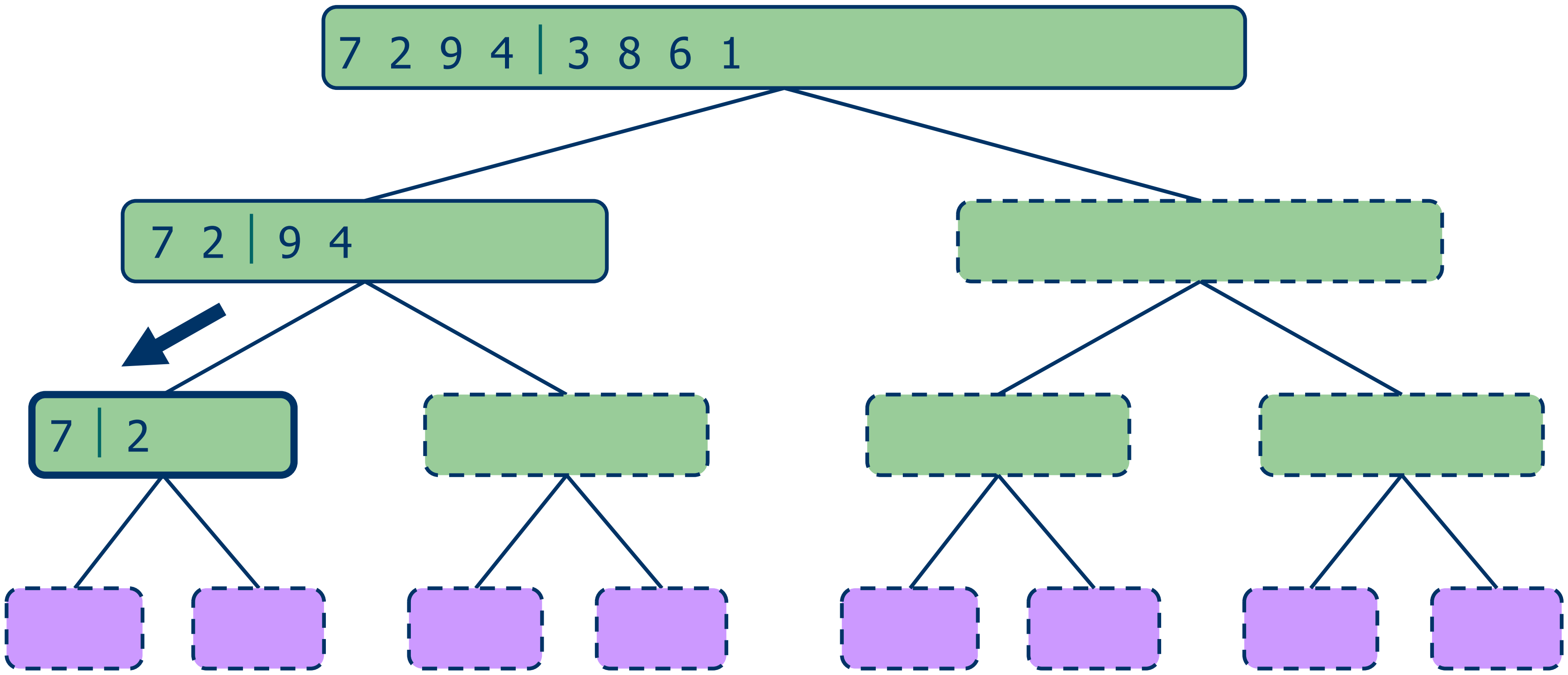
- Recursive call, partition



Divide and Conquer: Merge Sort …

Execution Example (cont.)

