Concepts of Algorithms
CS-211
Chapter 6
Divide-and-conquer Algorithm
Merge Sort

Dr. Fethi Fkih
College of Computer
Qassim University

Divide and Conquer

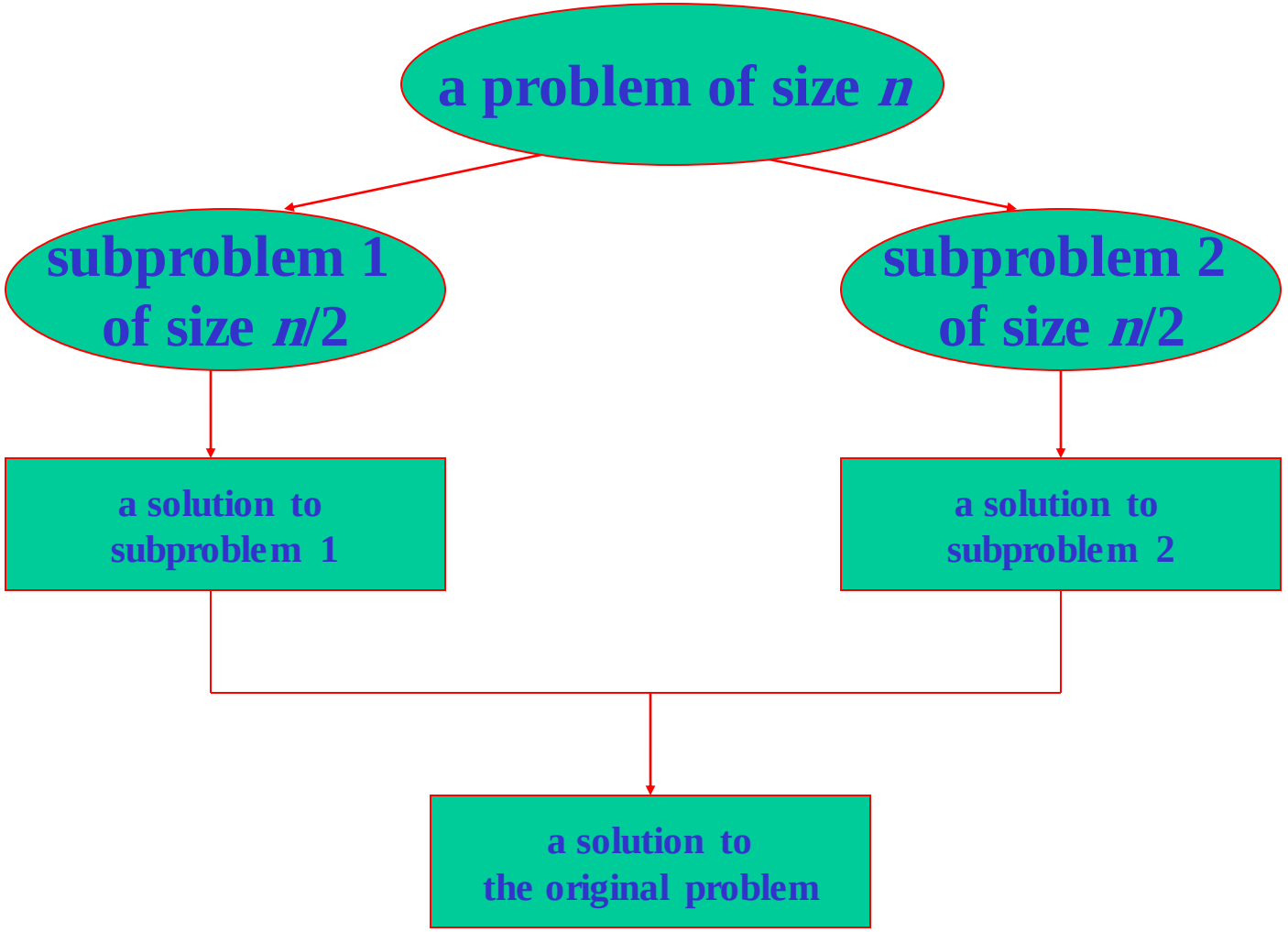
The most well known algorithm design strategy:

1. Divide a large problem into two or more smaller problems

2. Solve smaller problems recursively

3. Obtain solution to original (larger) problem by combining/merging these solutions of small problems.

Divide-and-conquer technique



Divide and Conquer Examples

- **Searching:** binary search
- **Sorting:** mergesort
- **Sorting:** quicksort

Divide and Conquer

1. Base case/criteria: the problem is small enough, solve directly

2. Divide the problem into two or more similar and smaller subproblems

3. Recursively solve the subproblems

4. Combine solutions to the subproblems

Divide and Conquer - Sort

Problem:

- Input: A[n] – **unsorted** array of **n ≥1** integers.
- Output: A[n] – **sorted** in non-decreasing order

Divide and Conquer - Sort

- Base case

  single element (n=1), return

- Divide A into two subarrays: FirstPart, SecondPart

  Two Subproblems:

  - sort the FirstPart
  - sort the SecondPart

- Recursively

  - sort FirstPart
  - sort SecondPart

- Combine sorted FirstPart and sorted second part

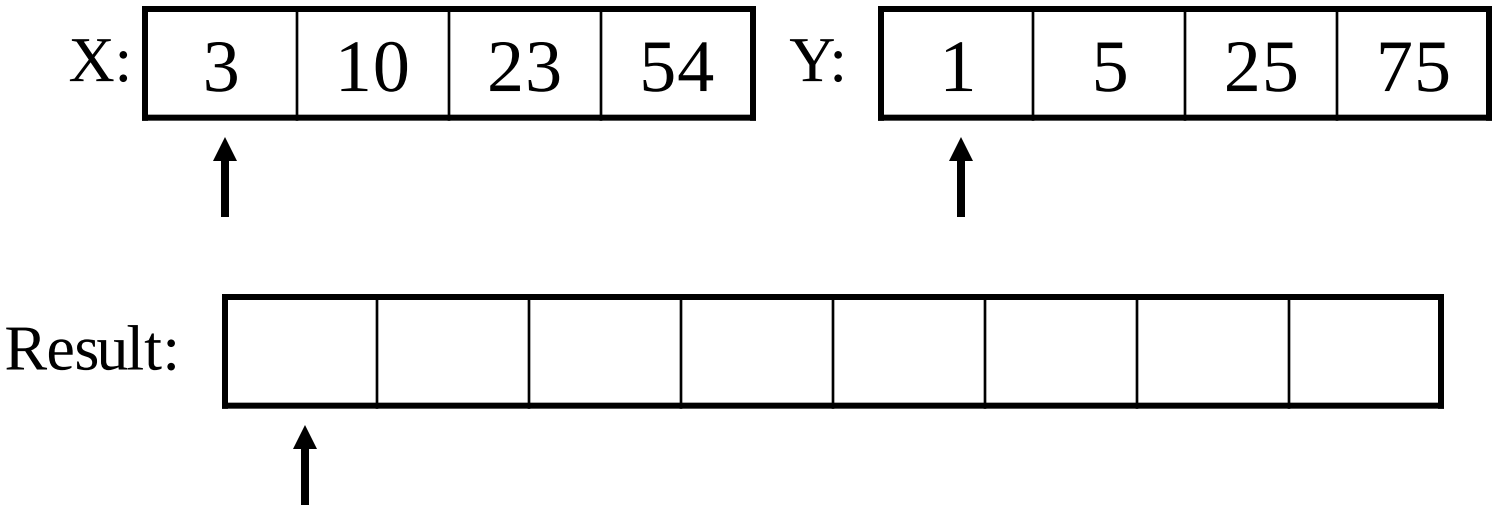
Merge Sort

Merging

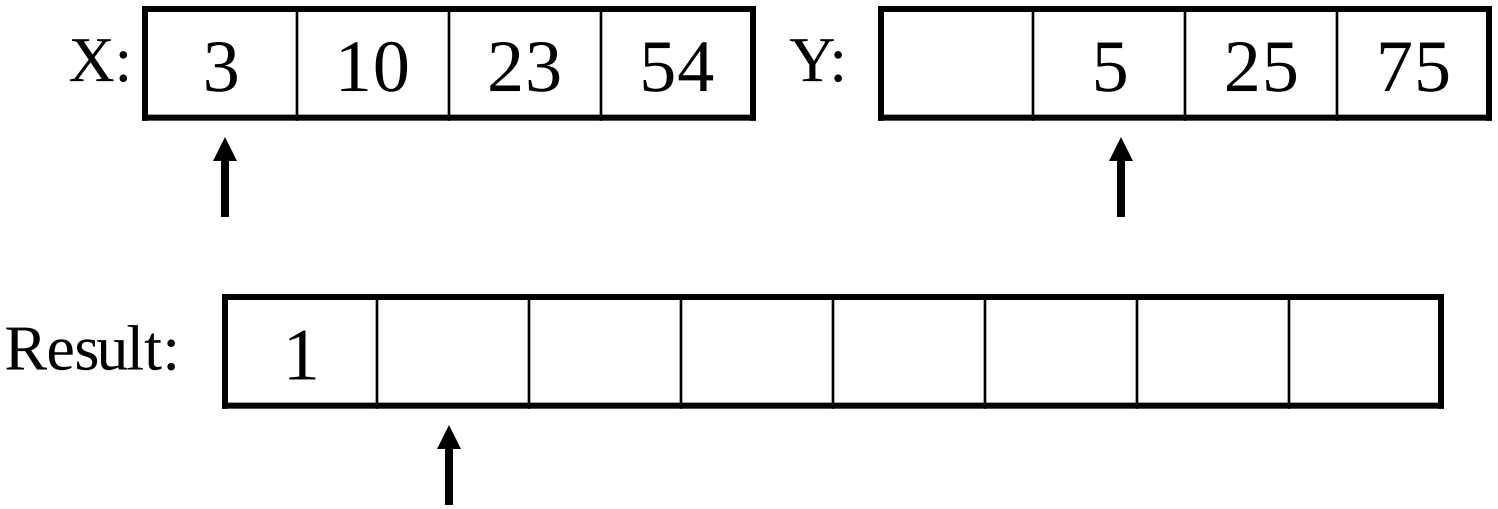
Suppose **A** is sorted list with **r** elements and **B** is a sorted list with **s** elements. The operation that combine the elements of **A** and **B** into a single sorted list **C** with **n= r+s** elements is called merging. One simple way to merge is to place the elements of **B** after the elements of **A** and then use some sorting algorithm on the entire list. This method does not take advantage of the fact that **A** and **B** are individually sorted. A much more efficient algorithm is merge sort algorithm.

Suppose one is given two sorted decks of cards. The decks are merged as:

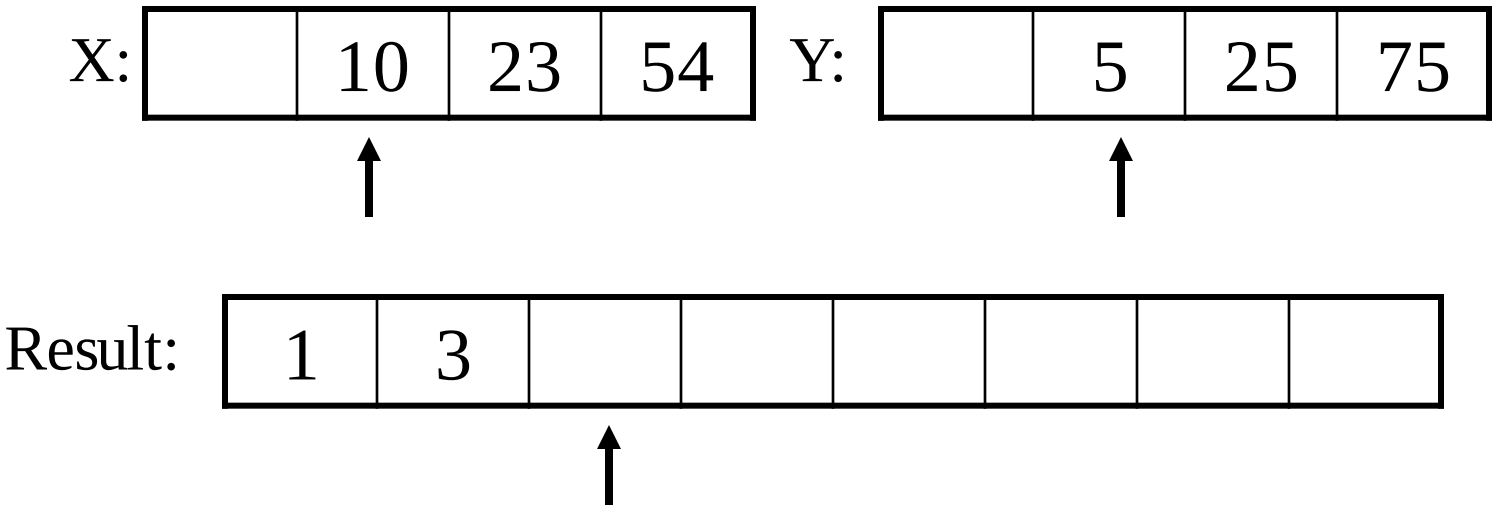
Merging (cont.)



