CSE-281: Data Structures and Algorithms

Divide and Conquer

Ref: Schaum's Outline Series, Theory and problems of Data Structures
By Seymour Lipschutz

And Online Resource

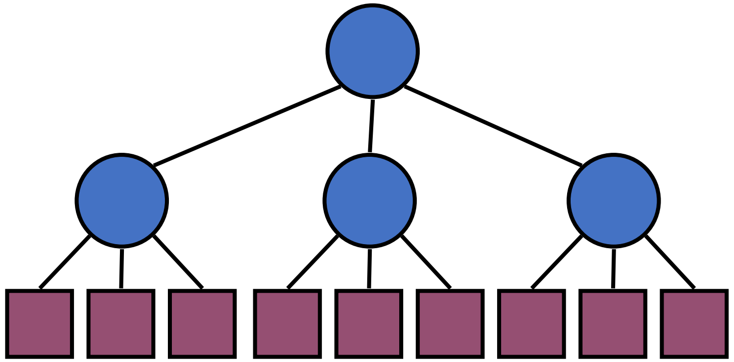
Eftekhar Hossain
Assistant Professor
Dept. of ETE, CUET

Topics to be Covered

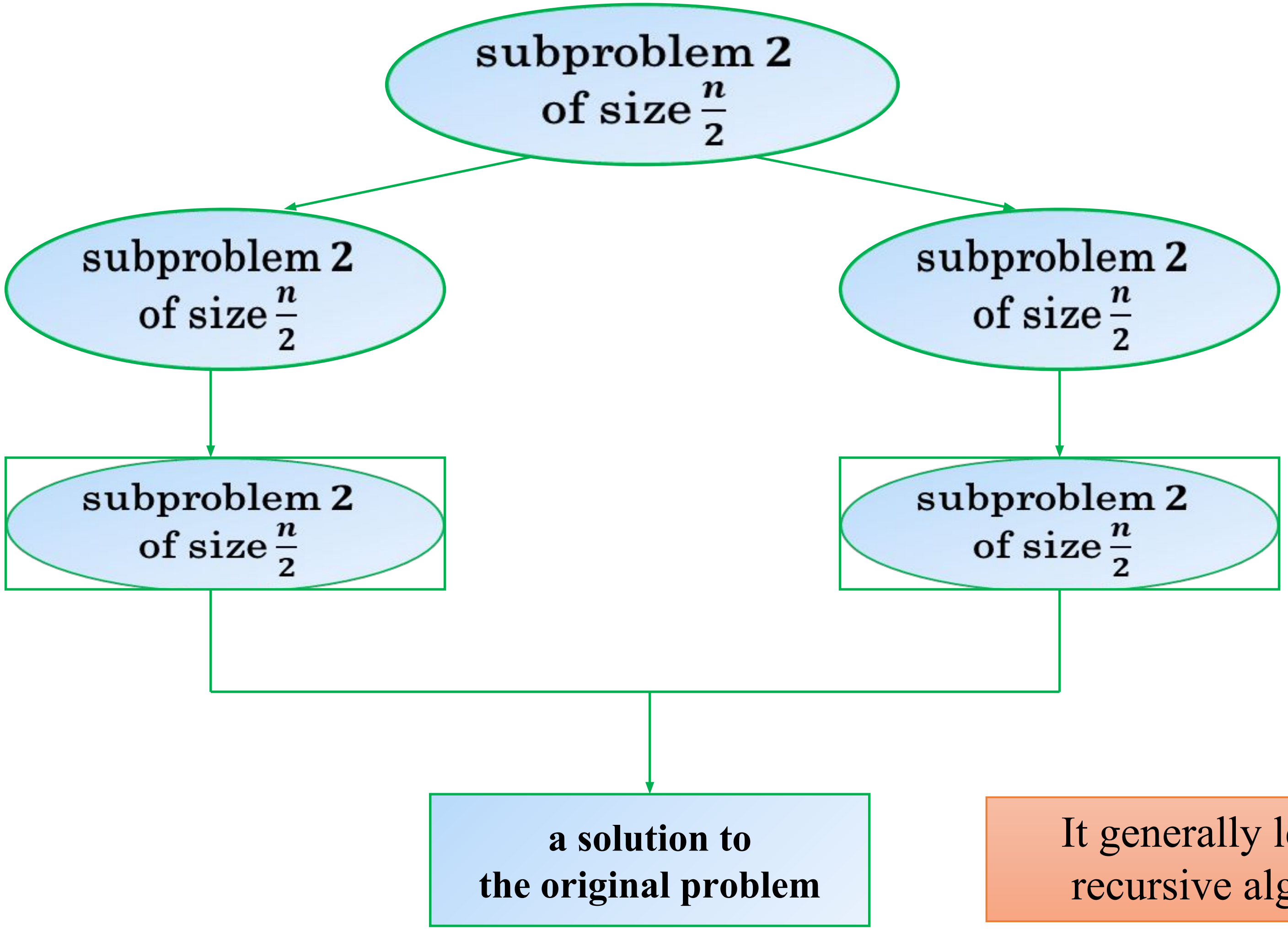
- Divide and Conquer
- Quick Sort
- Merge Sort
- Complexity

Divide and Conquer

- Divide-and conquer is a general algorithm design paradigm:
  - Divide the input data S in two or more disjoint subsets S_1, S_2, …
  - Conquer the subproblems by solving them recursively
    - base case for the recursion: If the subproblems are small enough just solve them
  - Combine the solutions for S_1, S_2, …, into a solution for S
- Analysis can be done using **recurrence equations**



Divide and Conquer



It generally leads to a recursive algorithm!

Quick Sort

0	1	2	3	4	5	6	7
lb							ub
5	3	1	9	8	2	4	7
i	i	i	i			j	j

Pivot = 5

0	1	2	3	4	5	6	7
5	3	1	4	8	2	9	7
			i	i	j	j	

0	1	2	3	4	5	6	7
5	3	1	4	2	8	9	7
				ij	ij		

0	1	2	3	4	5	6	7
2	3	1	4	5	8	9	7

Partition 1

0	1	2	3	4	5	6	7
lb			ub				
2	3	1	4	5	8	9	7
i	i	j	j				

Pivot = 2

0	1	2	3	4	5	6	7
2	1	3	4	5	8	9	7
	ij	ij					

0	1	2	3	4	5	6	7
1	2	3	4	5	8	9	7

Partition 2

0	1	2	3	4	5	6	7
		lb	ub				
1	2	3	4	5	8	9	7
		ij	ij				

Pivot = 3

```
A[i]<=Pivot
i++
```

```
A[j]>Pivot
J--
```

Quick Sort

					lb		ub
0	1	2	3	4	5	6	7
1	2	3	4	5	8	9	7
					i	i	j

Pivot = 8

0	1	2	3	4	5	6	7
1	2	3	4	5	8	7	9
						ij	i j

0	1	2	3	4	5	6	7
1	2	3	4	5	7	8	9

Algorithm

```
Partition (A, lb, ub)
{
Pivot = A [lb]
i = lb
j = ub
```

```
While(i<j){

While (A[i]<= pivot)
{
i ++
}

While (A[j] > pivot)
{
j --
}

if (i < j )
{
 swap (A[i], A[j])
}
}

swap(A[j], A[lb])
return j
}
```

```
QuickSort (A,lb,ub)
{
If (lb < ub)
{
Loc = partition(A,lb,ub)
QuickSort(A,lb,loc-1)
QuickSort(A,loc+1,ub)
}

}
```

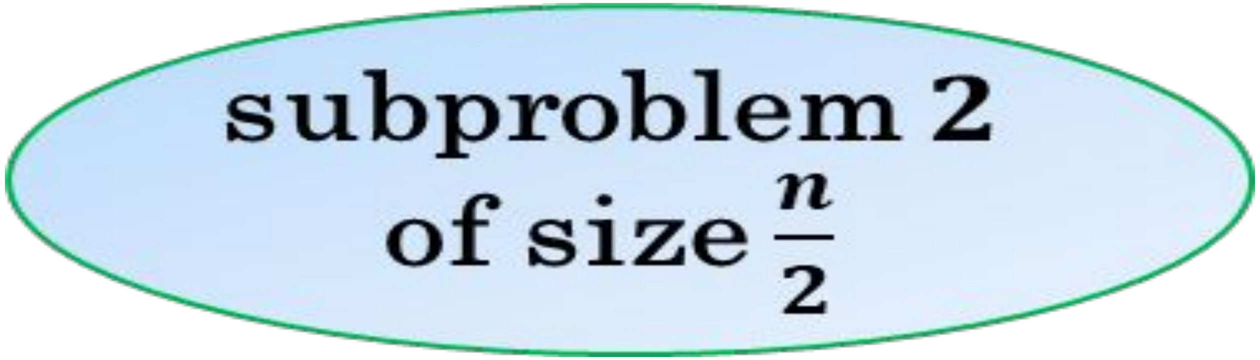
Recurrence Relation

Recurrences are used to analyze the computational complexity of divide-and-conquer algorithms.

subproblem 2 of size $\frac{n}{2}$

Quick Sort Complexity

- Best case and Average case: occurs when the list is divided into 2 parts after placement of pivot at its proper locations
- The time required for solving the given element using quicksort is given by the following recurrence relation :



Quick Sort Complexity

subproblem 2 of size $\frac{n}{2}$

subproblem 2 of size $\frac{n}{2}$

subproblem 2 of size $\frac{n}{2}$

subproblem 2 of size $\frac{n}{2}$

subproblem 2 of size $\frac{n}{2}$

subproblem 2

subproblem 2 of size $\frac{n}{2}$

subproblem 2 of size $\frac{n}{2}$

subproblem 2

subproblem 2 of size $\frac{n}{2}$

subproblem 2 of size $\frac{n}{2}$

Quick Sort Complexity

subproblem 2
of size $\frac{n}{2}$

subproblem 2

subproblem 2
of size $\frac{n}{2}$

subproblem 2
of size $\frac{n}{2}$

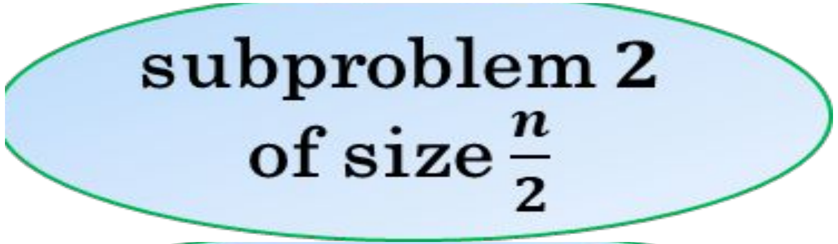
subproblem 2
of size $\frac{n}{}$

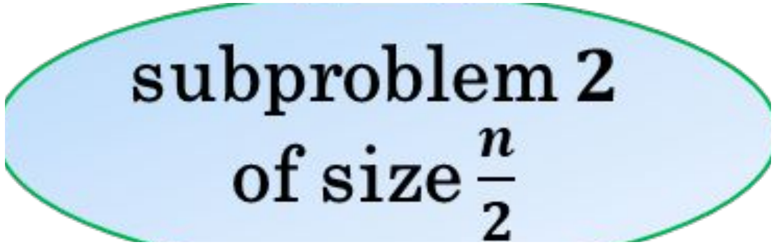
subproblem 2
of size $\frac{n}{2}$

subproblem 2
of size $\frac{n}{2}$

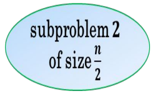
Quick Sort Complexity

- Worst case occurs when the list is sorted in decreasing order or the elements are same. For example-