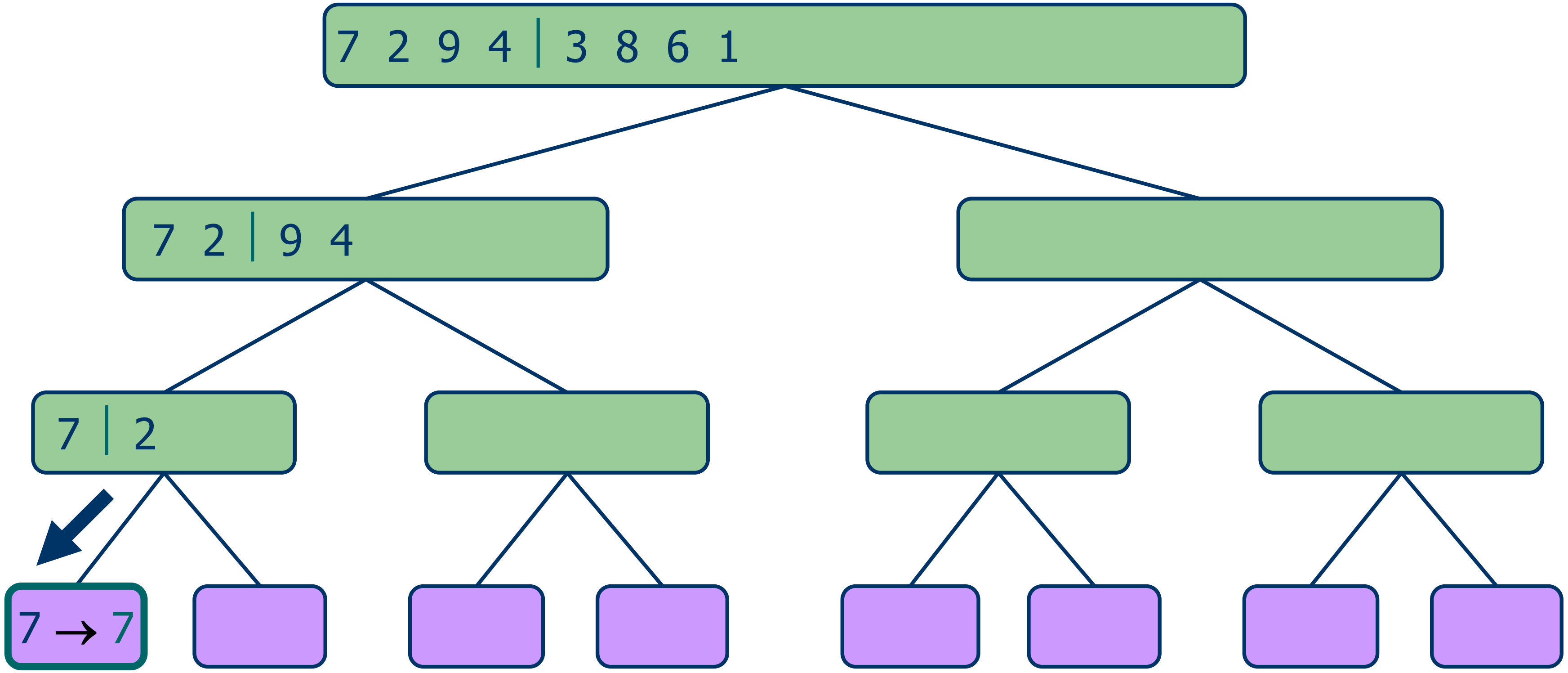
- Recursive call, partition



Divide and Conquer: Merge Sort ...

Execution Example (cont.)

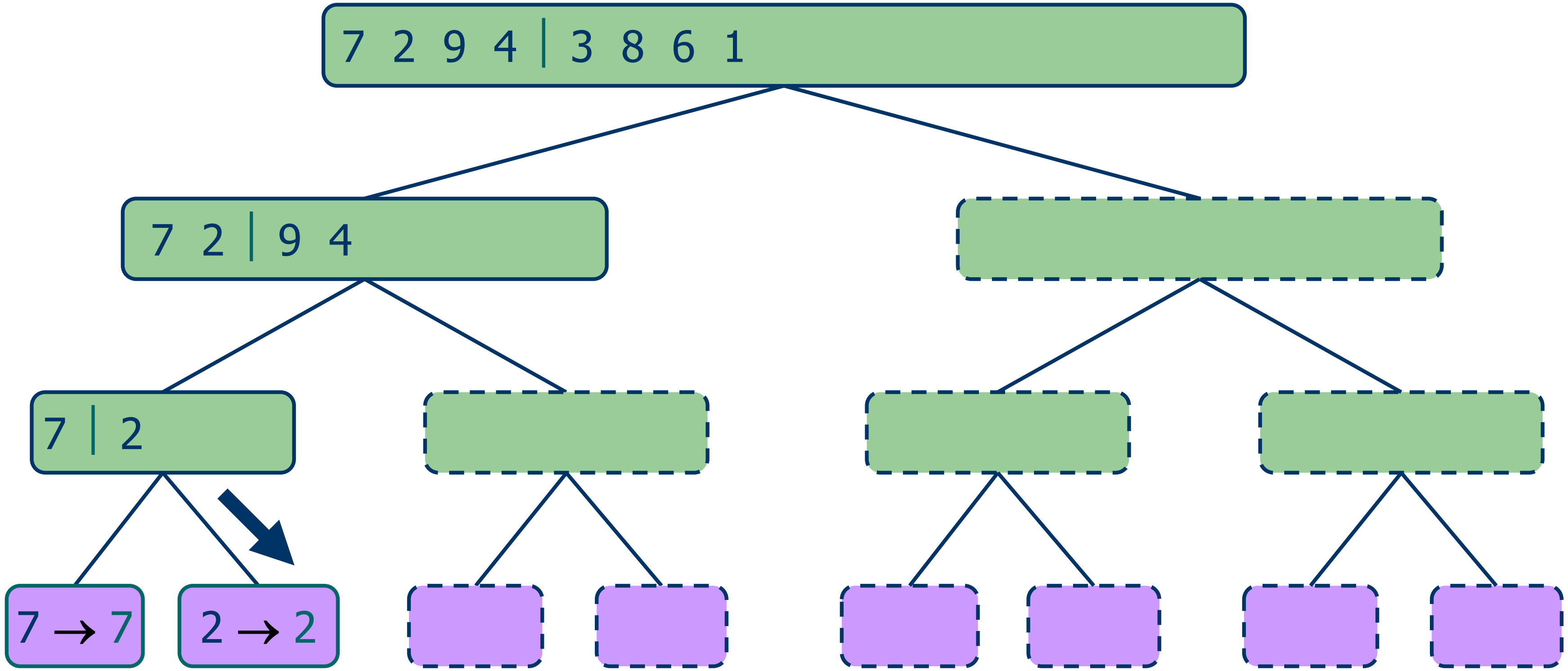
- Recursive call, base case



Divide and Conquer: Merge Sort ...