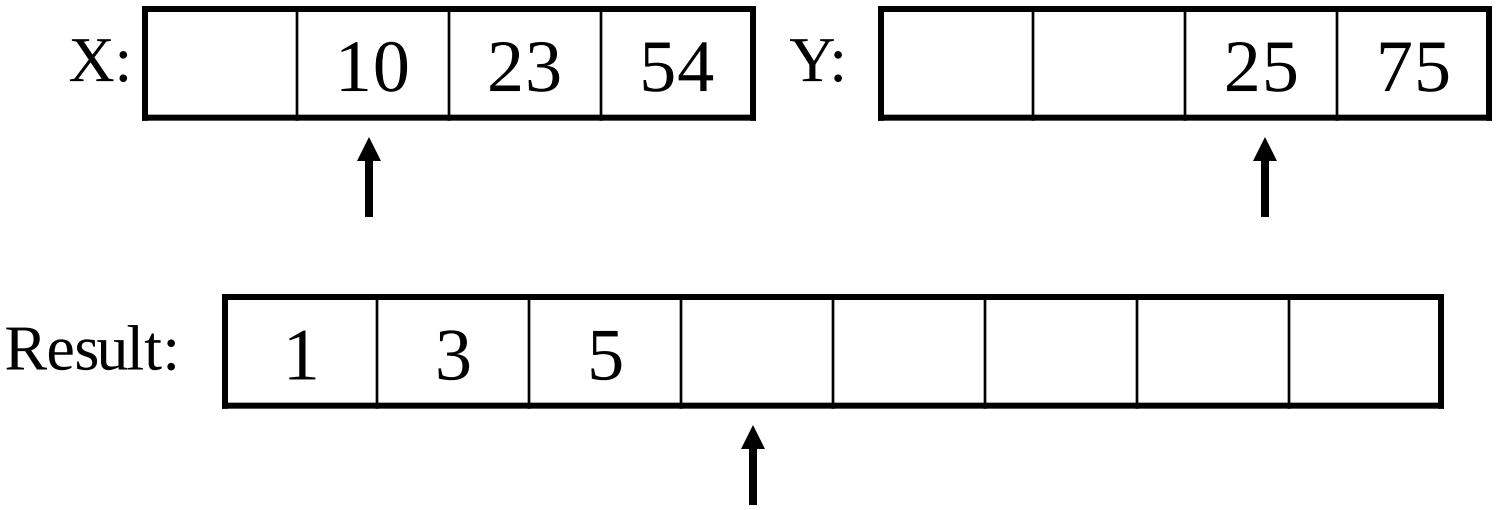
Merging (cont.)



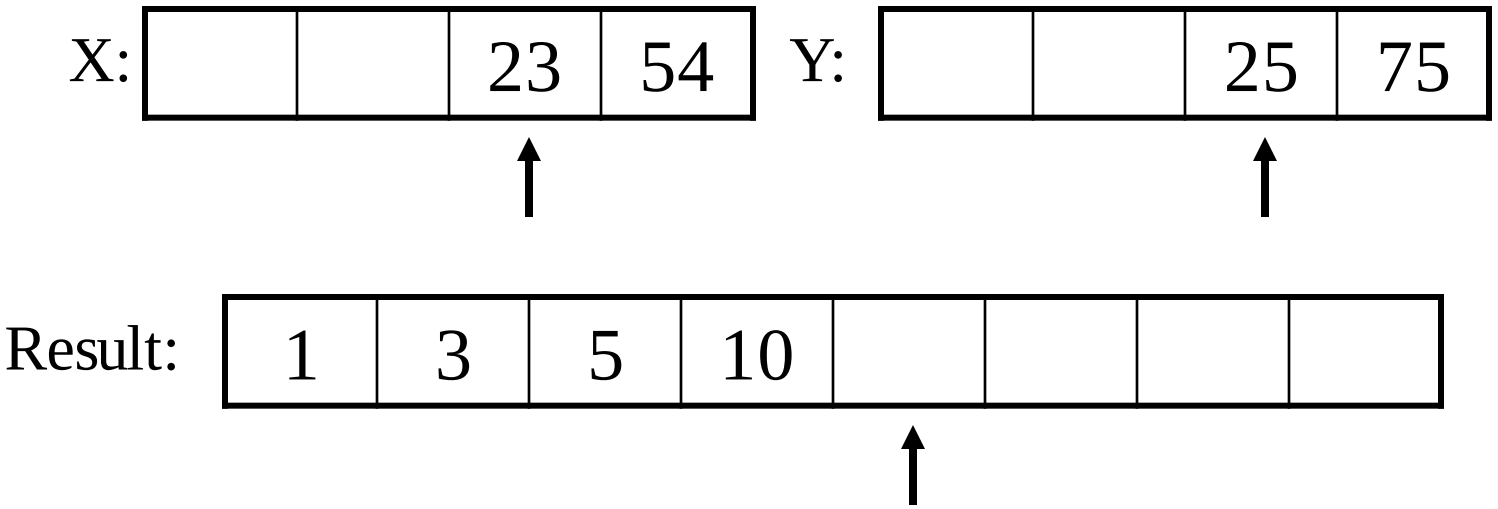
Merging (cont.)



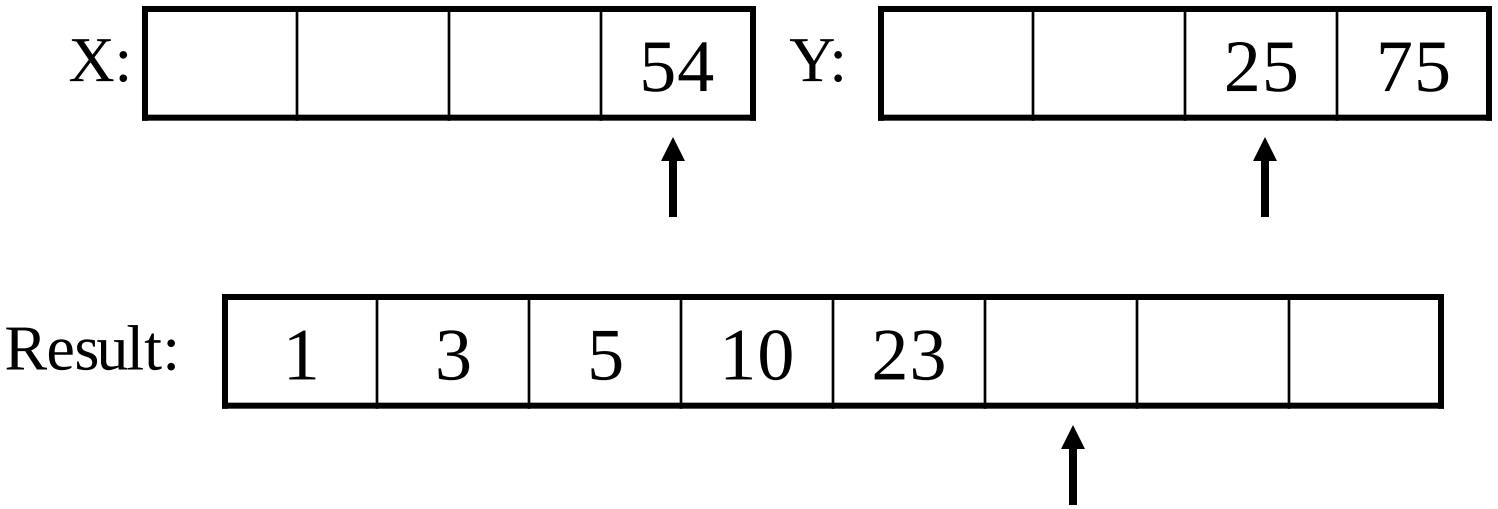
Merging (cont.)