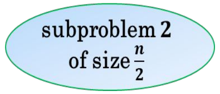




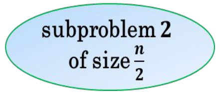
5 4 3 2 1



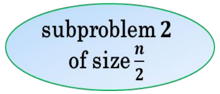
1 4 3 2 5



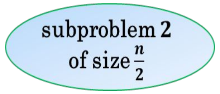
1 4 3 2 5



1 2 3 4 5



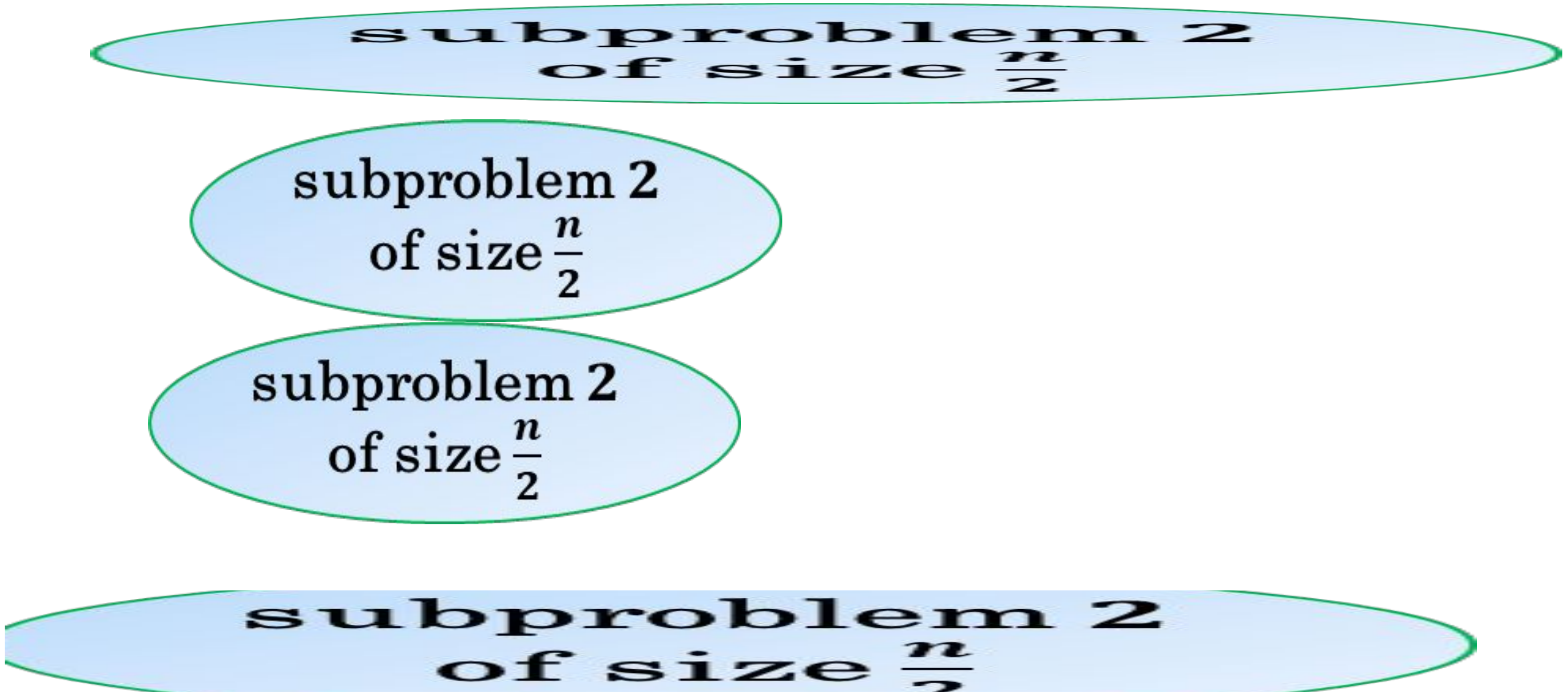
1 2 3 4 5



- This continues until there is single element in list

Quick Sort Complexity

Total number of comparisons required =



Merge Sort

- Merge-sort is a sorting algorithm based on the divide-and-conquer paradigm
  - It has $\boldsymbol{O}\,(\boldsymbol{n} \log \boldsymbol{n})$ running time

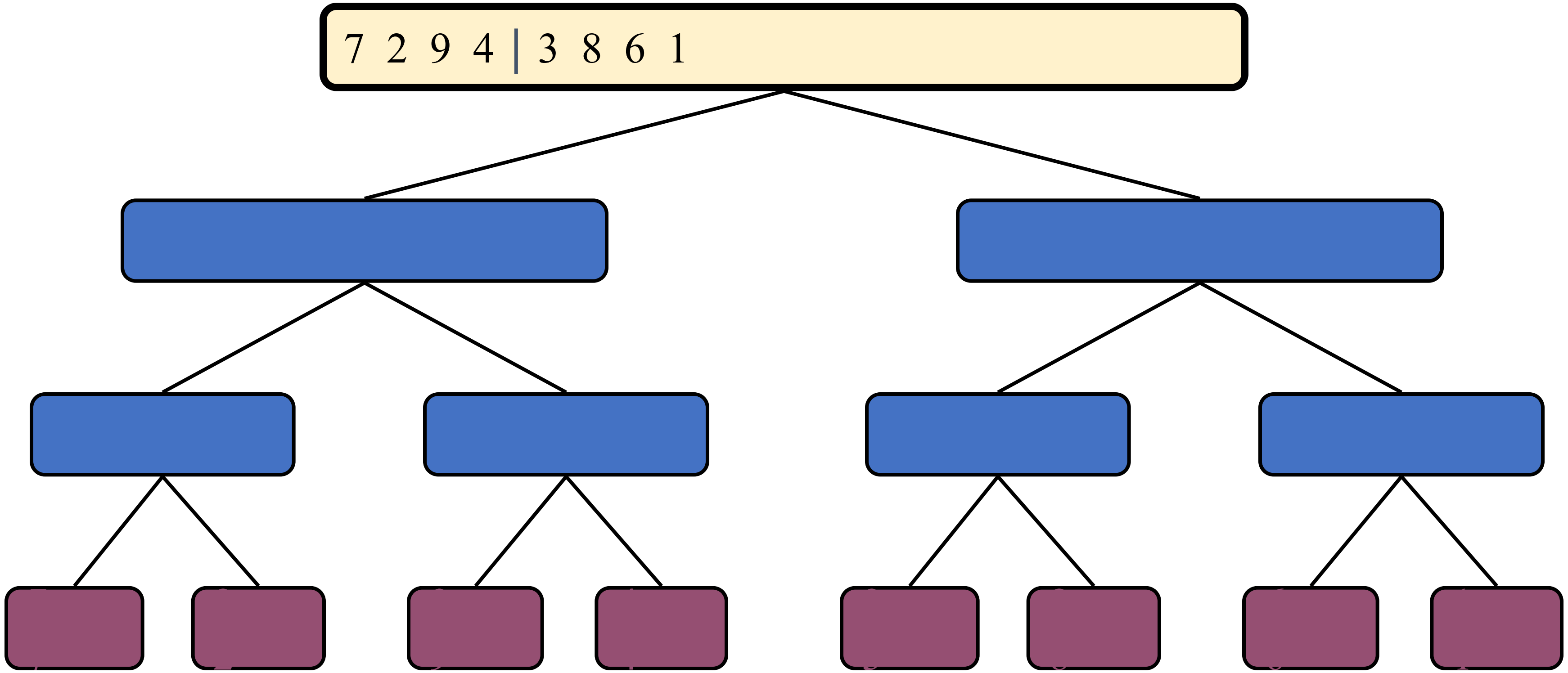
- Merge-sort on an input sequence ***S*** with ***n*** elements consists of three steps:
  - Divide
  - Recursion
  - Conquer

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

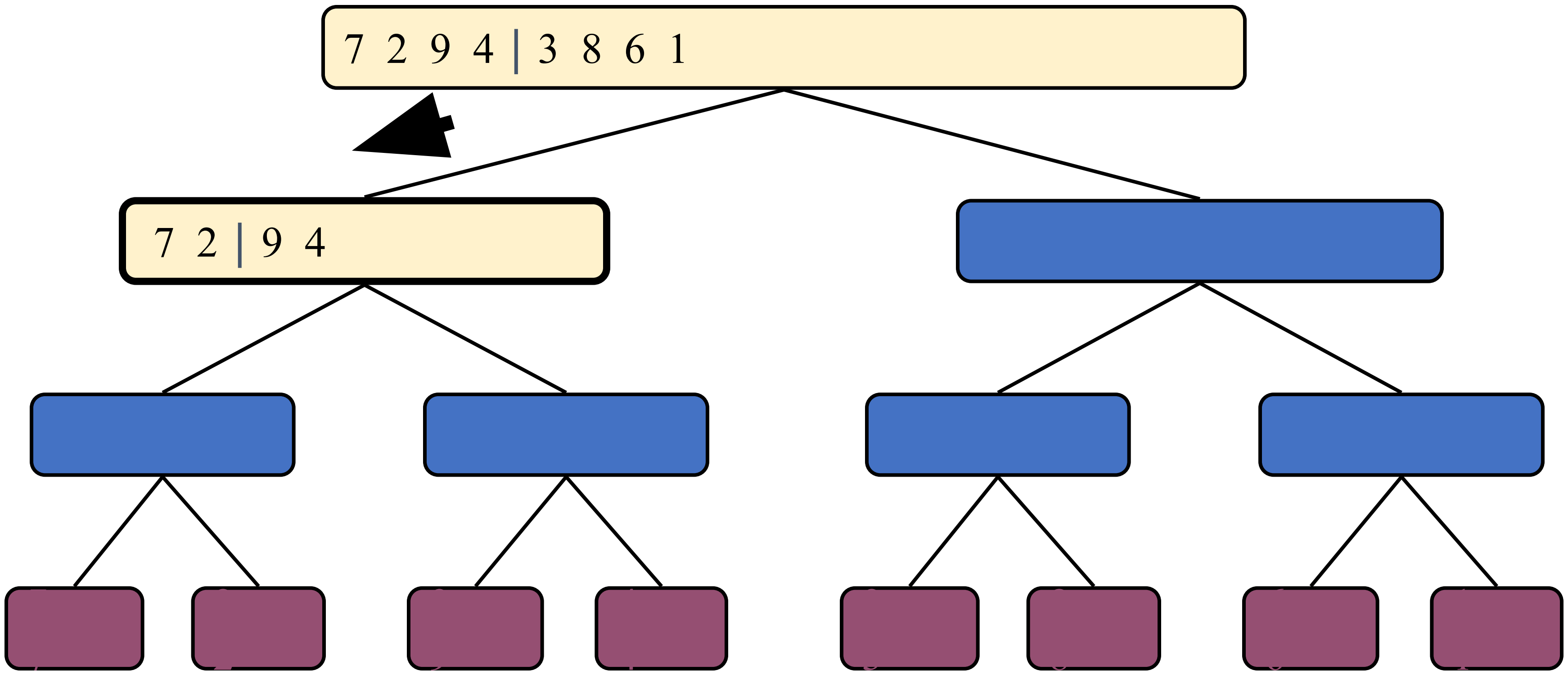
Merge Sort Tree (Example)