Execution Example (cont.)

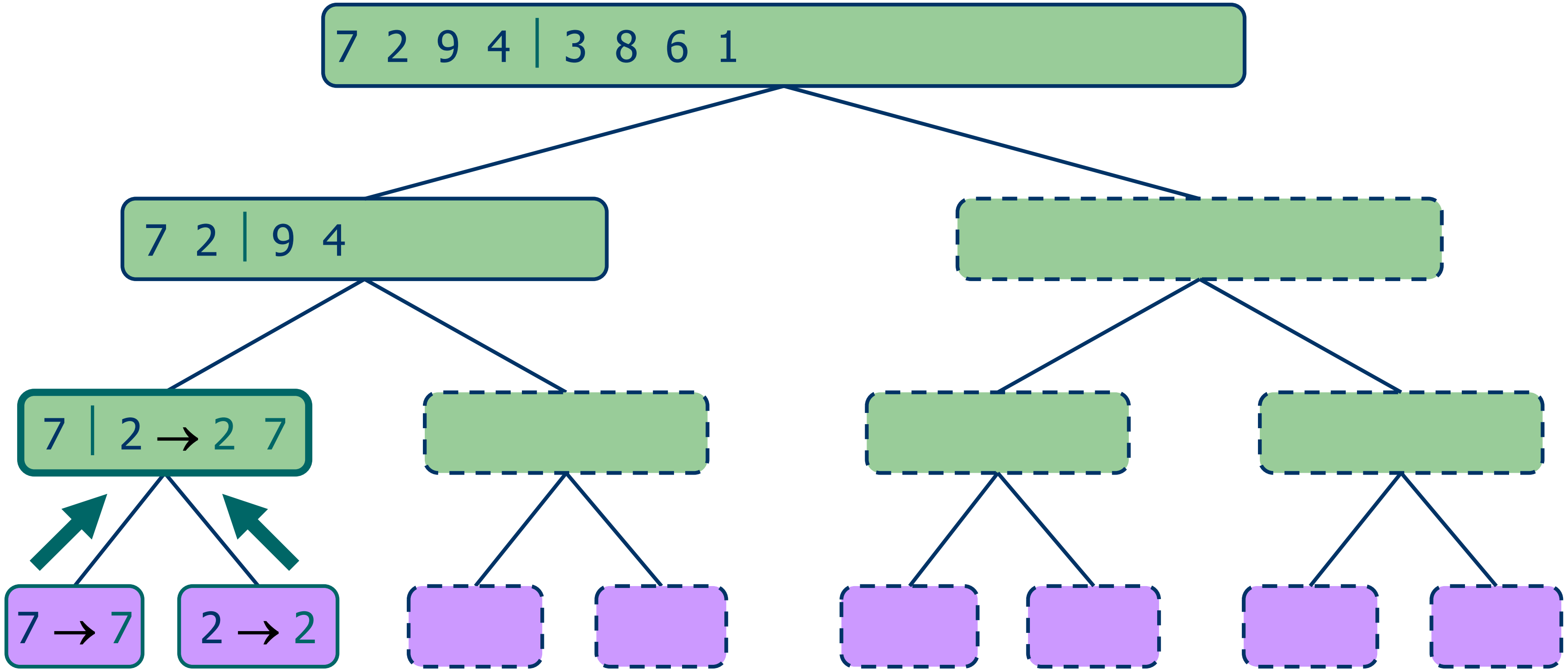
- Recursive call, base case



Divide and Conquer: Merge Sort ...

Execution Example (cont.)