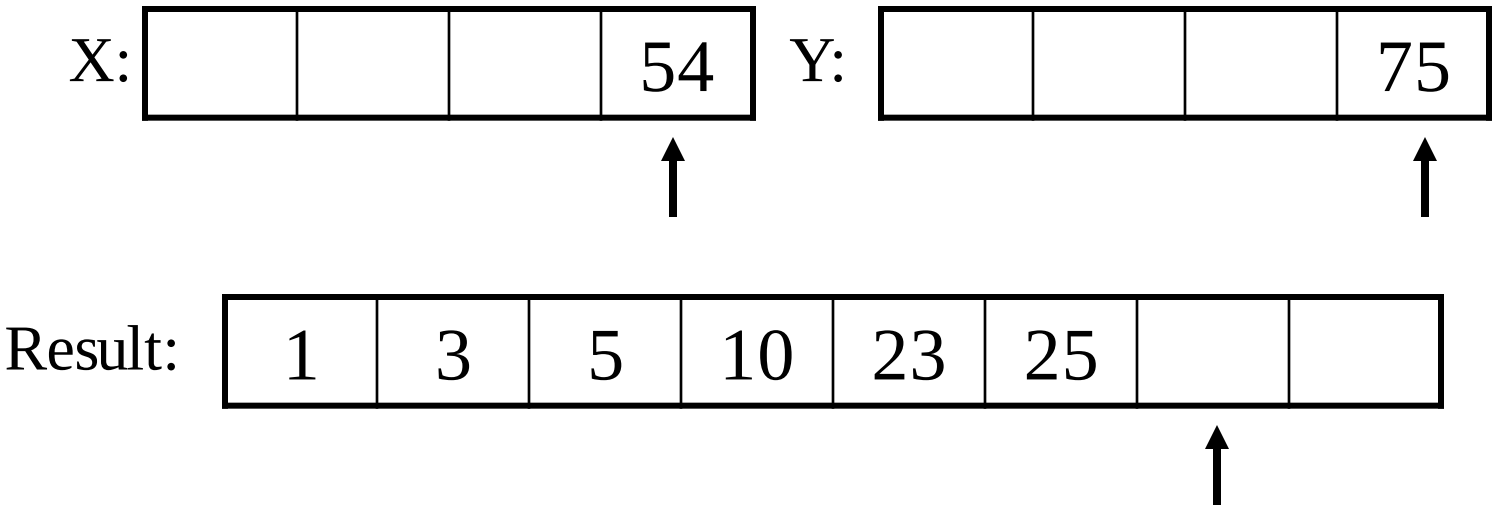
Merging (cont.)



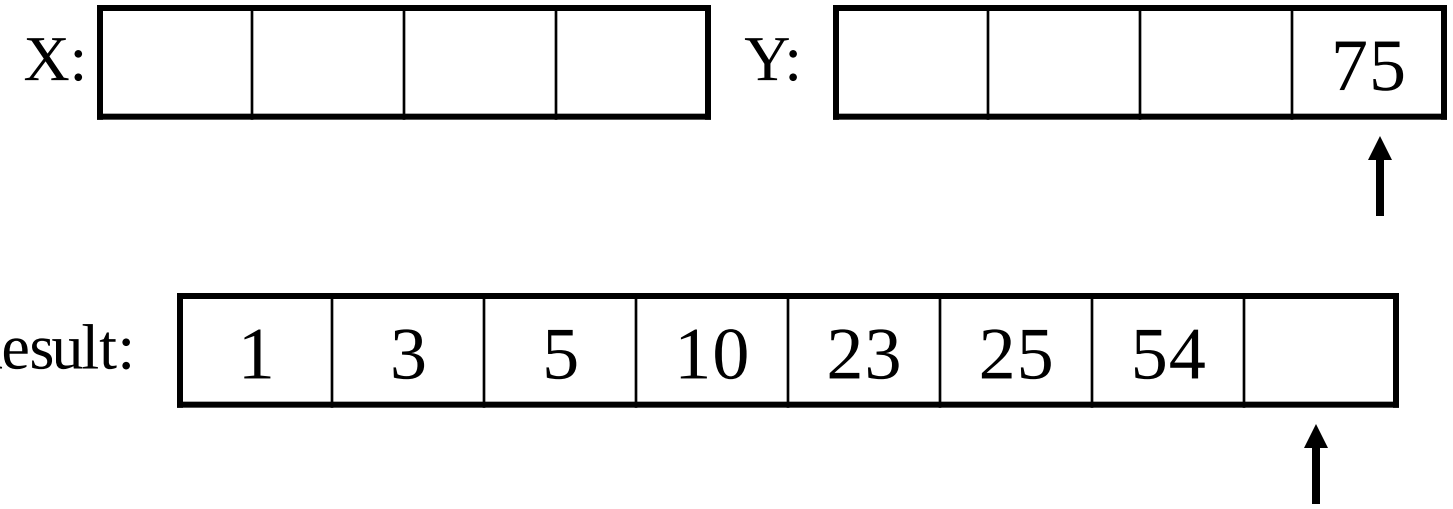
Merging (cont.)