- Partition



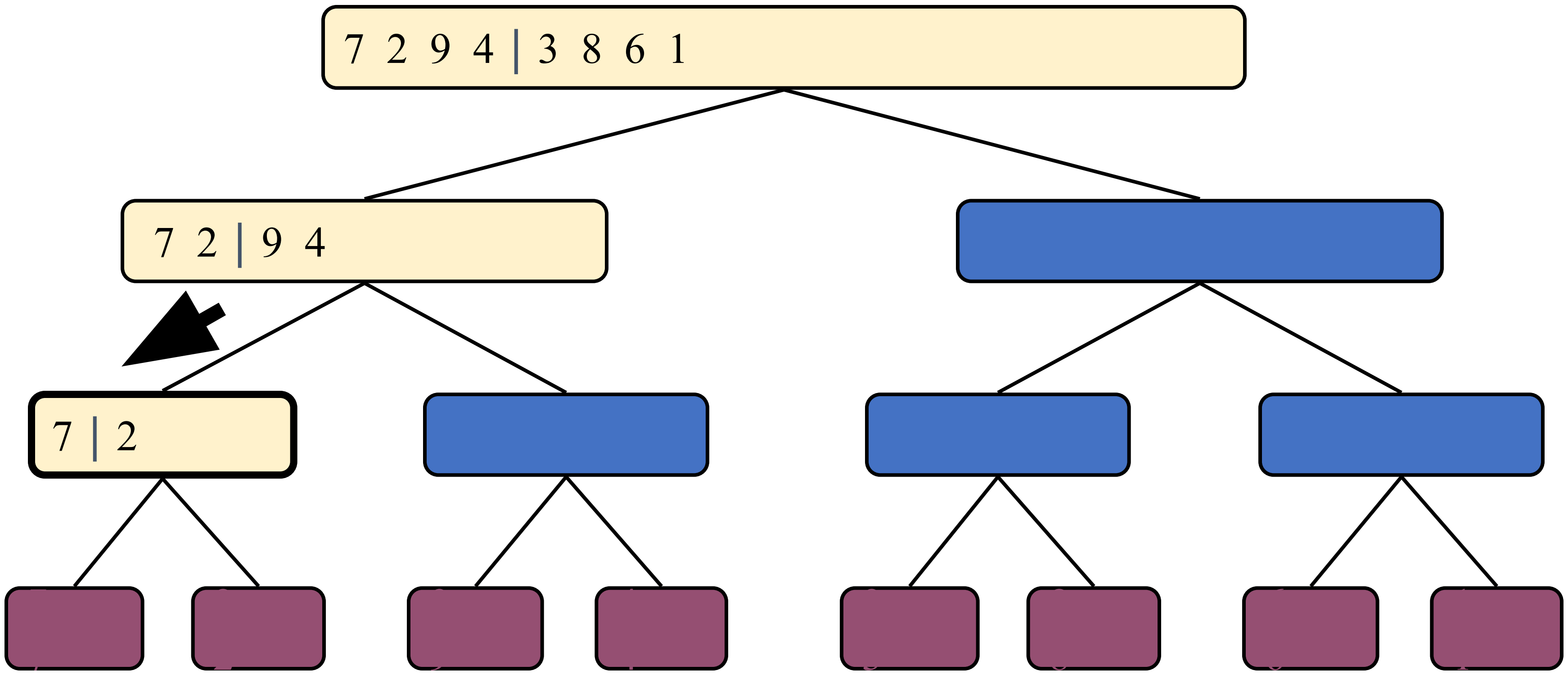
Merge Sort Tree (Example)

- Recursive call, partition



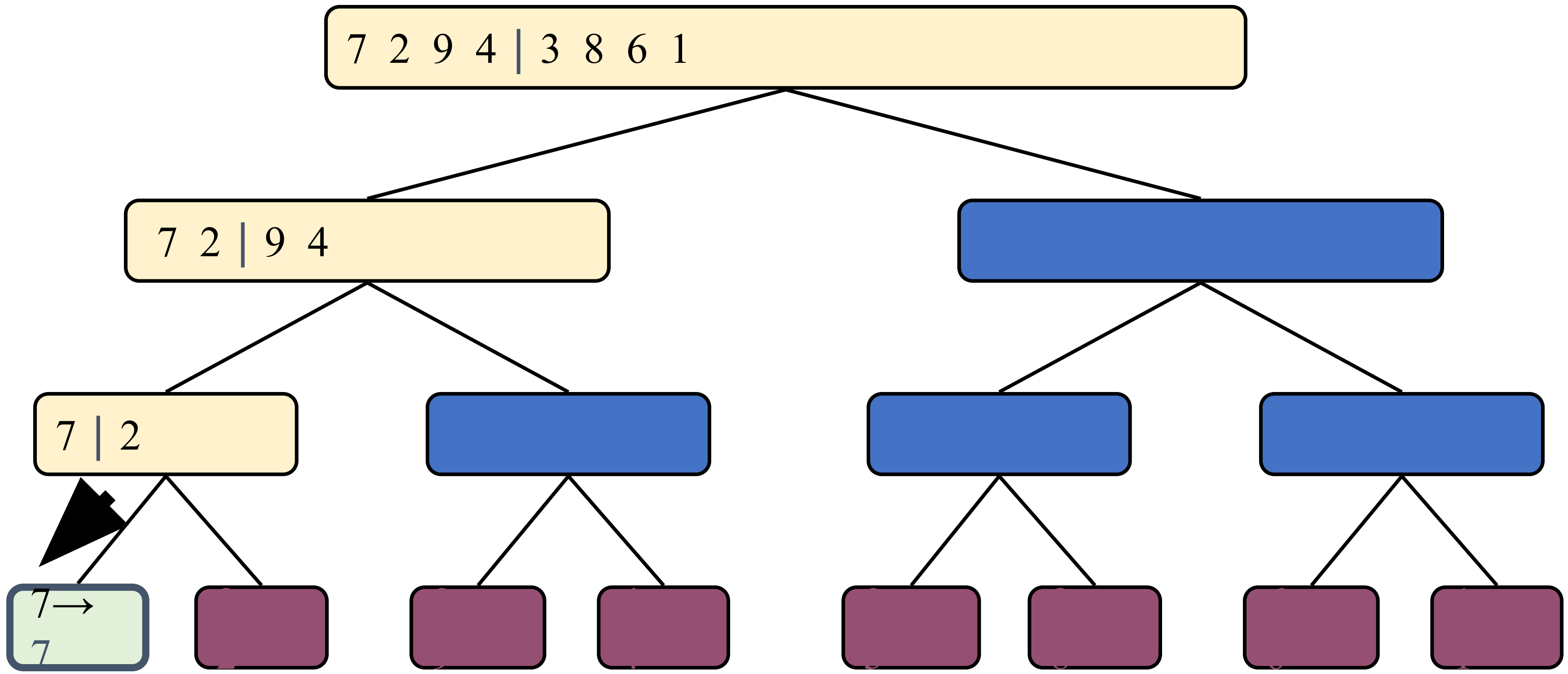
Merge Sort Tree (Example)

- Recursive call, partition



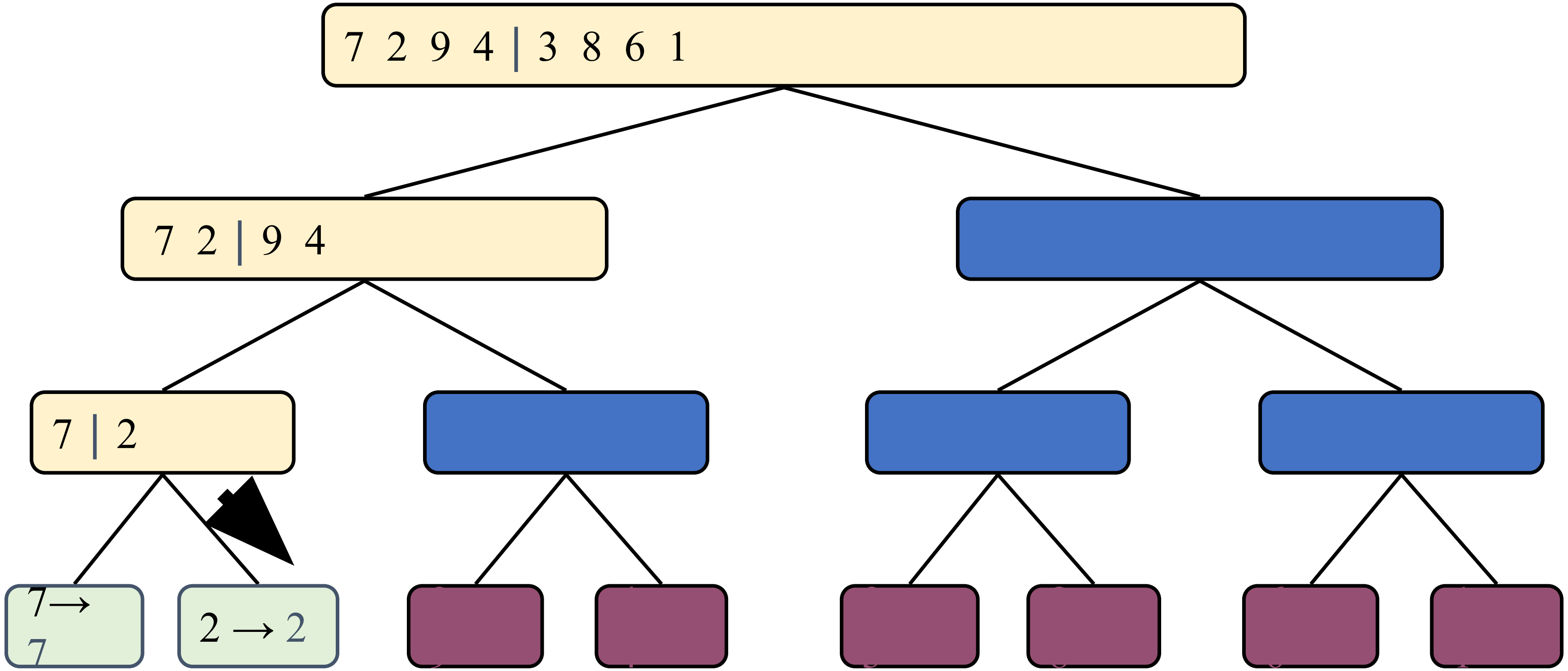
Merge Sort Tree (Example)

- Recursive call, base case



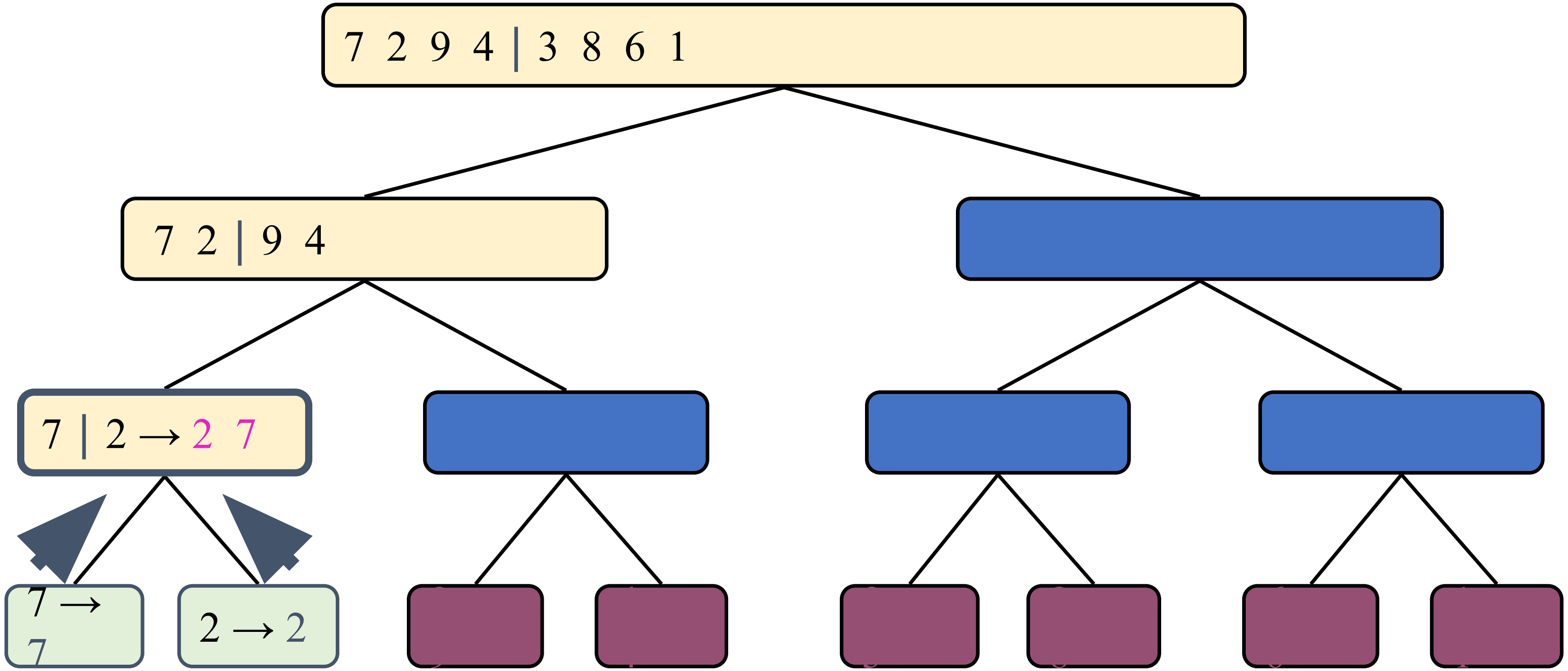
Merge Sort Tree (Example)