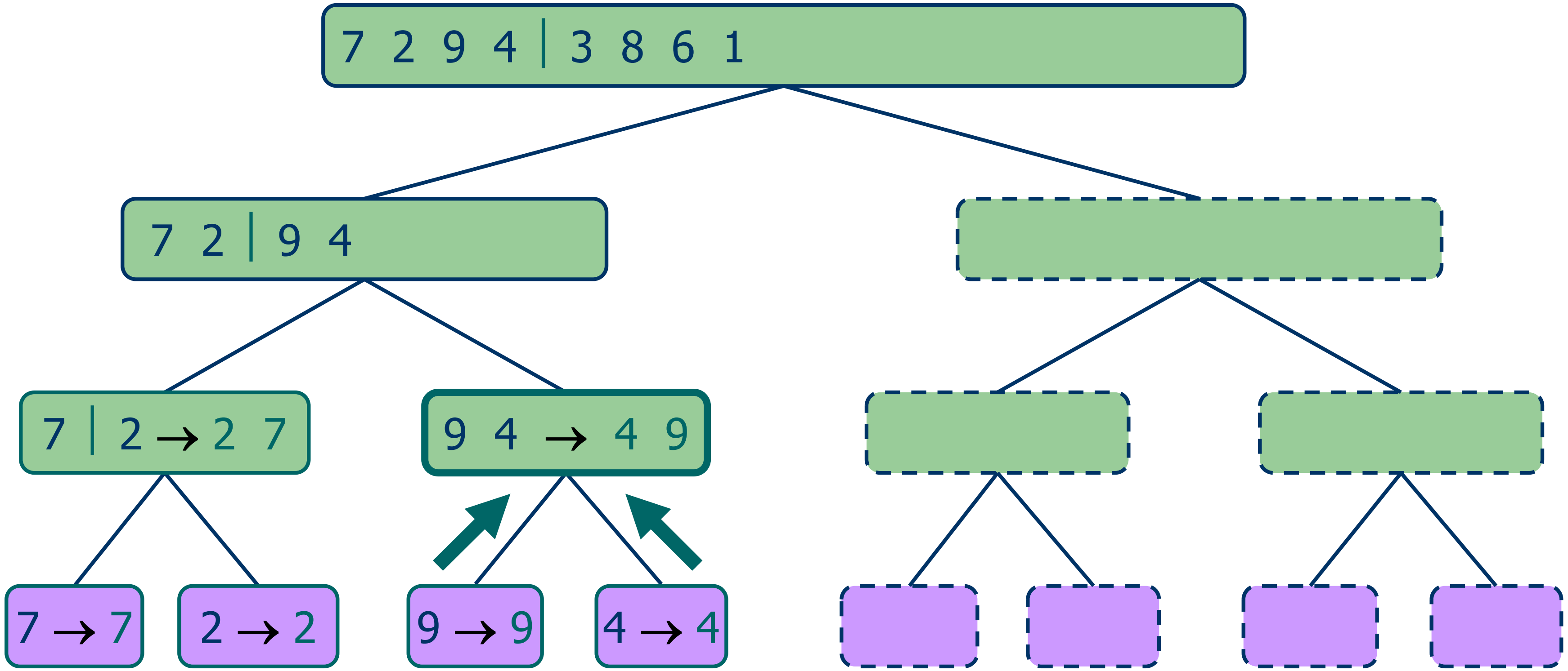
- Merge



Divide and Conquer: Merge Sort ...

Execution Example (cont.)

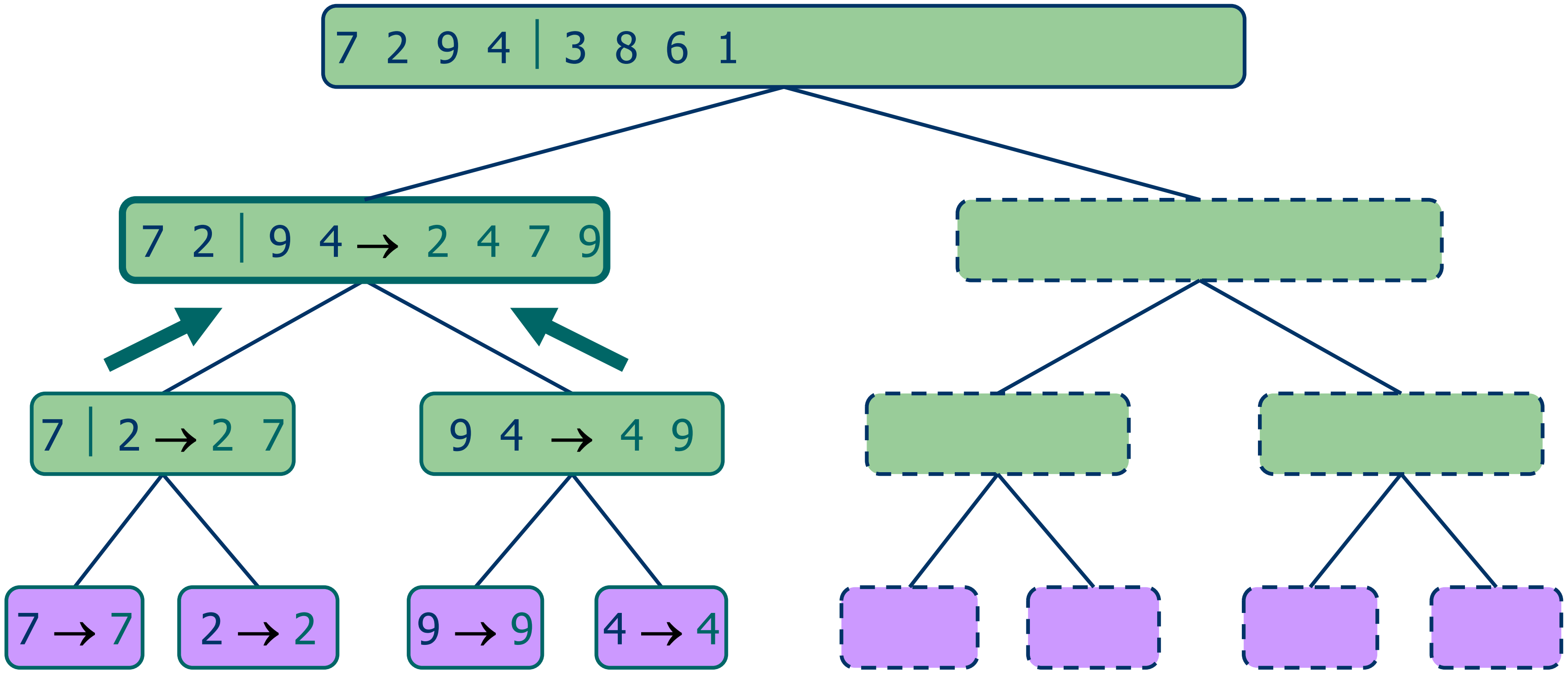
- Recursive call, …, base case, merge



Divide and Conquer: Merge Sort ...