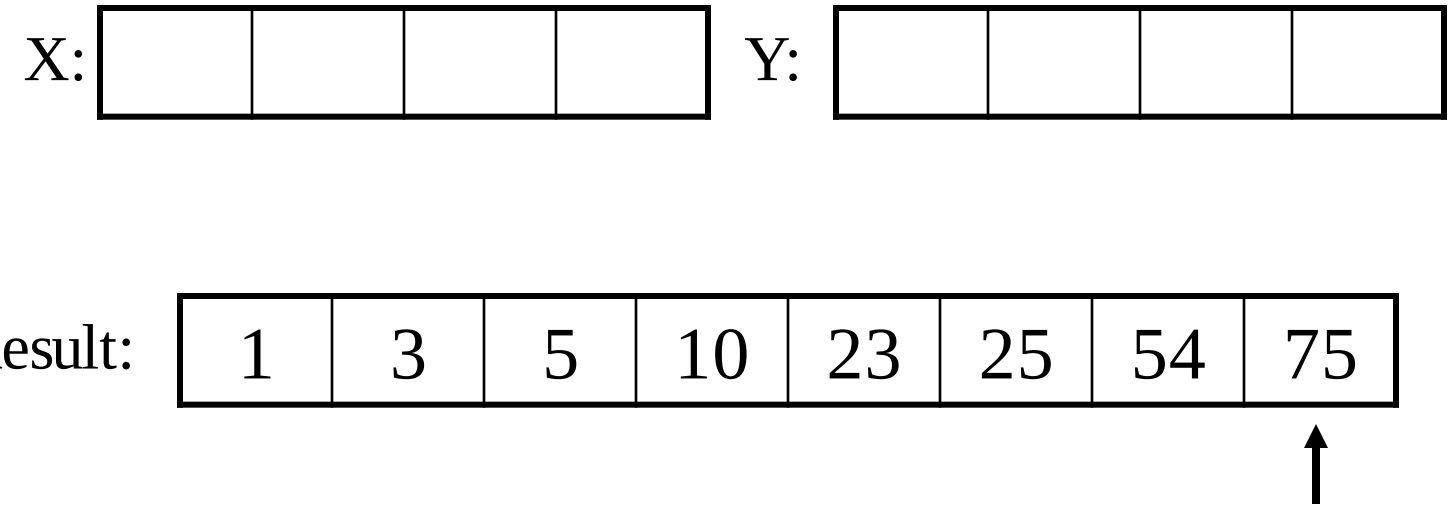
Merging (cont.)



Merging (cont.)



Merging (cont.)