- Recursive call, base case



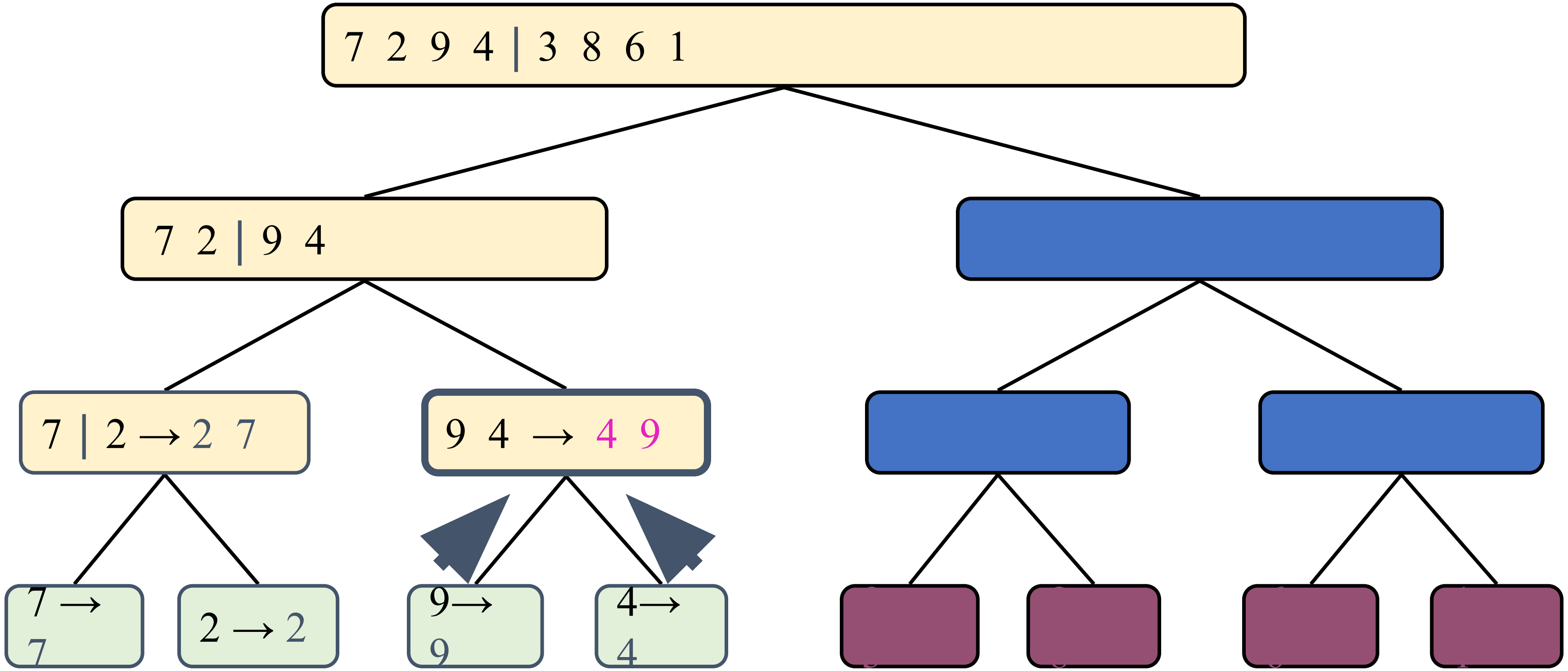
Merge Sort Tree (Example)

- Merge



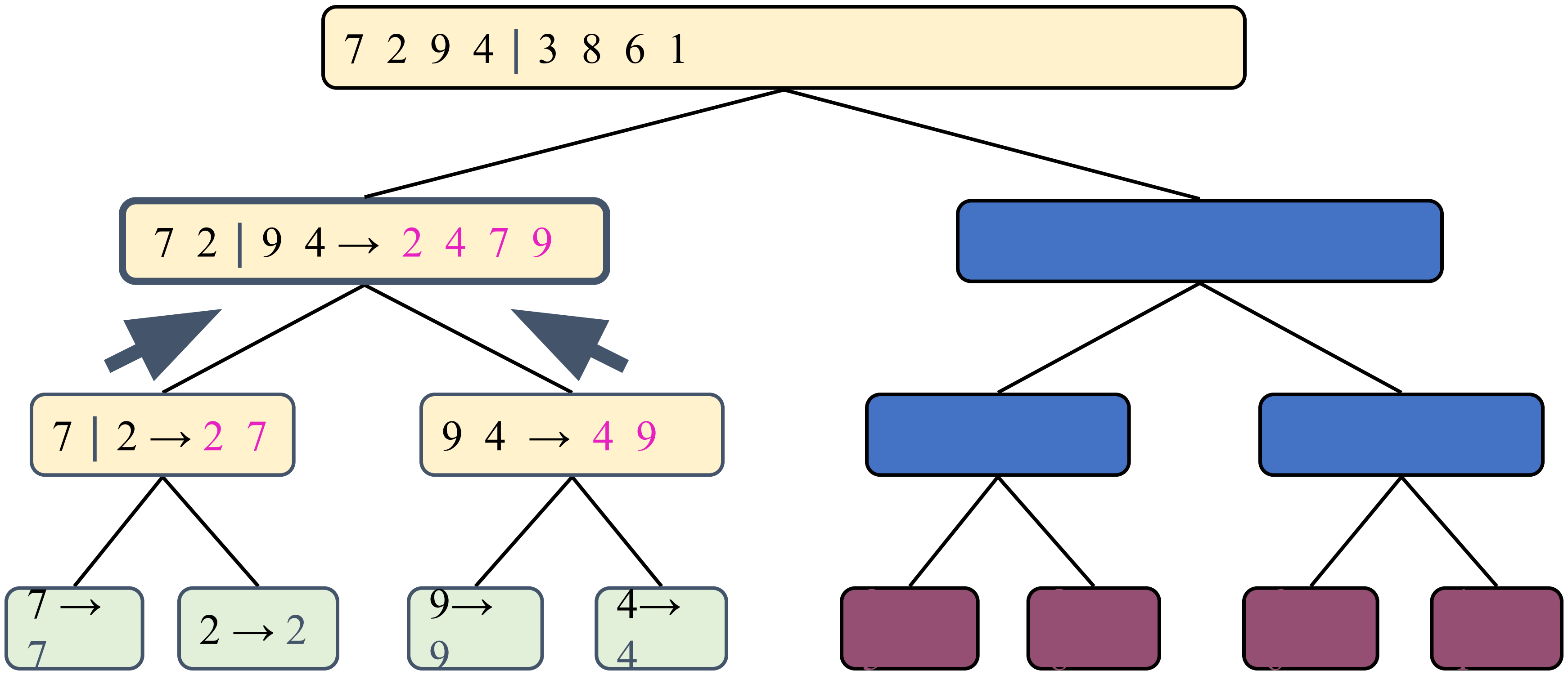
Merge Sort Tree (Example)

- Recursive call, …, base case, merge



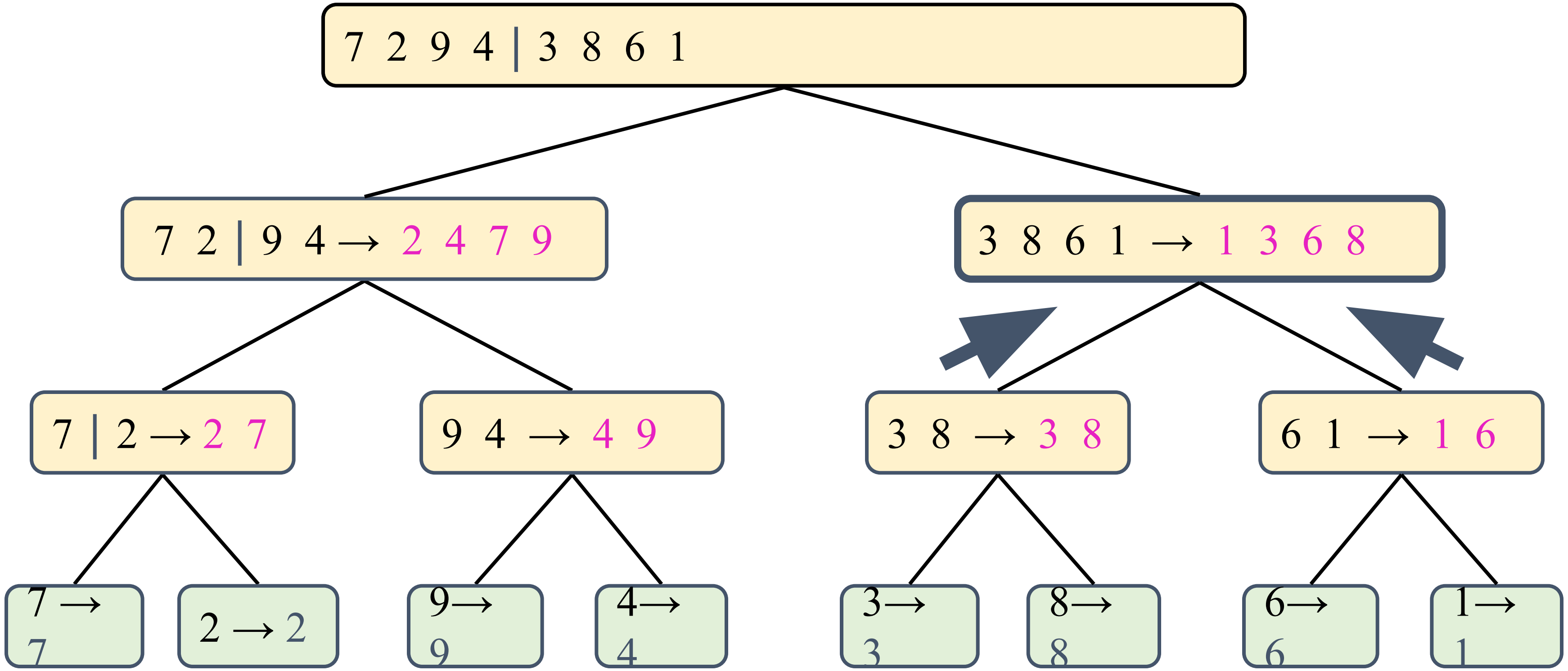
Merge Sort Tree (Example)

