Data Structure
Algorithms & Applications

CT-159

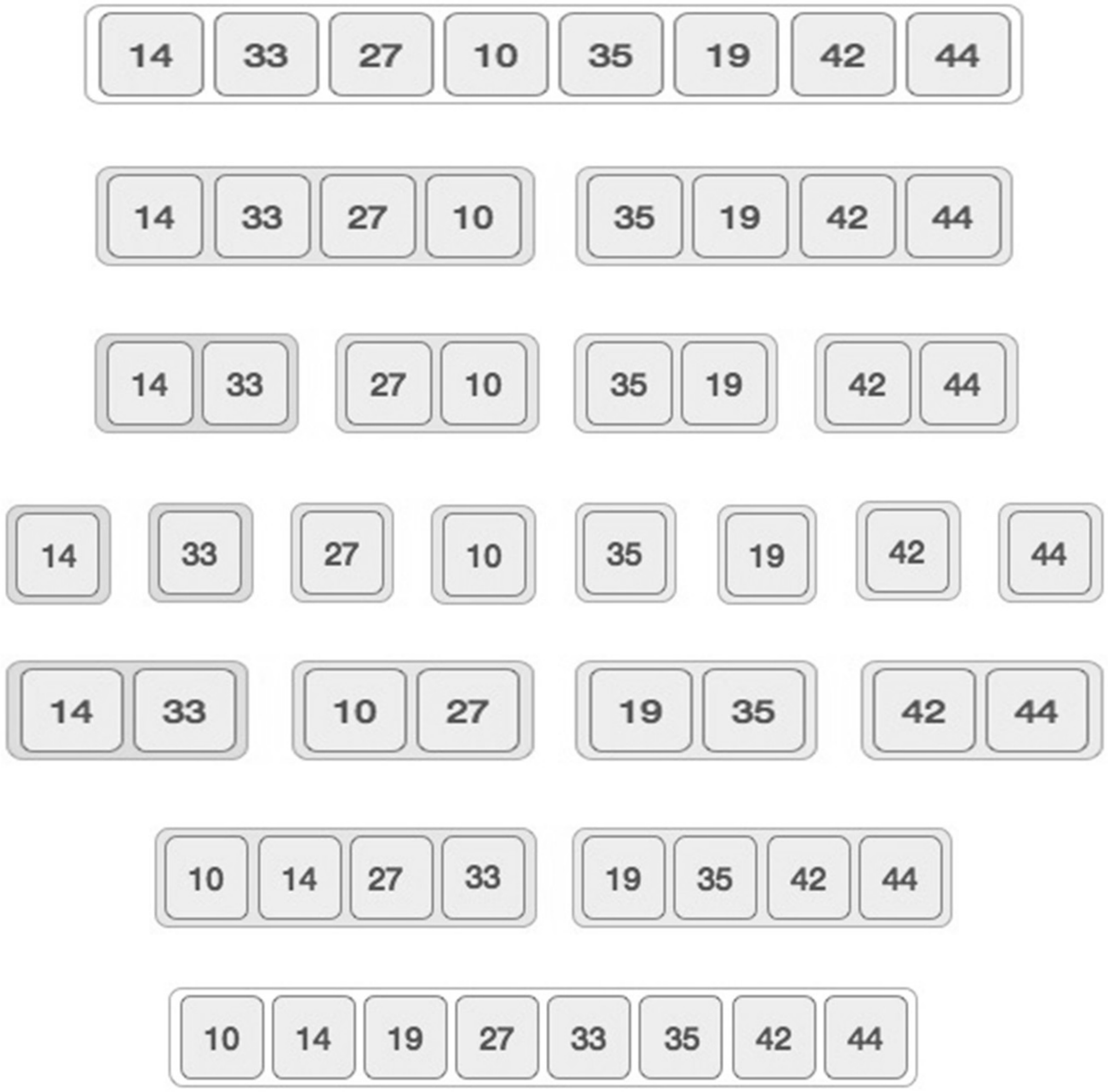
Prepared by

Muhammad Kamran

Merge Sort

- Merge sort is a sorting technique based on divide and conquer technique.
- With worst-case time complexity being O(n log n), it is one of the most respected algorithms.
- Merge sort first divides the array into equal halves and then combines them in a sorted manner.