Merge Sort Example

99	6	86	15	58	35	86	4	0

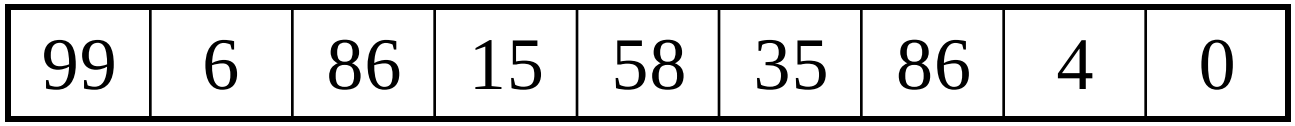
Merge Sort Example

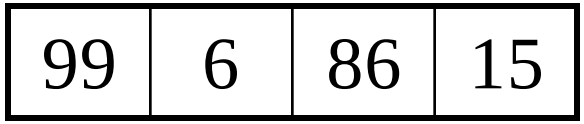
99	6	86	15	58	35	86	4	0

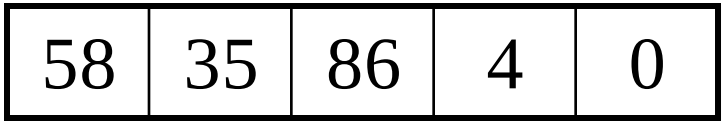
99	6	86	15

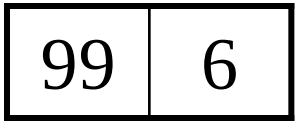
58	35	86	4	0

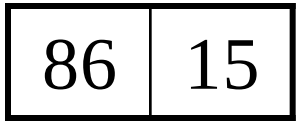
Merge Sort Example

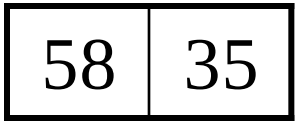






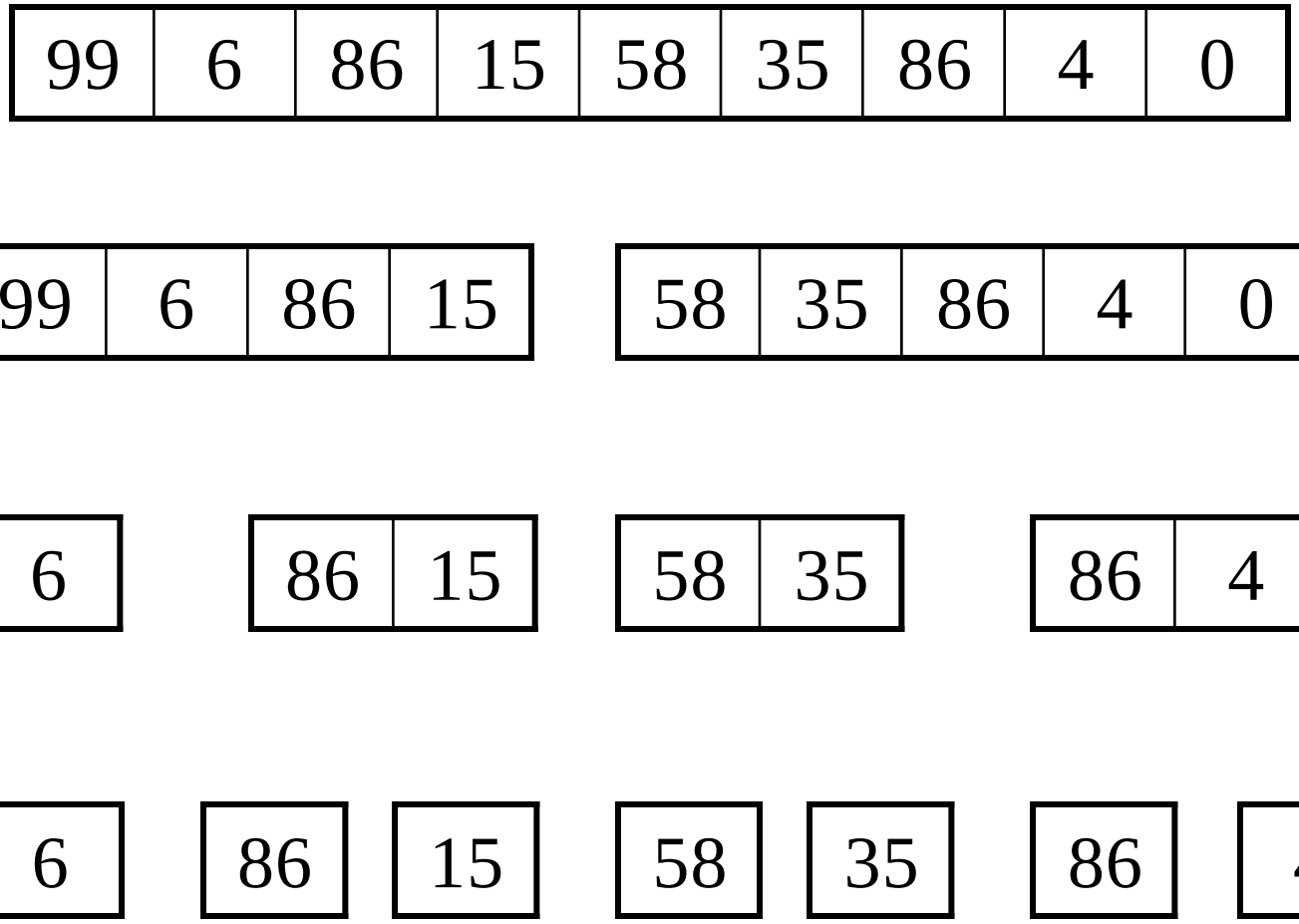




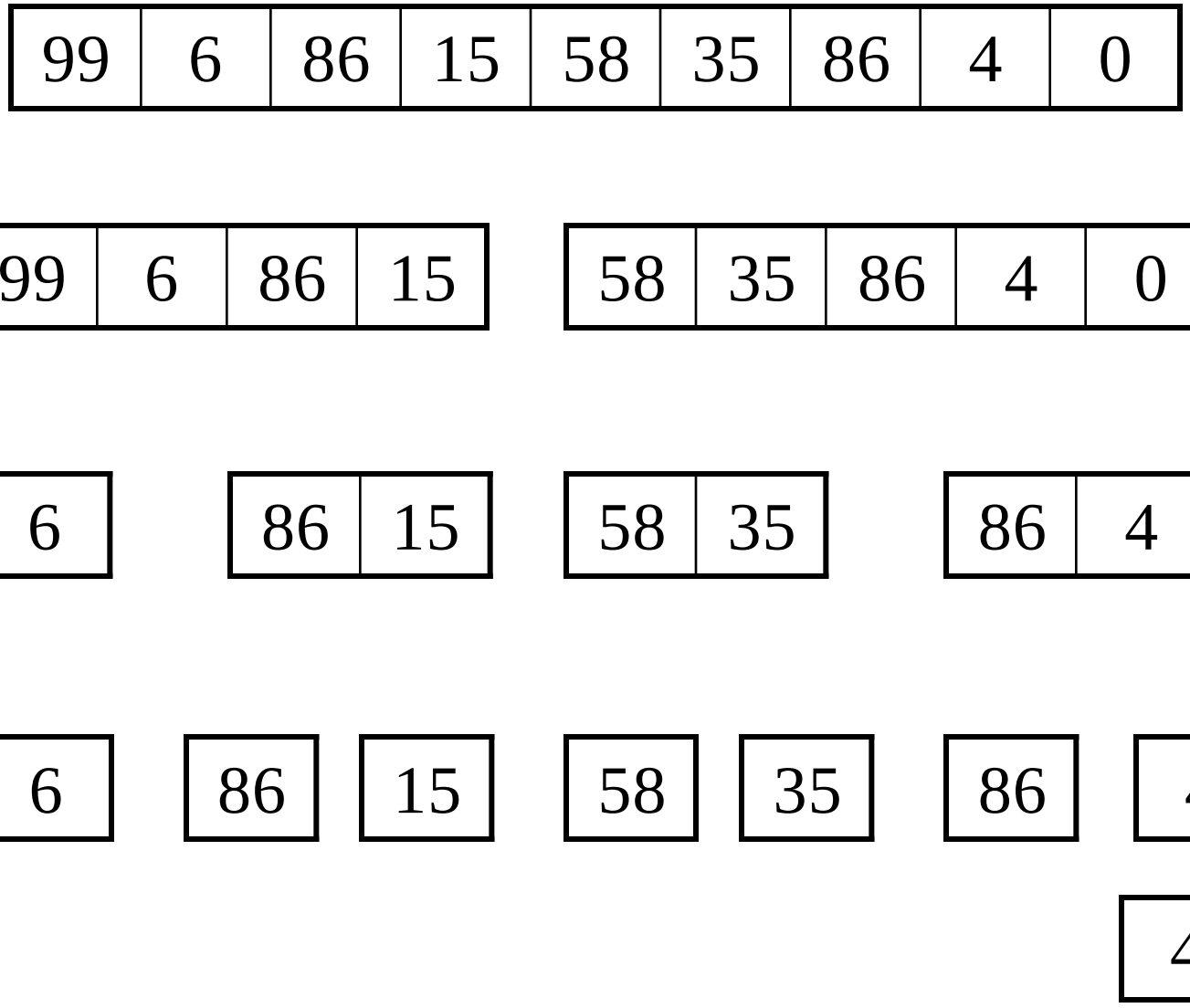




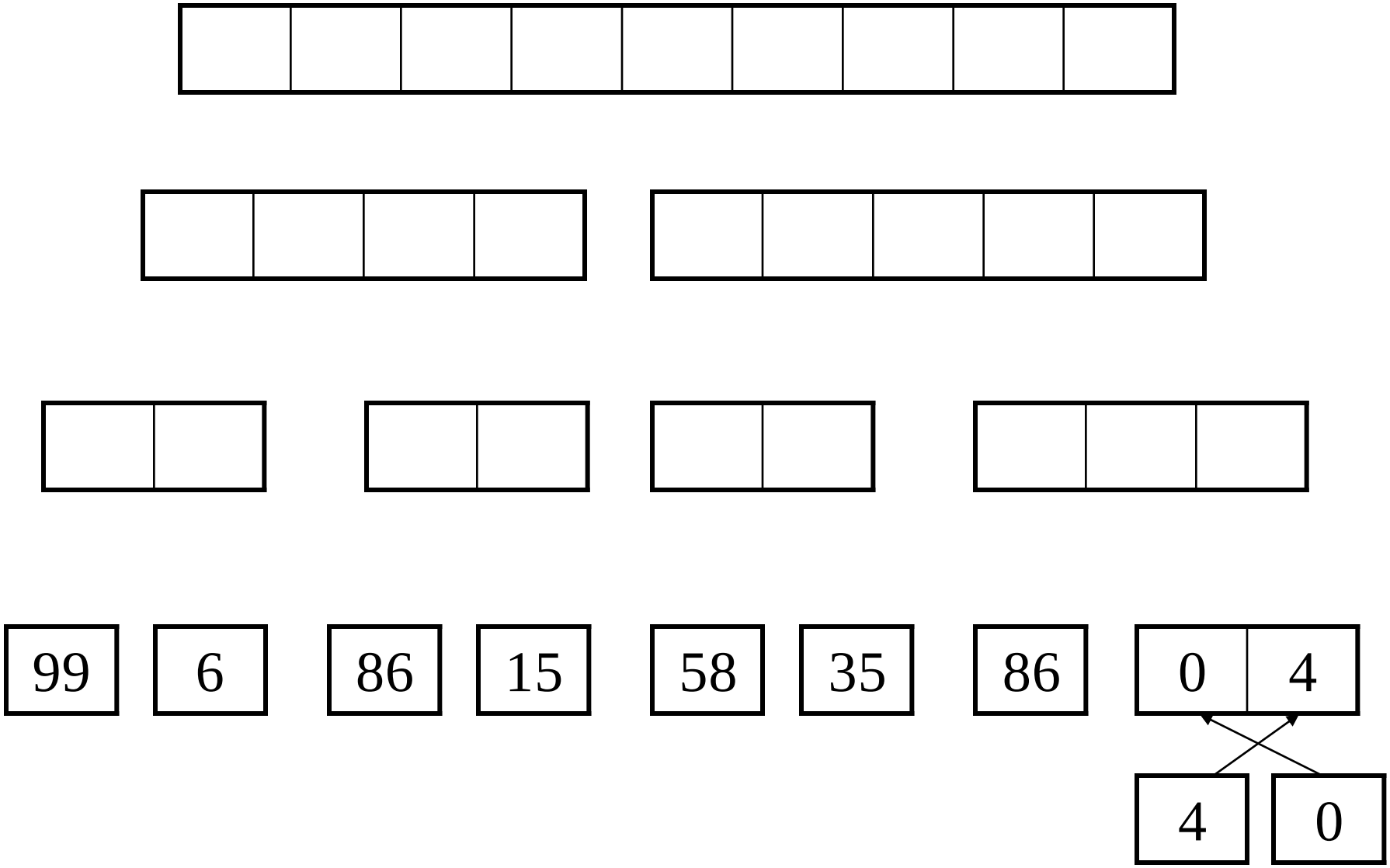
Merge Sort Example



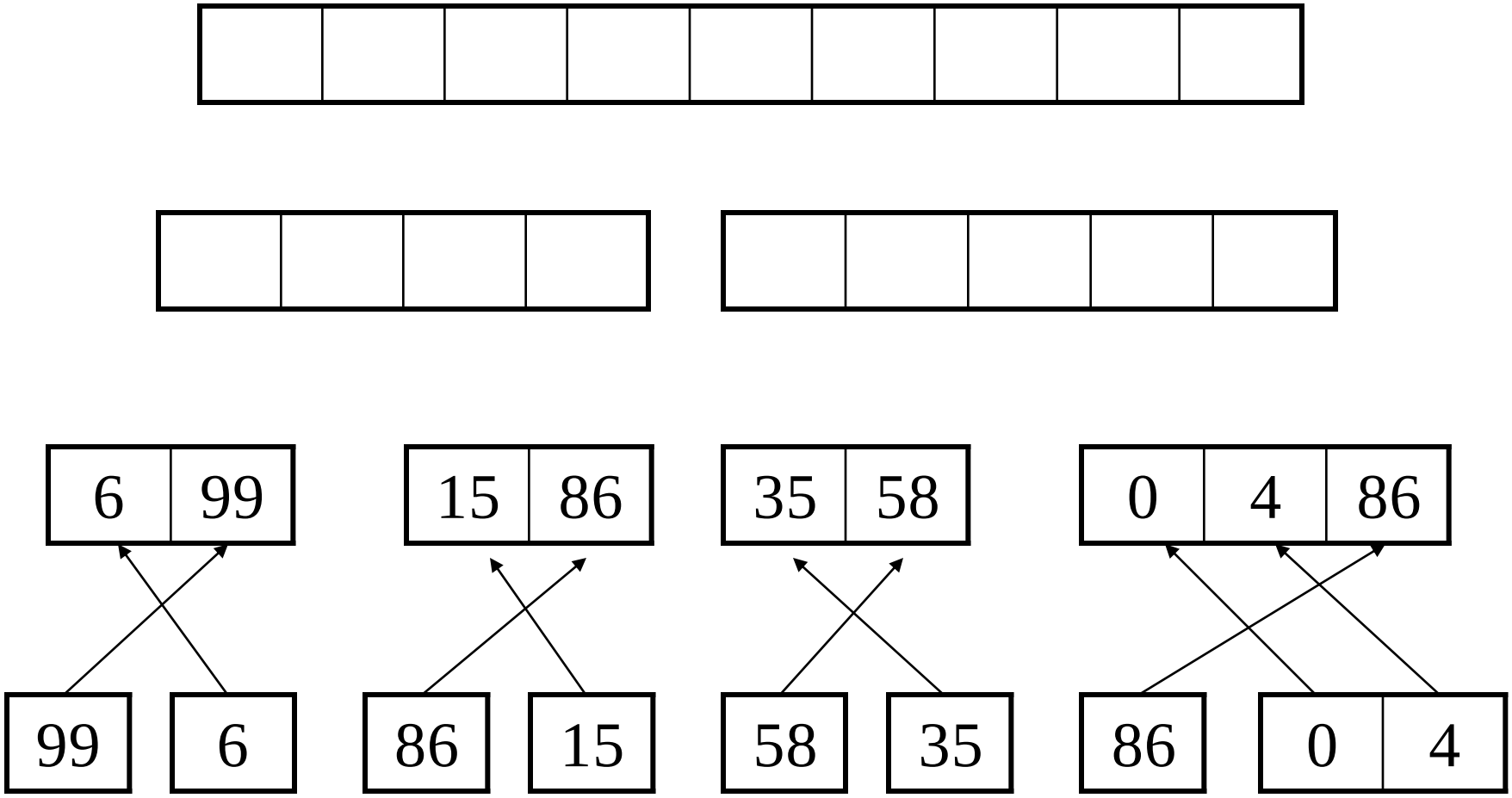
Merge Sort Example



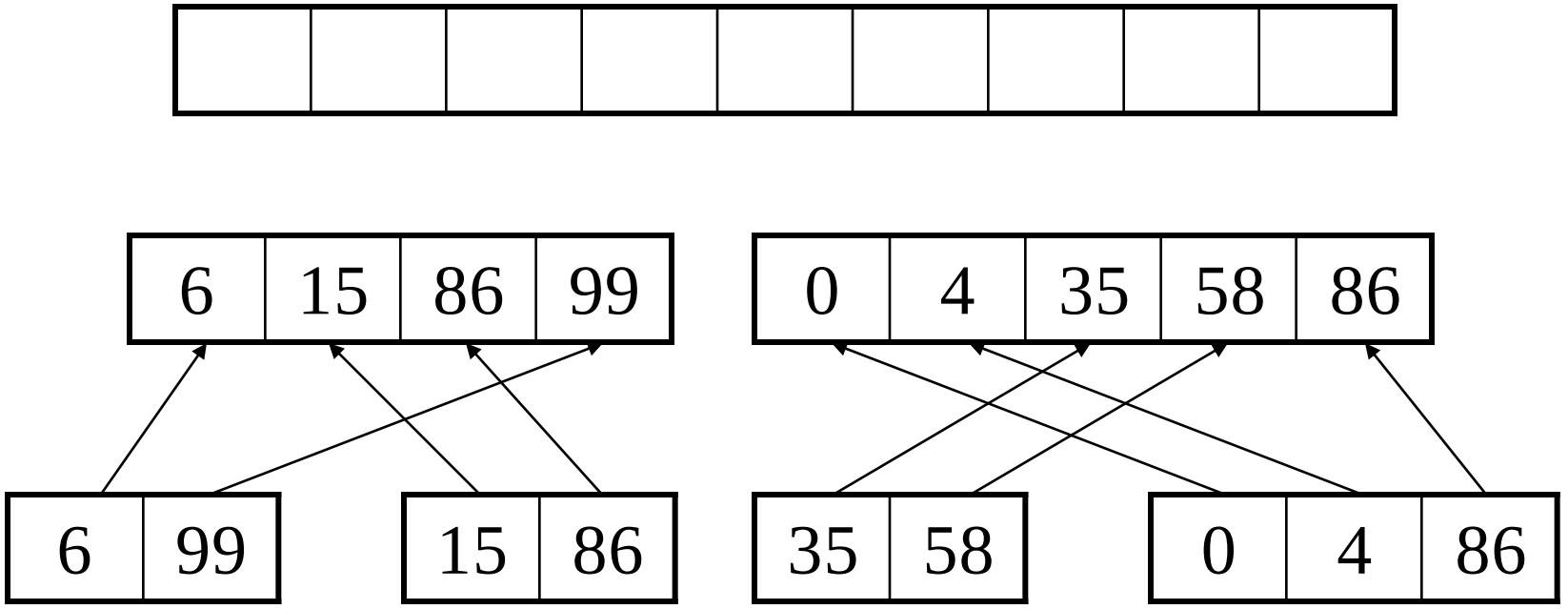