Execution Example (cont.)

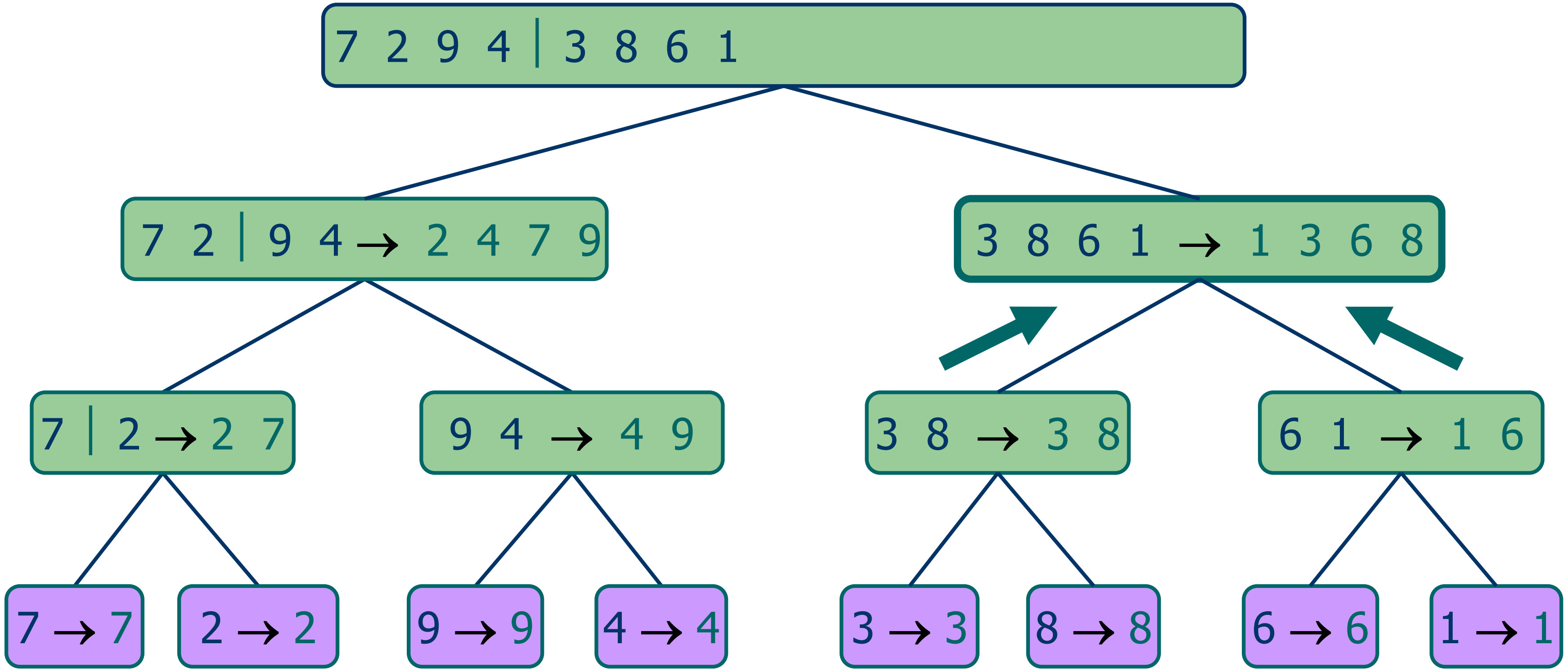
- Merge



Divide and Conquer: Merge Sort ...

Execution Example (cont.)