- Merge



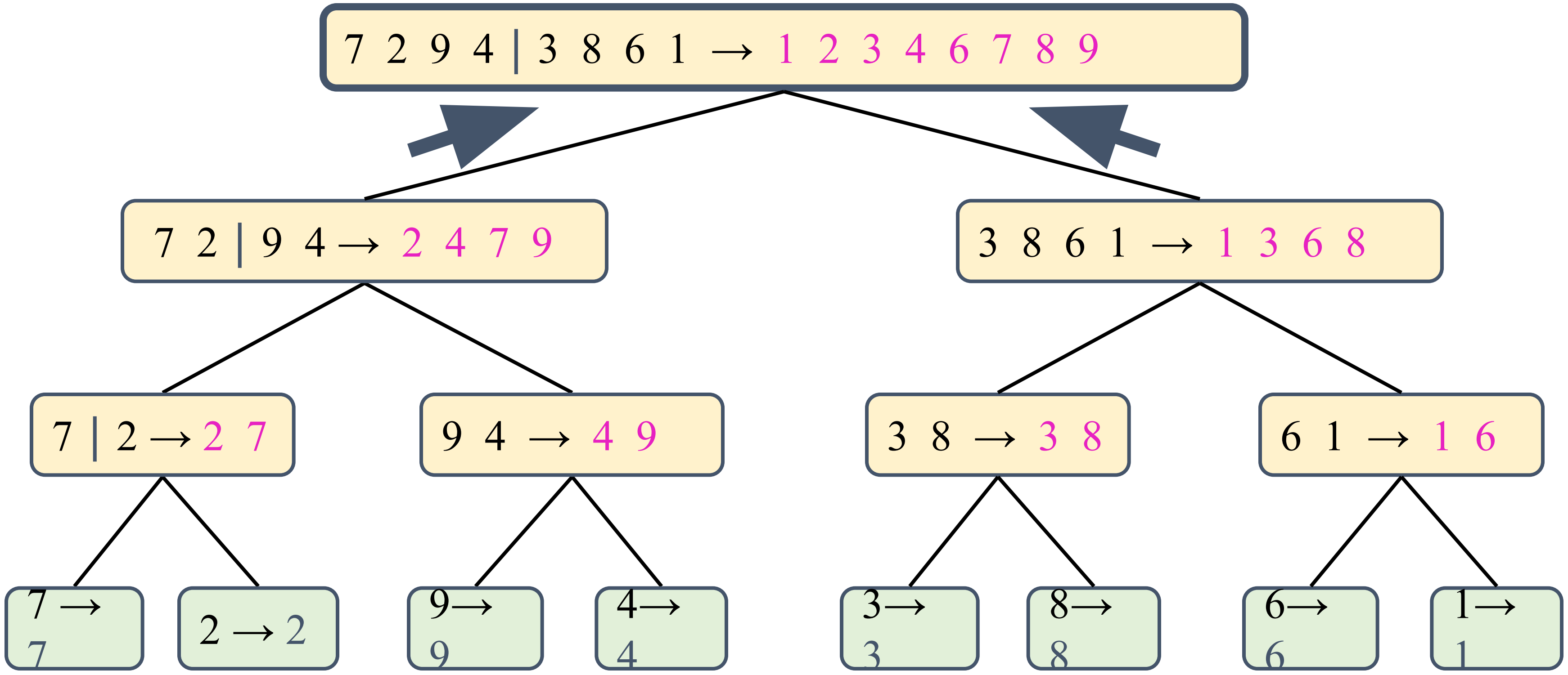
Merge Sort Tree (Example)

- Recursive call, …, merge, merge



Merge Sort Tree (Example)

- Merge



Merge Sort Tree (Example)

99	6	86	15	58	35	86	4	0

Merge Sort Tree (Example)

99	6	86	15	58	35	86	4	0

99	6	86	15

58	35	86	4	0

Merge Sort Tree (Example)

99	6	86	15	58	35	86	4	0

99	6	86	15

58	35	86	4	0

99	6

86	15

58	35

86	4	0

Merge Sort Tree (Example)