Merge Sort Example



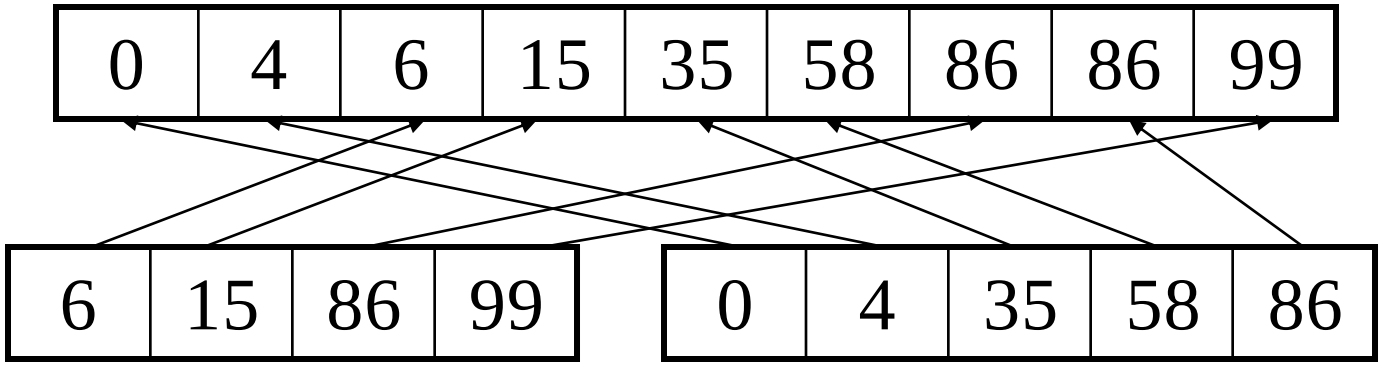
Merge Sort Example



Merge Sort Example



Merge Sort Example



Merge Sort Example

0	4	6	15	35	58	86	86	99

Another Example MergeSort

Original	24	13	26	1	12	27	38	15							
Divide in 2	24	13	26	1		12	27	38	15						
Divide in 4	24	13		26	1		12	27		38	15				
Divide in 8	24		13		26		1		12		27		38		15
Merge 2	13	24			1	26			12	27			15	38	
Merge 4	1	13	24	26					12	15	27	38			
Merge 8	1	12	13	15	24	26	27	38							

Merge Sort: Algorithm

Algorithm: MERGESORT (A, N)

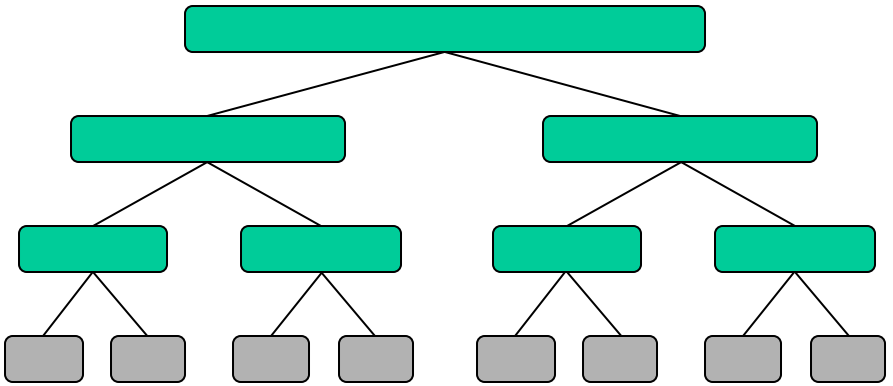
1- If N=1, Return.
2- N1=N/2, N2=N-N1.
3- for i=0,1,2 . . . . . (N1-1)
L [i]=A [i].
4- for j=0,1,2 . . . . . (N2-1)
R [j]=A [N1+j].
5- Call MERGESORT (L, N1).
6 - Call MERGESORT (R, N2).
7- Call MERGE (A, L, N1, R,N2).
8- Return.

Analysis of Merge-Sort

- The height h of the merge-sort tree is $O(\log n)$
  - at each recursive call we divide in half the sequence,
- The overall amount or work done at the nodes of depth i is $O(n)$
  - we partition and merge 2^i sequences of size $n/2^i$

- Thus, the total running time of merge-sort is $O(n \log n)$

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



https://www.youtube.com/watch?v=MBxqi6GrfN8

END