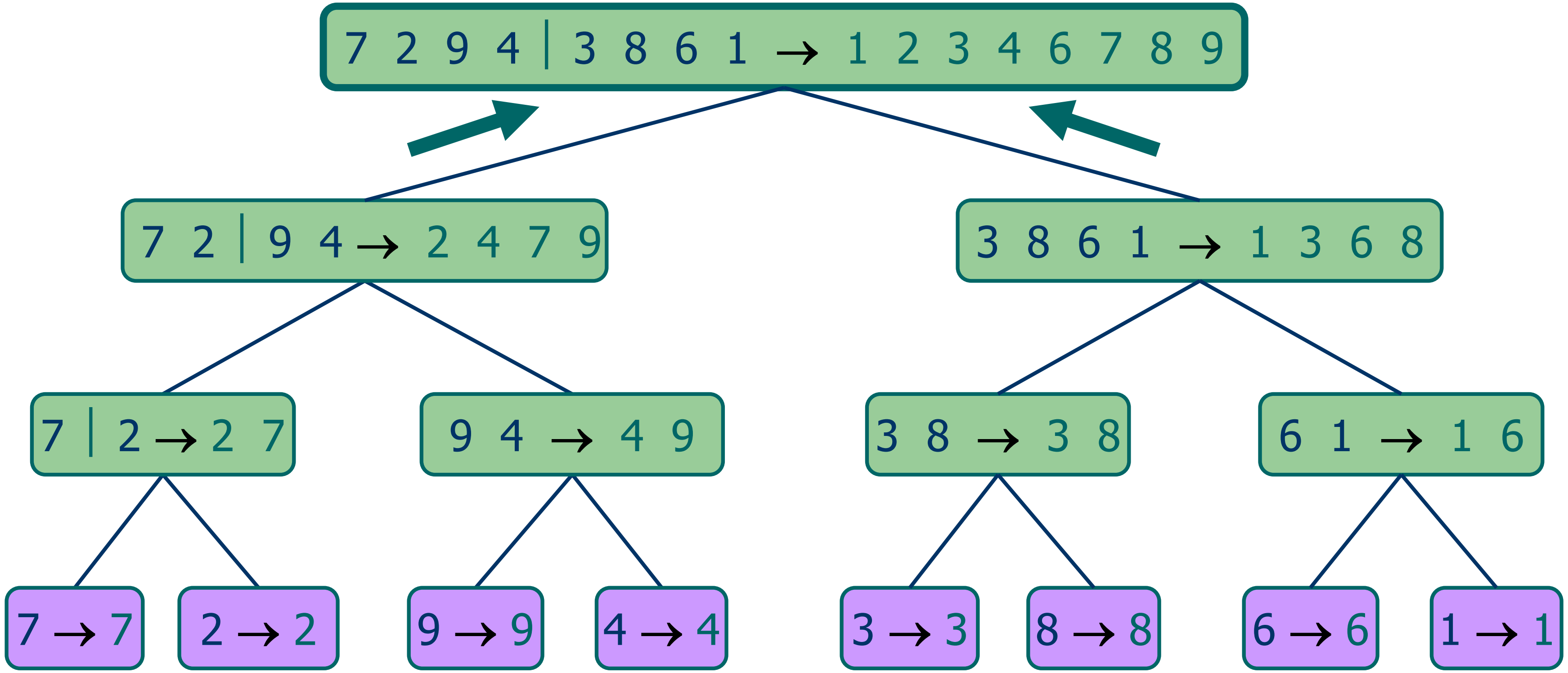
- Recursive call, …, merge, merge



Divide and Conquer: Merge Sort ...

Execution Example (cont.)

- Merge



Divide and Conquer: Merge Sort ...

INPUT: a sequence of *n* numbers stored in array A

OUTPUT: an ordered sequence of *n* numbers

```
MergeSort (A, p, r)   // sort A[p..r] by divide & conquer
if p < r
    then q ← ⌊(p+r)/2⌋
        MergeSort (A, p, q)
        MergeSort (A, q+1, r)
    Merge (A, p, q, r) // merges A[p..q] with A[q+1..r]
```

Initial Call: MergeSort(A, 1, n)

Divide and Conquer: Merge Sort ...

Accredited with A Grade by NAAC

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

```
n1 ← q – p + 1
n2 ← r – q
    for i ← 1 to n1
        do L[i] ← A[p + i – 1]
    for j ← 1 to n2
        do R[j] ← A[q + j]
    L[n1+1] ← ∞
    R[n2+1] ← ∞
    i ← 1
    j ← 1
    for k ← p to r
        do if L[i] ≤ R[j]
            then A[k] ← L[i]
                i ← i + 1
            else A[k] ← R[j]
                j ← j + 1
```

Input: Array containing sorted subarrays $A[p..q]$ and $A[q+1..r]$.

Output: Merged sorted subarray in $A[p..r]$.

Sentinels, to avoid having to check if either subarray is fully copied at each step.

Divide and Conquer: Merge Sort ...

Merge – Example

A

…	1	6	8	9	26	32	42	43	…

k

L

6	8	26	32	∞

i

R

1	9	42	43	∞

j

Divide and Conquer: Merge Sort ...

Analysis of Merge Sort

Running time $T(n)$ of Merge Sort:

- Divide: computing the middle takes $\Theta(1)$
- Conquer: solving 2 subproblems takes $2T(n/2)$
- Combine: merging n elements takes $\Theta(n)$
- Total:

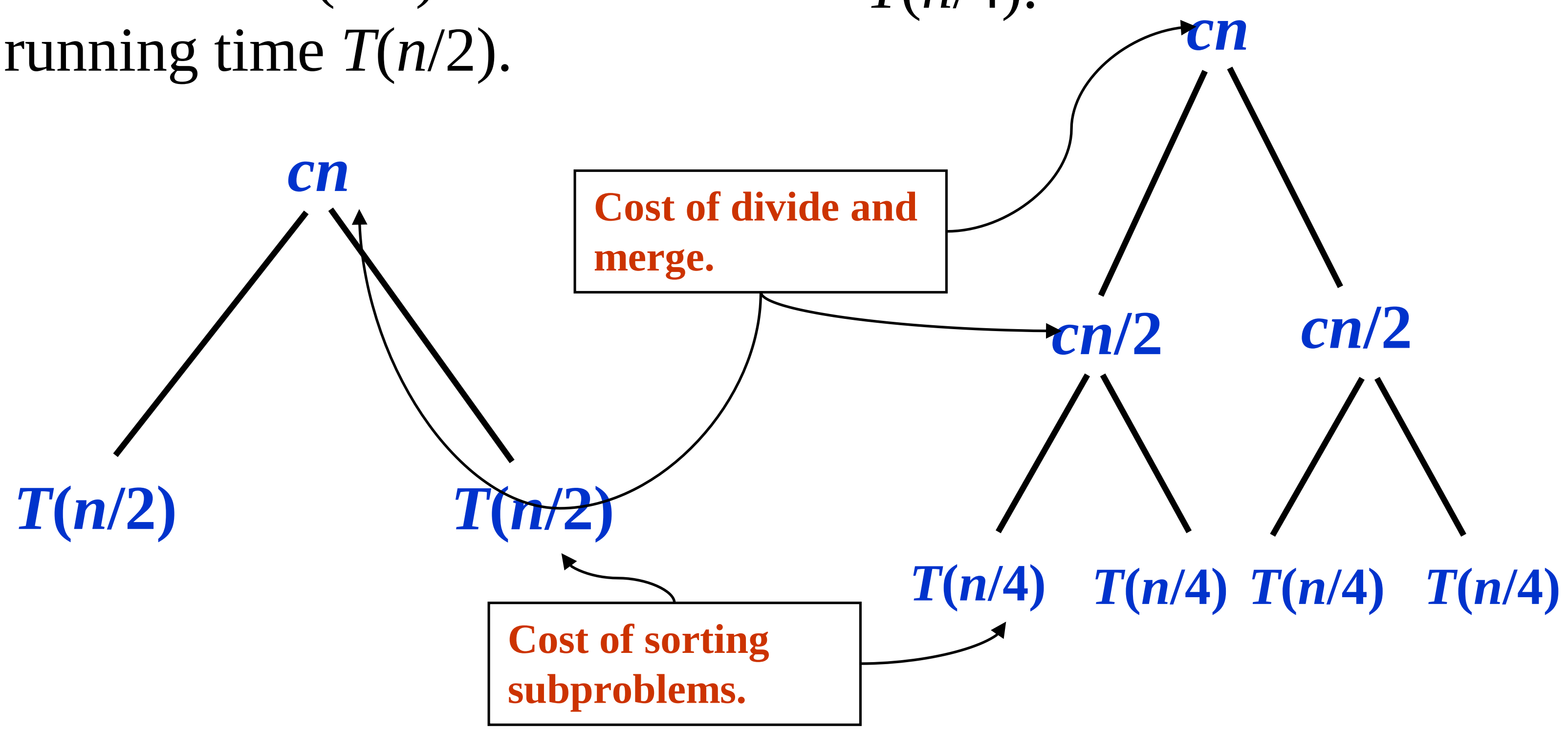
$$T(n) = \Theta(1) \qquad \text{if } n = 1$$
$$T(n) = 2T(n/2) + \Theta(n) \qquad \text{if } n > 1$$