99	6	86	15	58	35	86	4	0

99	6	86	15

58	35	86	4	0

99	6

86	15

58	35

86	4	0

99

6

86

15

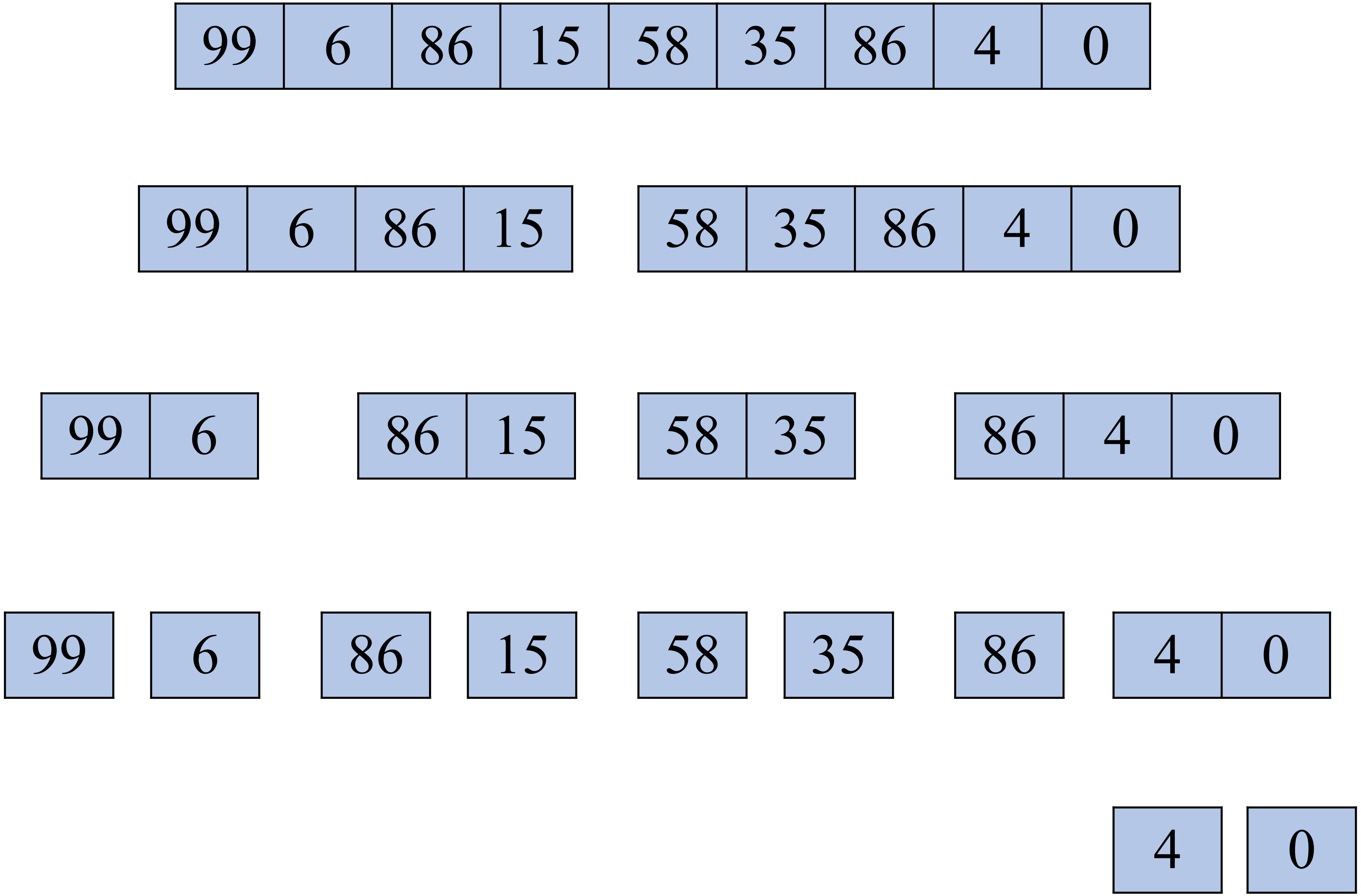
58

35

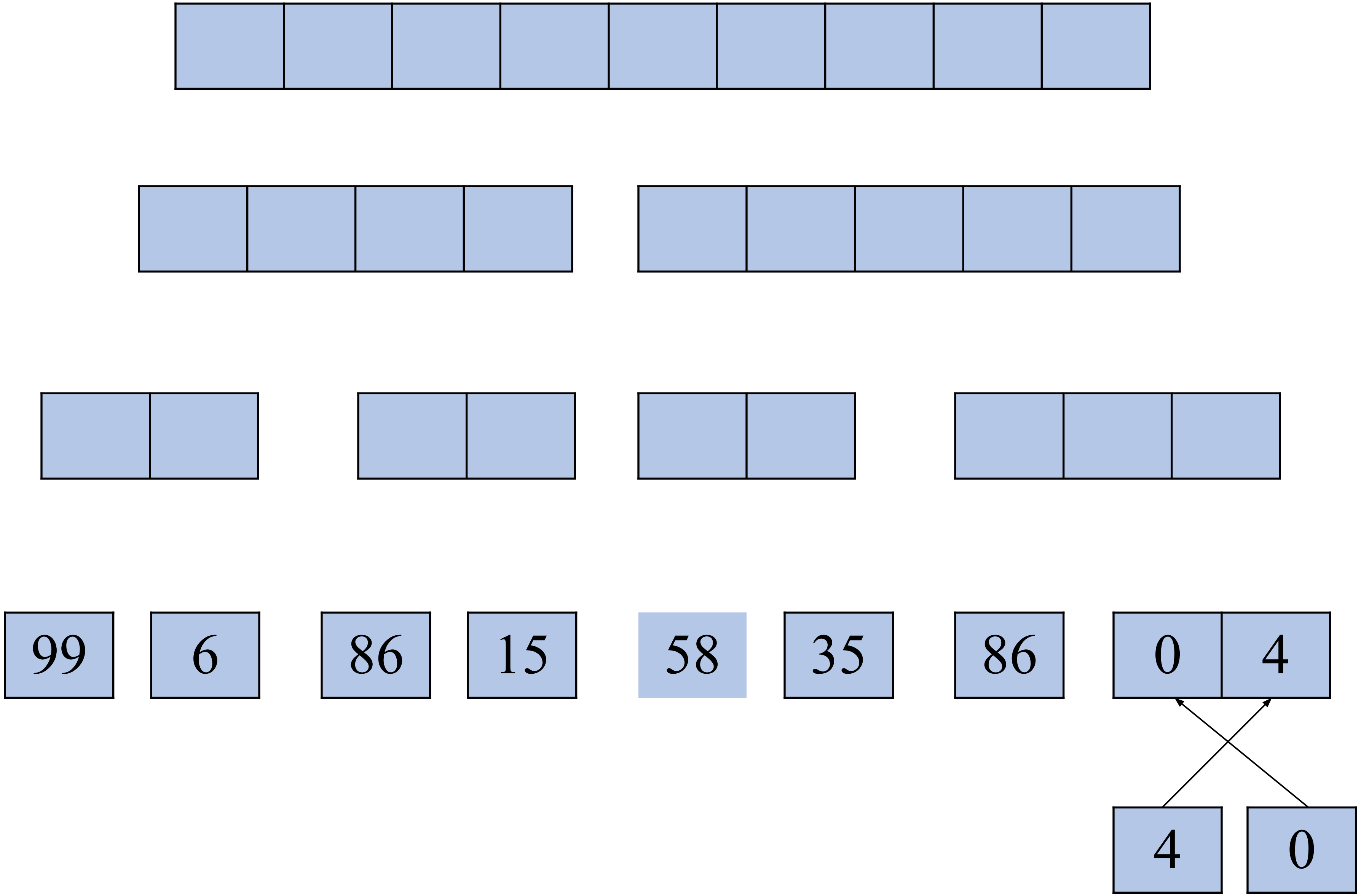
86

4	0

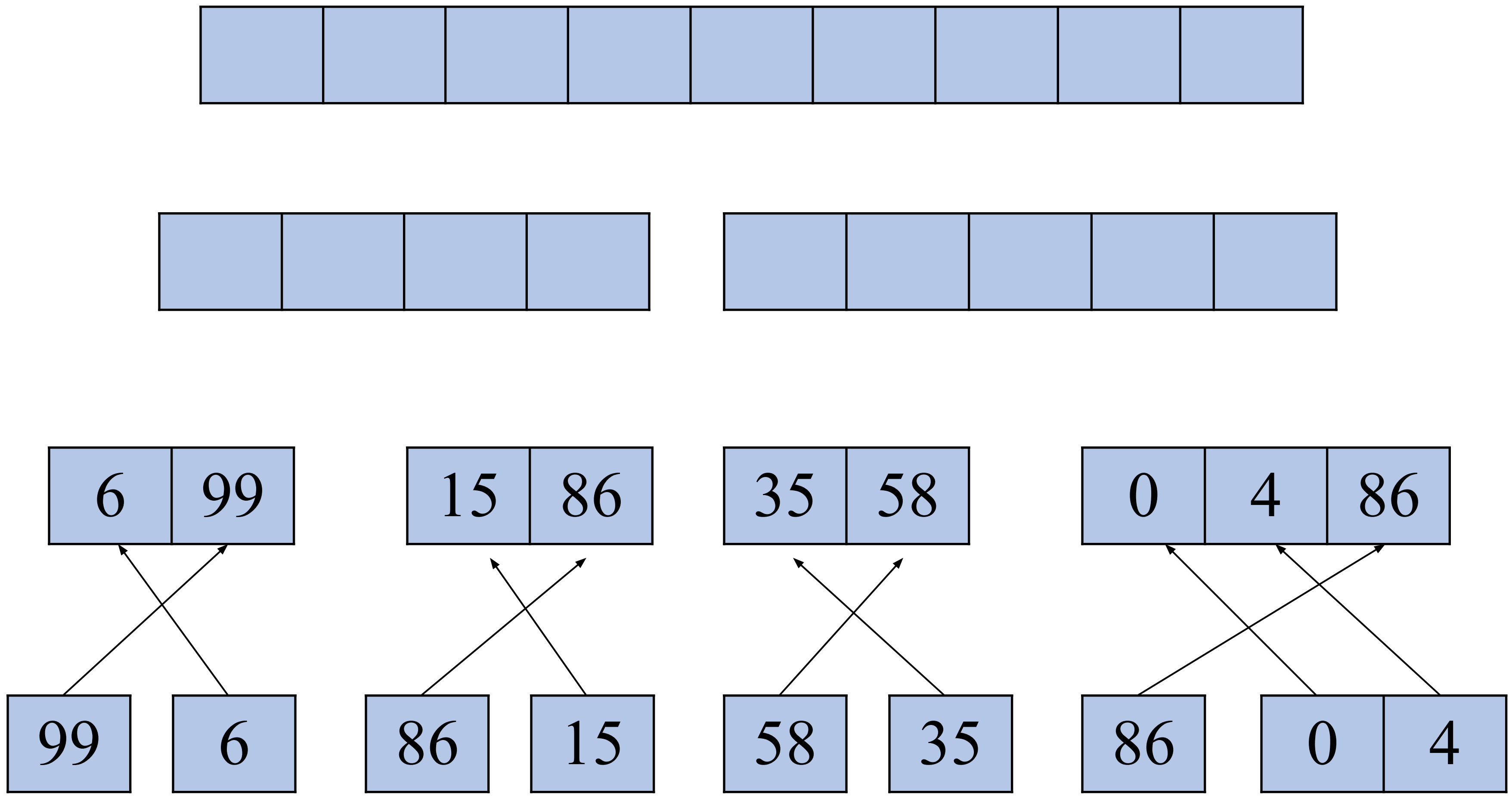
Merge Sort Tree (Example)



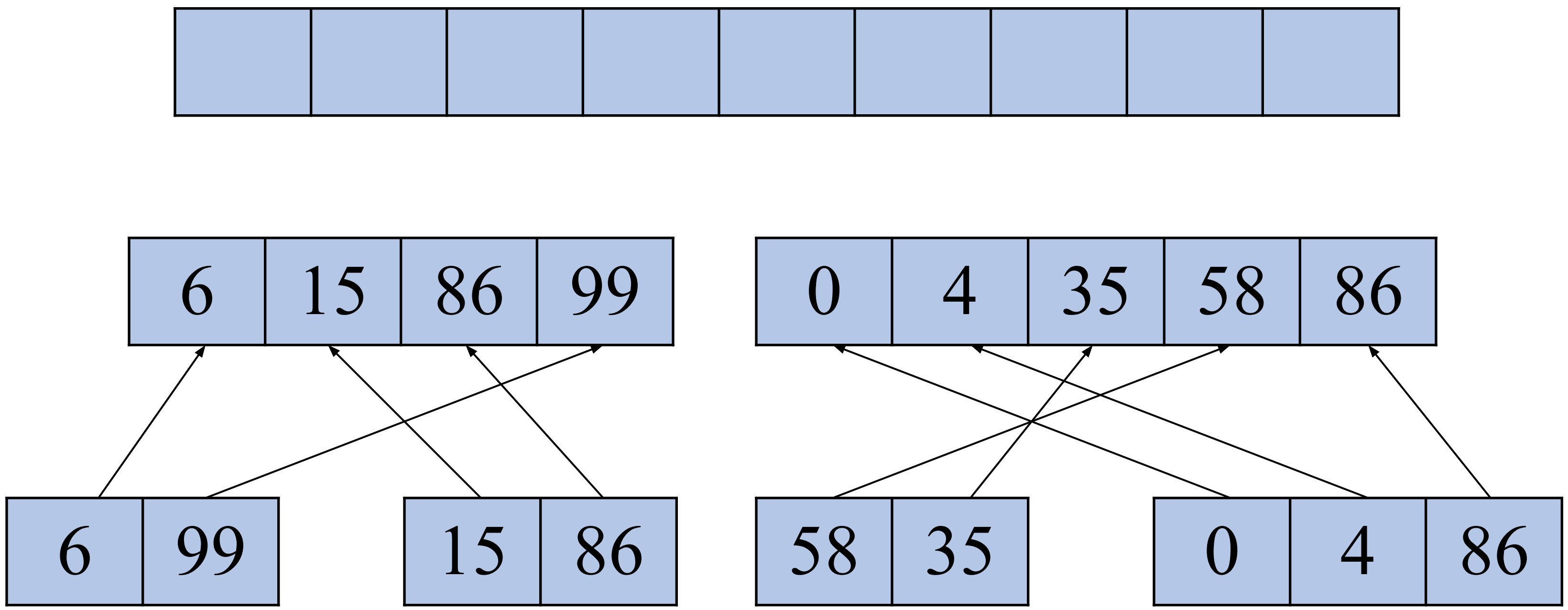
Merge Sort Tree (Example)



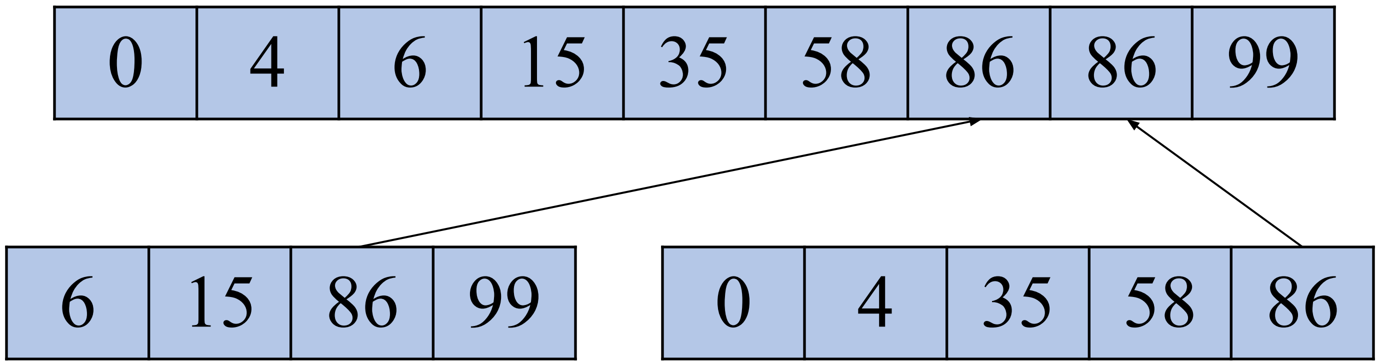
Merge Sort Tree (Example)



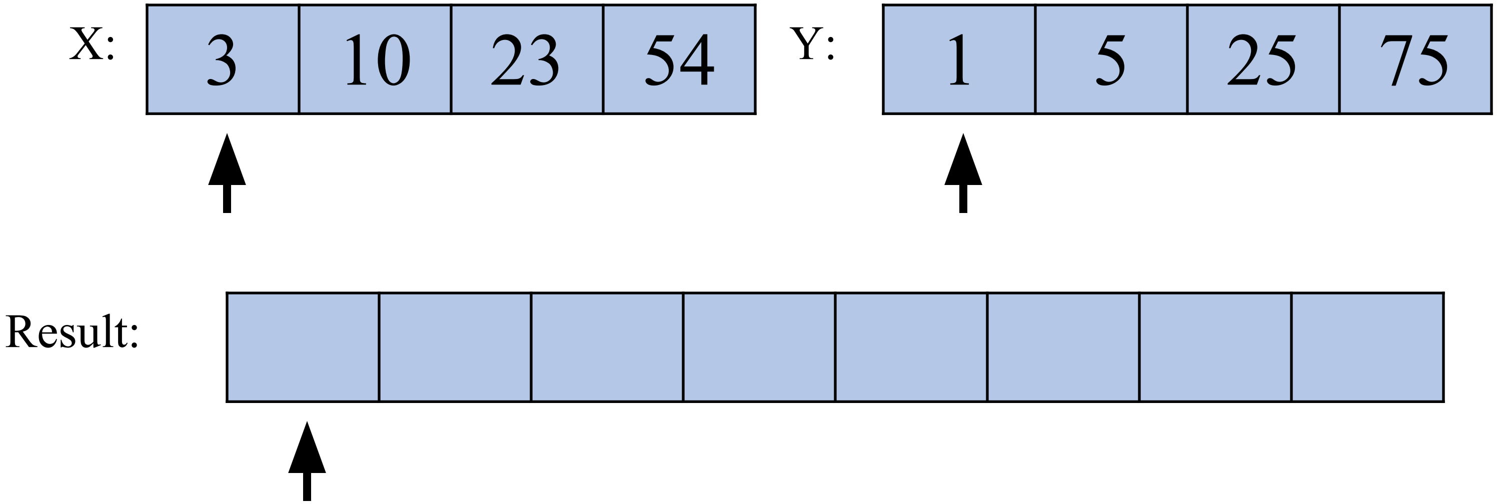
Merge Sort Tree (Example)