$$\Rightarrow T(n) = \Theta(n \lg n)$$

Recursion Tree for Merge Sort

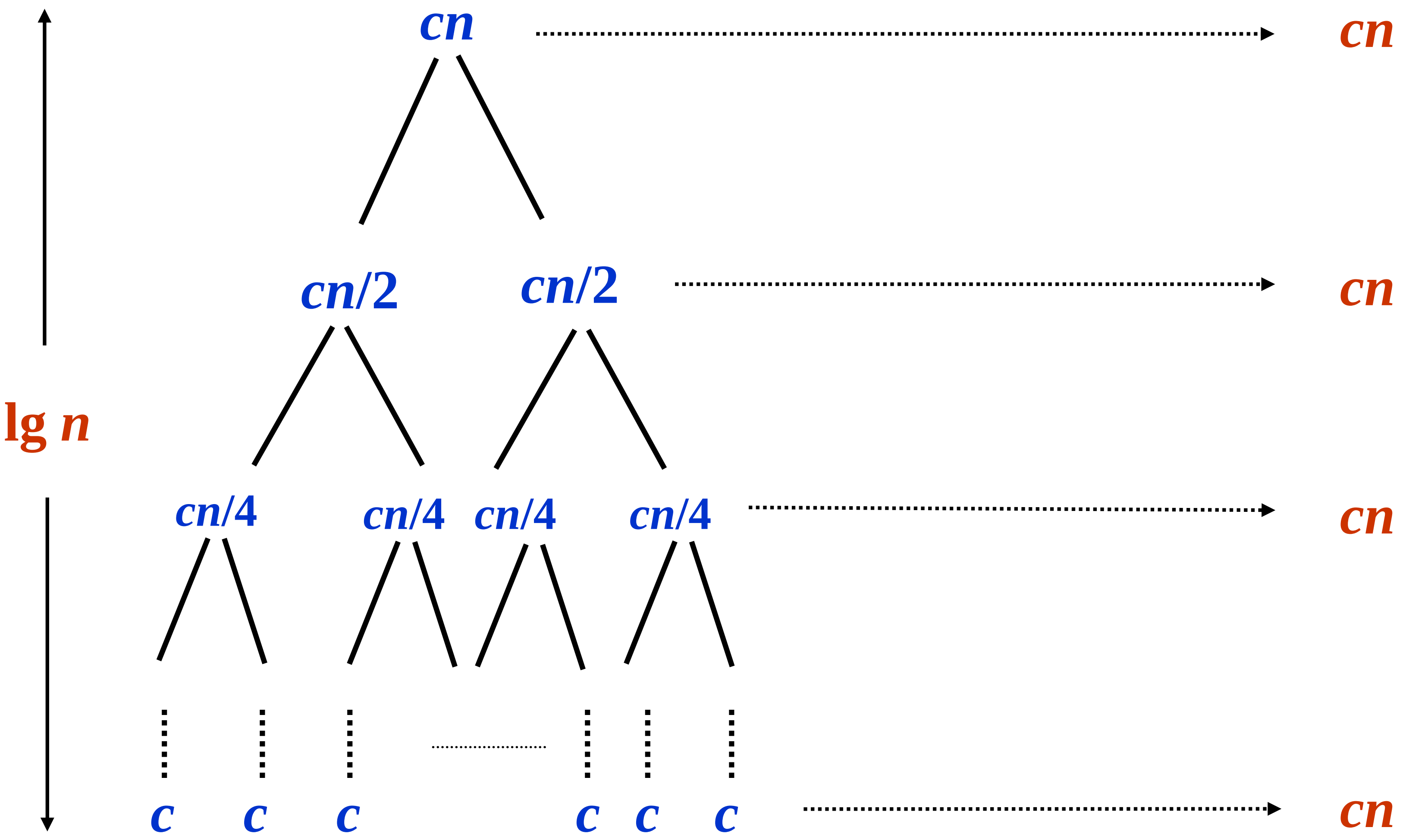
For the original problem, we have a cost of cn, plus two subproblems each of size $(n/2)$ and running time $T(n/2)$.

Each of the size $n/2$ problems has a cost of $cn/2$ plus two subproblems, each costing $T(n/4)$.



Recursion Tree for Merge Sort

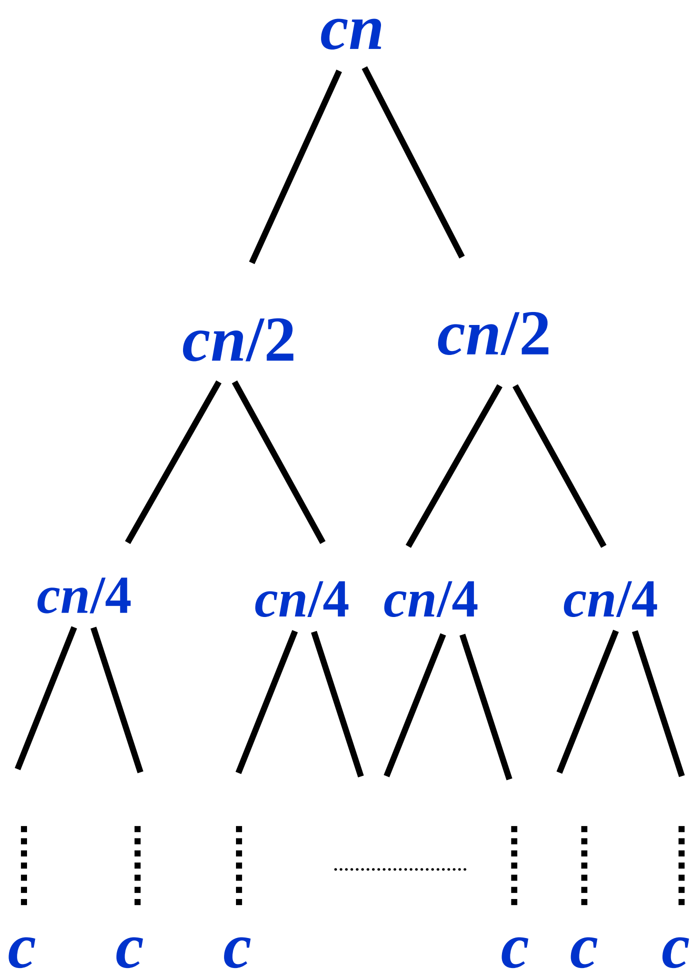
Continue expanding until the problem size reduces to 1.



Total : $cn\lg n + cn$

Recursion Tree for Merge Sort

Continue expanding until the problem size reduces to 1.



- Each level has total cost ***cn***.
- Each time we go down one level, the number of subproblems doubles, but the cost per subproblem halves $\Rightarrow$ *cost per level remains the same*.
- There are lg n + 1 levels, height is lg n. (Assuming n is a power of 2.)
    - Can be proved by induction.
- Total cost = sum of costs at each level = (lg n + 1)cn = cnlg$n$ + cn = $\Theta(n \lg n)$.

Any Questions ?

Dr. Anand Singh Jalal
Professor
Email: asjalal@gla.ac.in