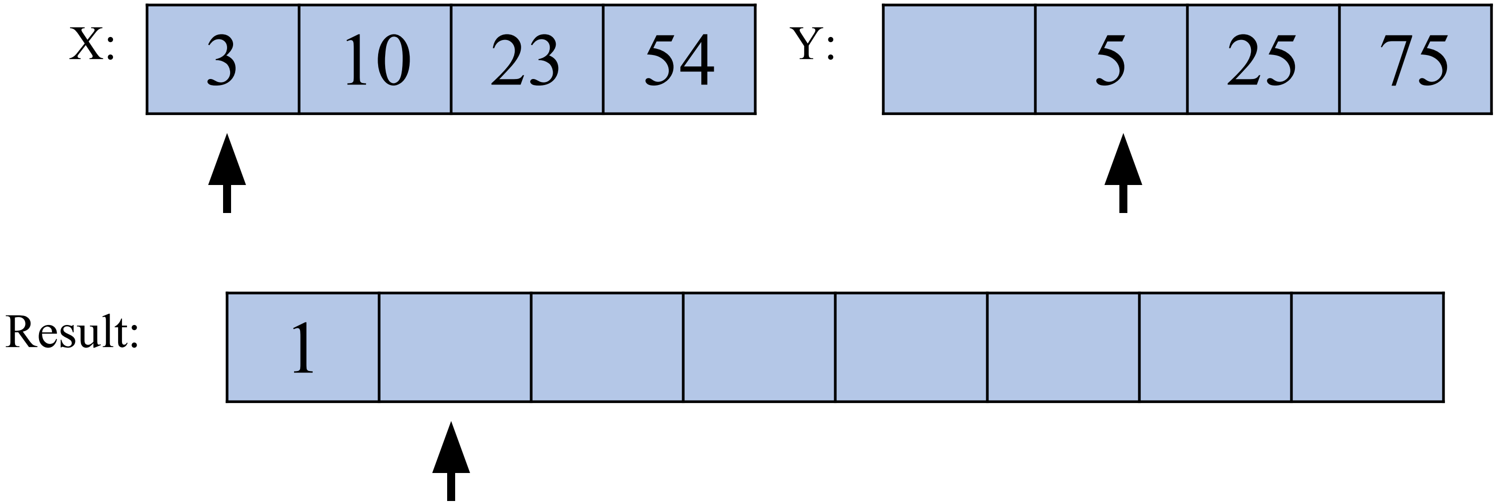
Merge Sort Tree (Example)



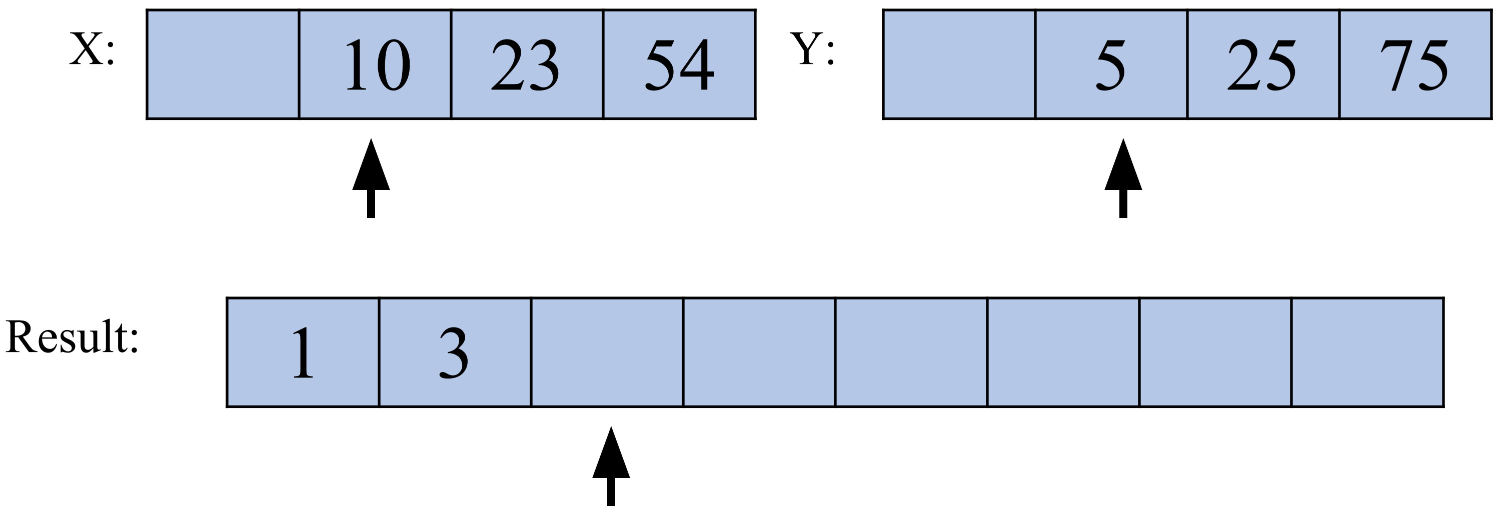
Merge Sort Tree (Merging)



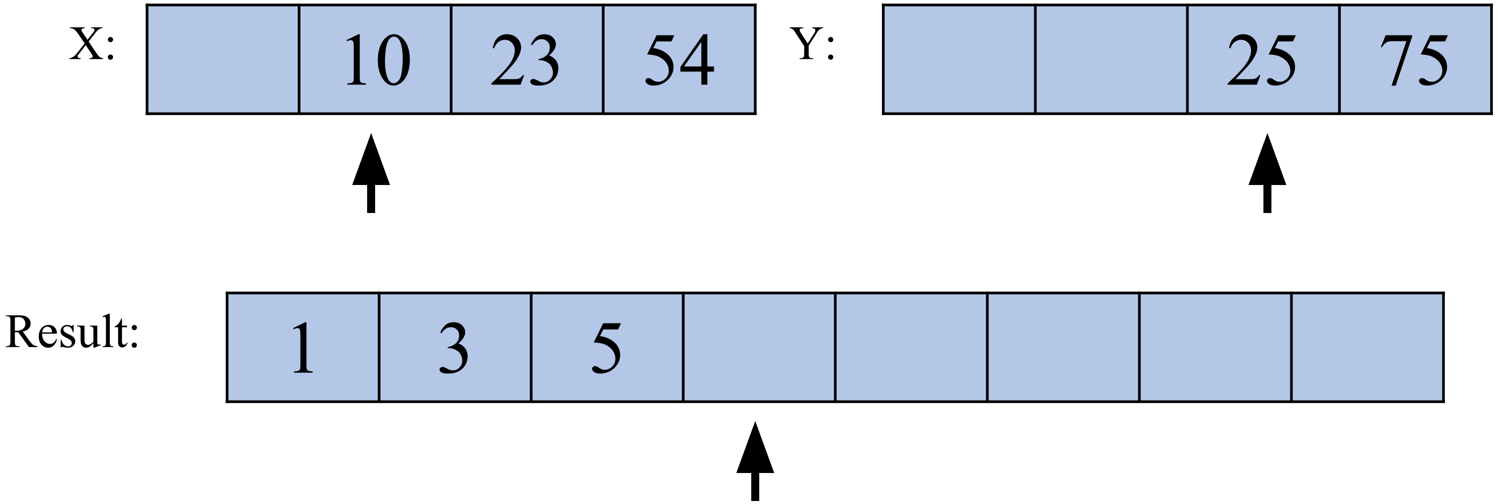
Merge Sort Tree (Merging)



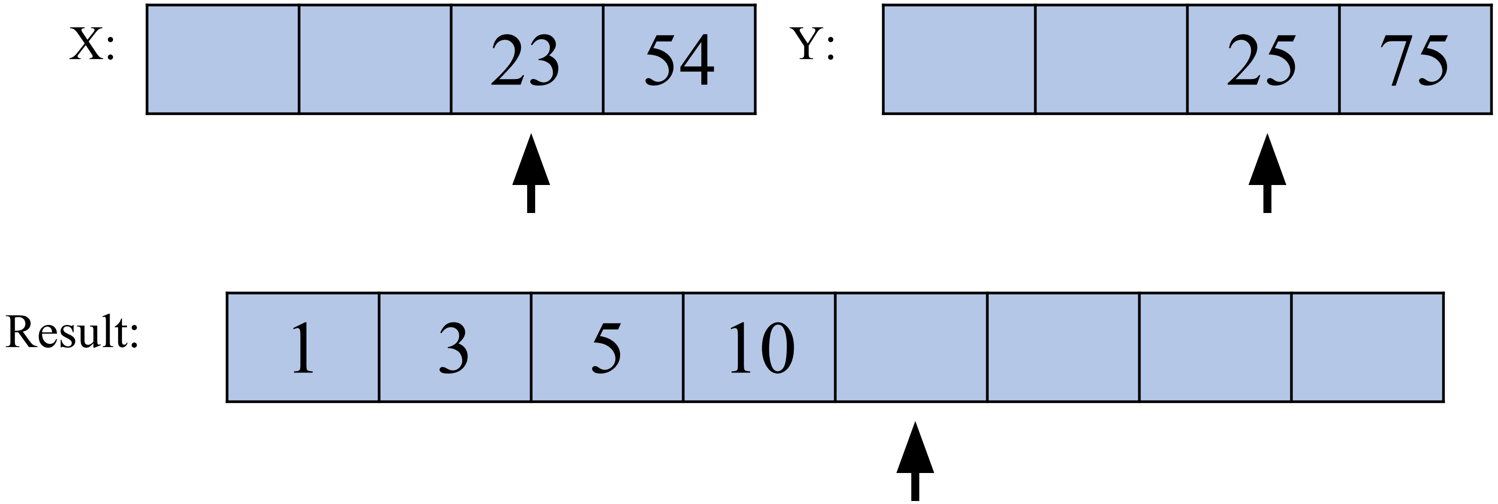
Merge Sort Tree (Merging)



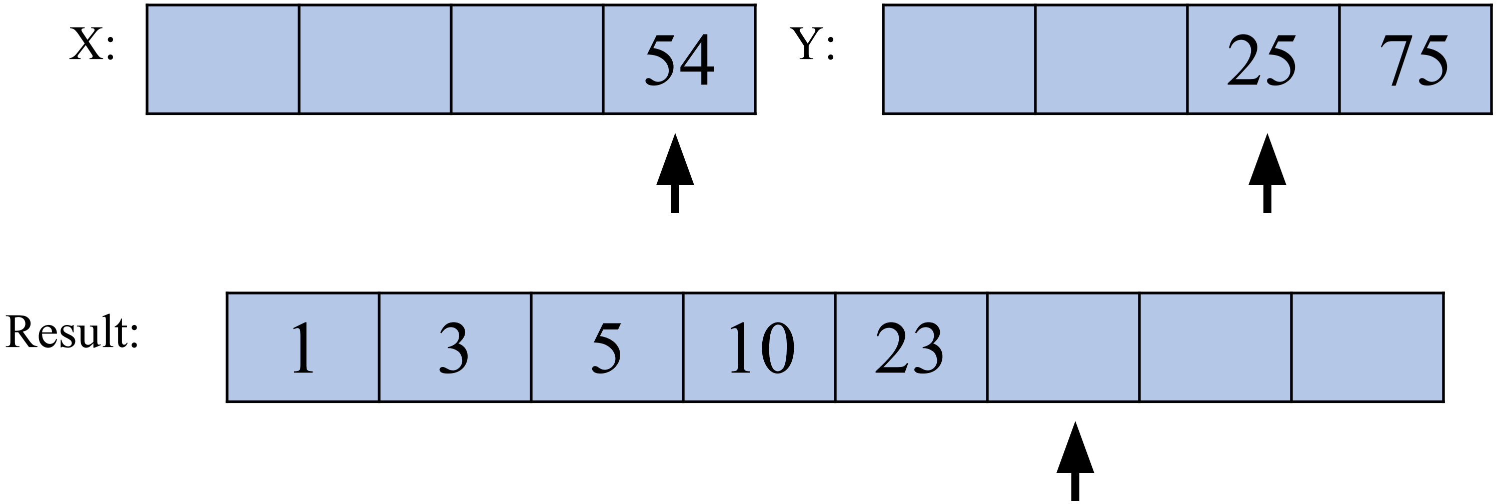
Merge Sort Tree (Merging)



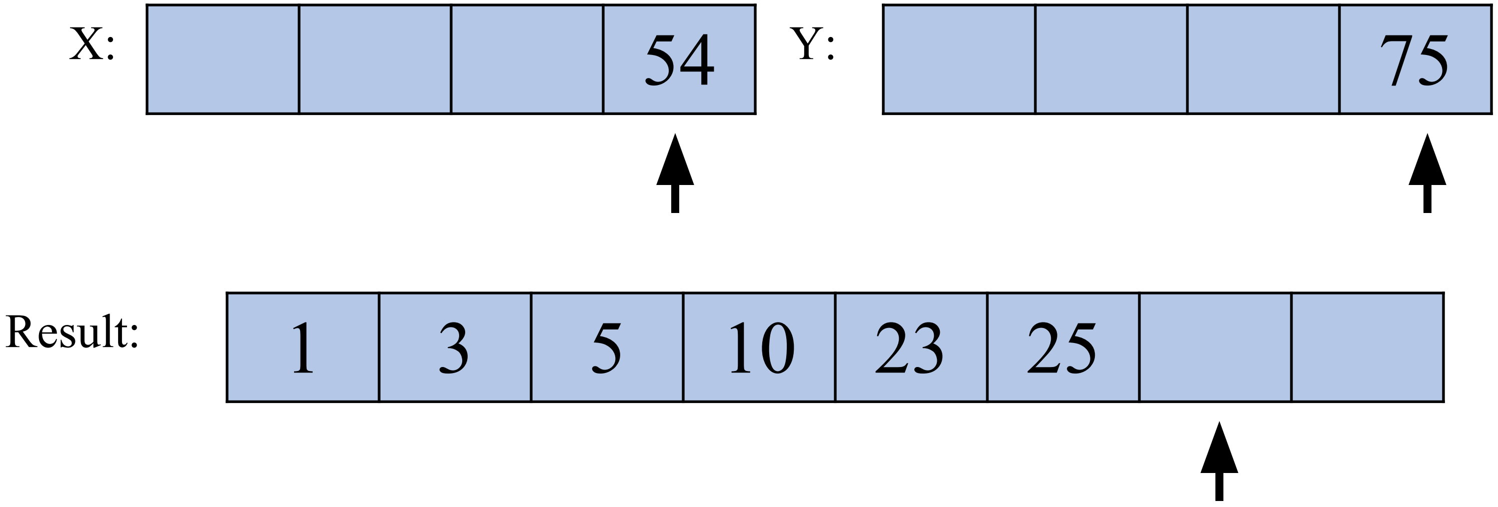
Merge Sort Tree (Merging)



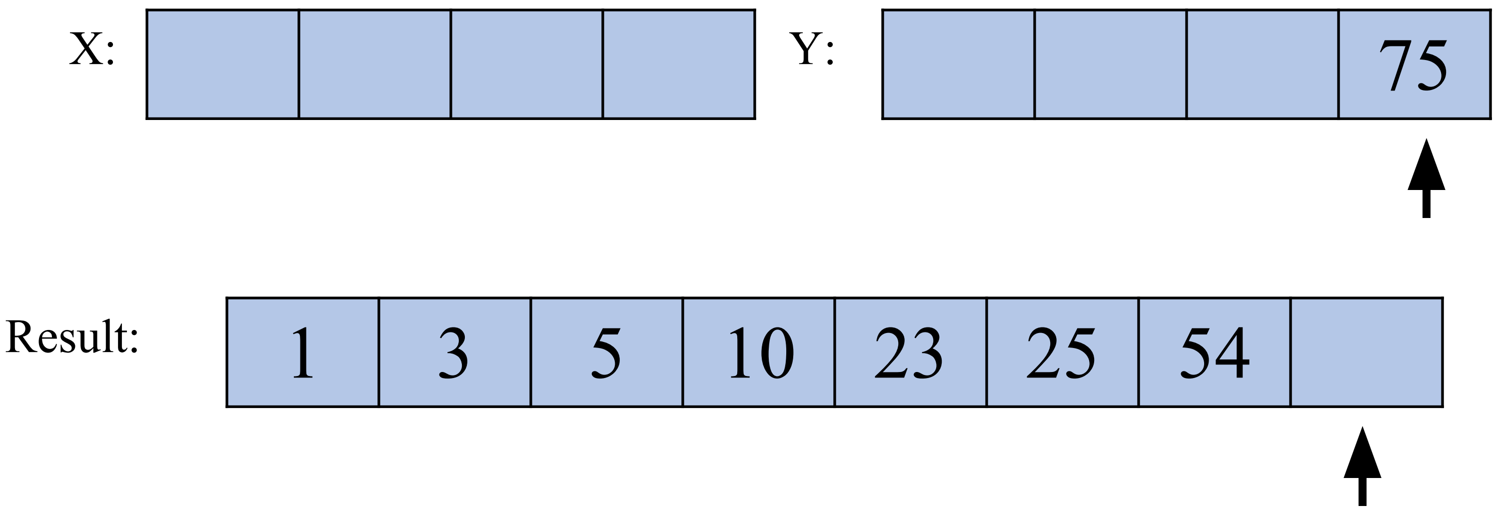
Merge Sort Tree (Merging)



Merge Sort Tree (Merging)



Merge Sort Tree (Merging)



Merge Sort Tree (Merging)

X:

Y:

Result:

1	3	5	10	23	25	54	75

Merge Sort Algorithm

```
MergeSort(array A, int p, int r) {
    if (p<r) {                          // we have at least 2 items
        q = (p + r)/2
        MergeSort (A, p, q)             // sort A[p…q]
        MergeSort (A, q+1, r)           // sort A[q+1…r]
        Merge (A, p ,q, r)              // merge
    }
}

Merge (array A, int p , int q, int r){                    // merges A[p..q] with A[q+1…r]
    array B[p…r]
    i = k = p
    j = q+1
    while (i<=q and j<=r) {                               // while both subarrays are nonempty
        if (A[i] < = A[j])   B[k++] = A[i++]              // copy from left subarray
        else                 B[k++] = A[j++]              // copy from right subarray
    }
    while (i<=q)   B[k++] = A[i++]                        // copy any left over to B
    while (j<=r)   B[k++] = A[j++]
    for i = p to r do A[i] = B[i]                         // copy B back to A
}
```

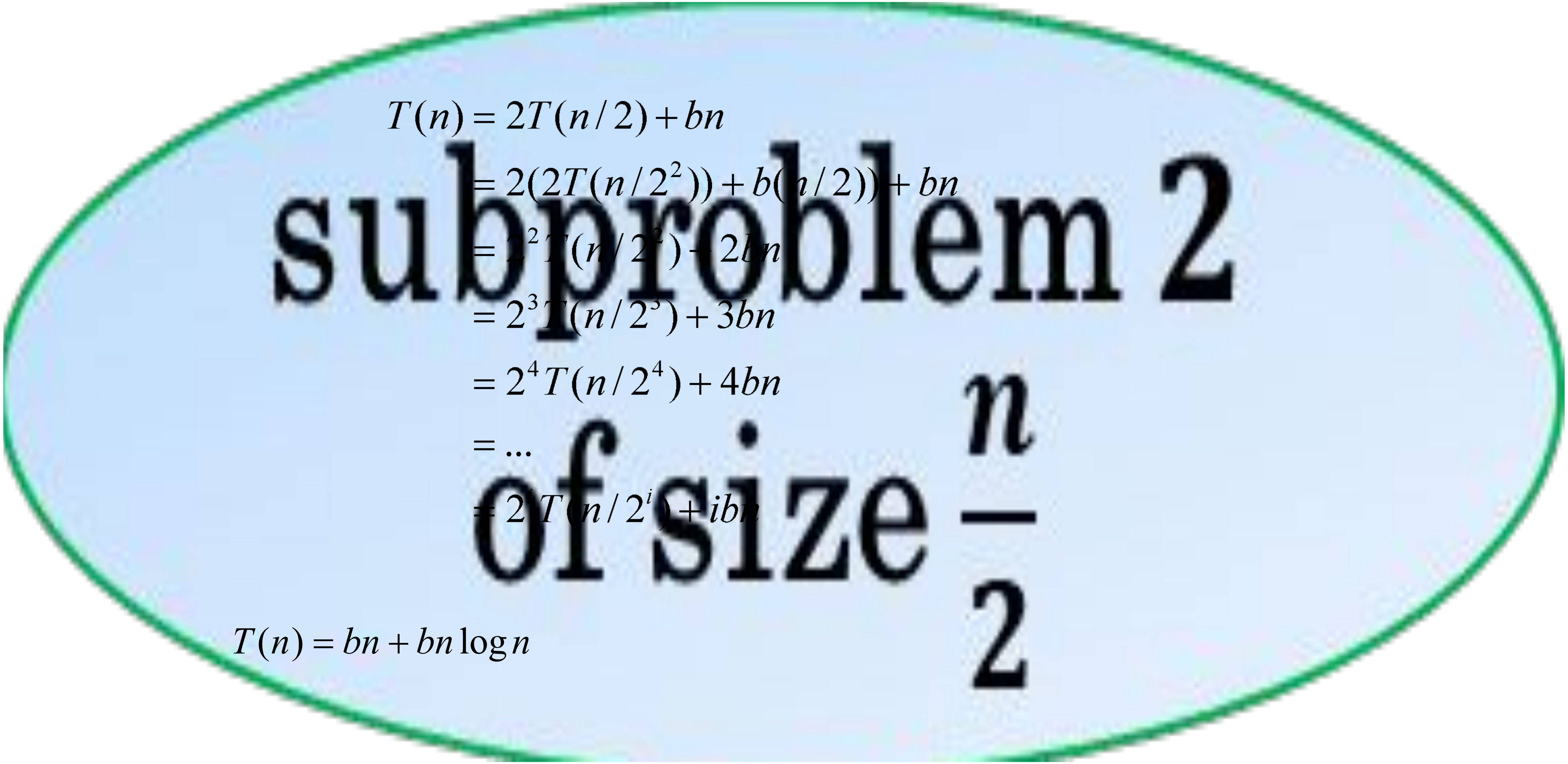
Recurrence Equation Analysis

subproblem 2

of size $\frac{n}{2}$

$$T(n) = \begin{cases} b & \text{if } n < 2 \\ 2T(n/2) + bn & \text{if } n \geq 2 \end{cases}$$

REA (iterative substitution)



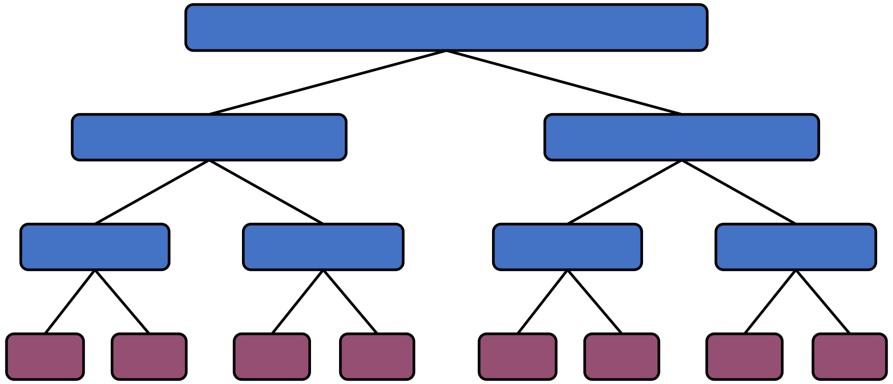
$$
\begin{aligned}
T(n) &= 2T(n/2) + bn \\
&= 2(2T(n/2^2)) + b(n/2)) + bn \\
&= 2^2 T(n/2^2) + 2bn \\
&= 2^3 T(n/2^3) + 3bn \\
&= 2^4 T(n/2^4) + 4bn \\
&= \ldots \\
&= 2^i T(n/2^i) + ibn
\end{aligned}
$$

$$T(n) = bn + bn \log n$$

Analysis of Merge Sort

subproblem 2
of size $\frac{n}{2}$

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



Thank You