Merge Sort

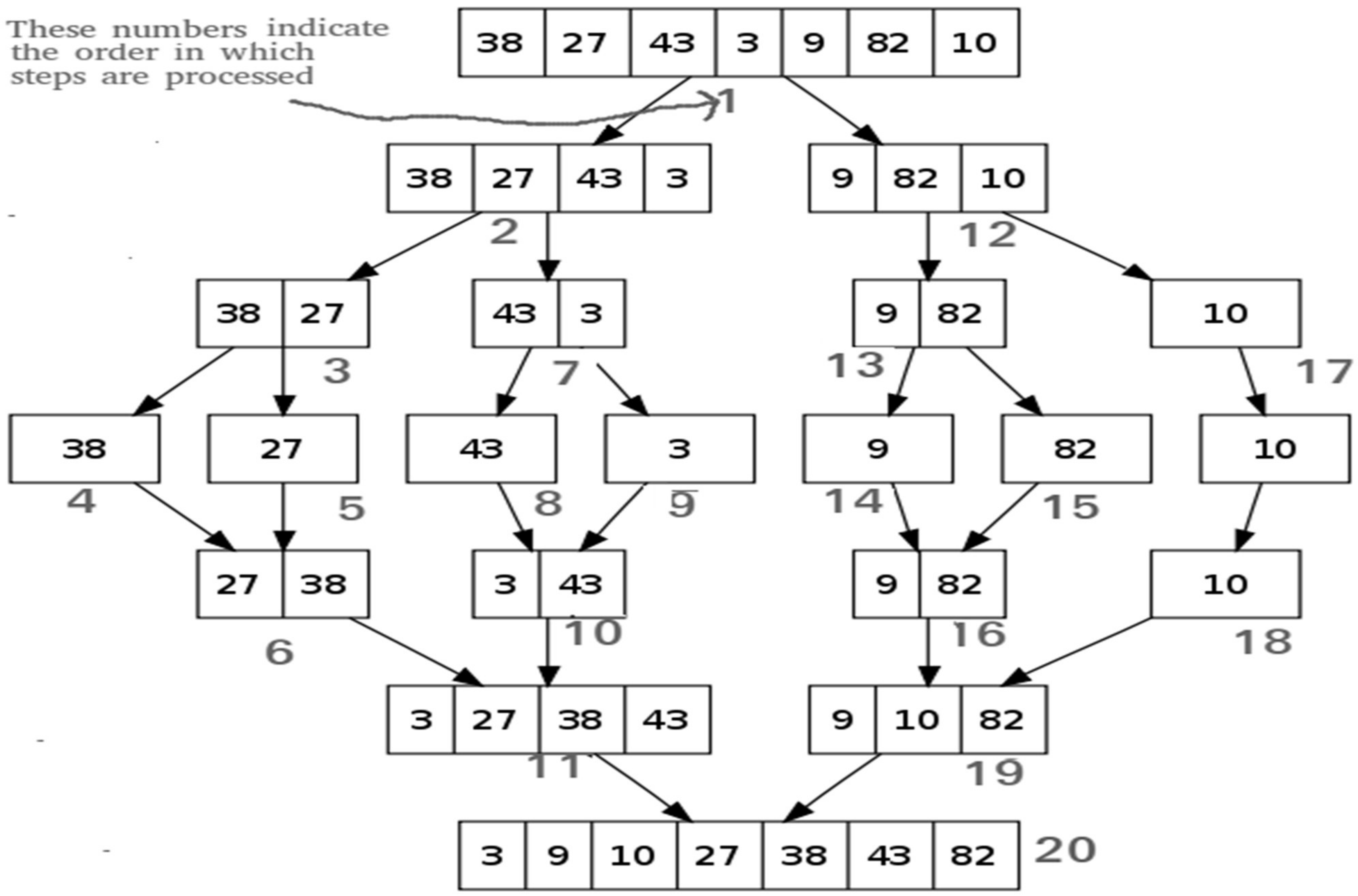


14 33 27 10 35 19 42 44
14 33 27 10 | 35 19 42 44
14 33 | 27 10 | 35 19 | 42 44
14 | 33 | 27 | 10 | 35 | 19 | 42 | 44
14 33 | 10 27 | 19 35 | 42 44
10 14 27 33 | 19 35 42 44
10 14 19 27 33 35 42 44

Merge Sort Algorithm

- **Step 1** – if it is only one element in the list it is already sorted, return.
- **Step 2** – divide the list recursively into two halves until it can no more be divided.
- **Step 3** – merge the smaller lists into new list in sorted order.

Merge Sort


These numbers indicate the order in which steps are processed
38 27 43 3 9 82 10
1
38 27 43 3
2
9 82 10
12
38 27
3
43 3
7
9 82
13
10
17
38
4
27
5
43
8
3
9
9
14
82
15
10
27 38
6
3 43
10
9 82
16
10
18
3 27 38 43
11
9 10 82
19
3 9 10 27 38 43 82
20

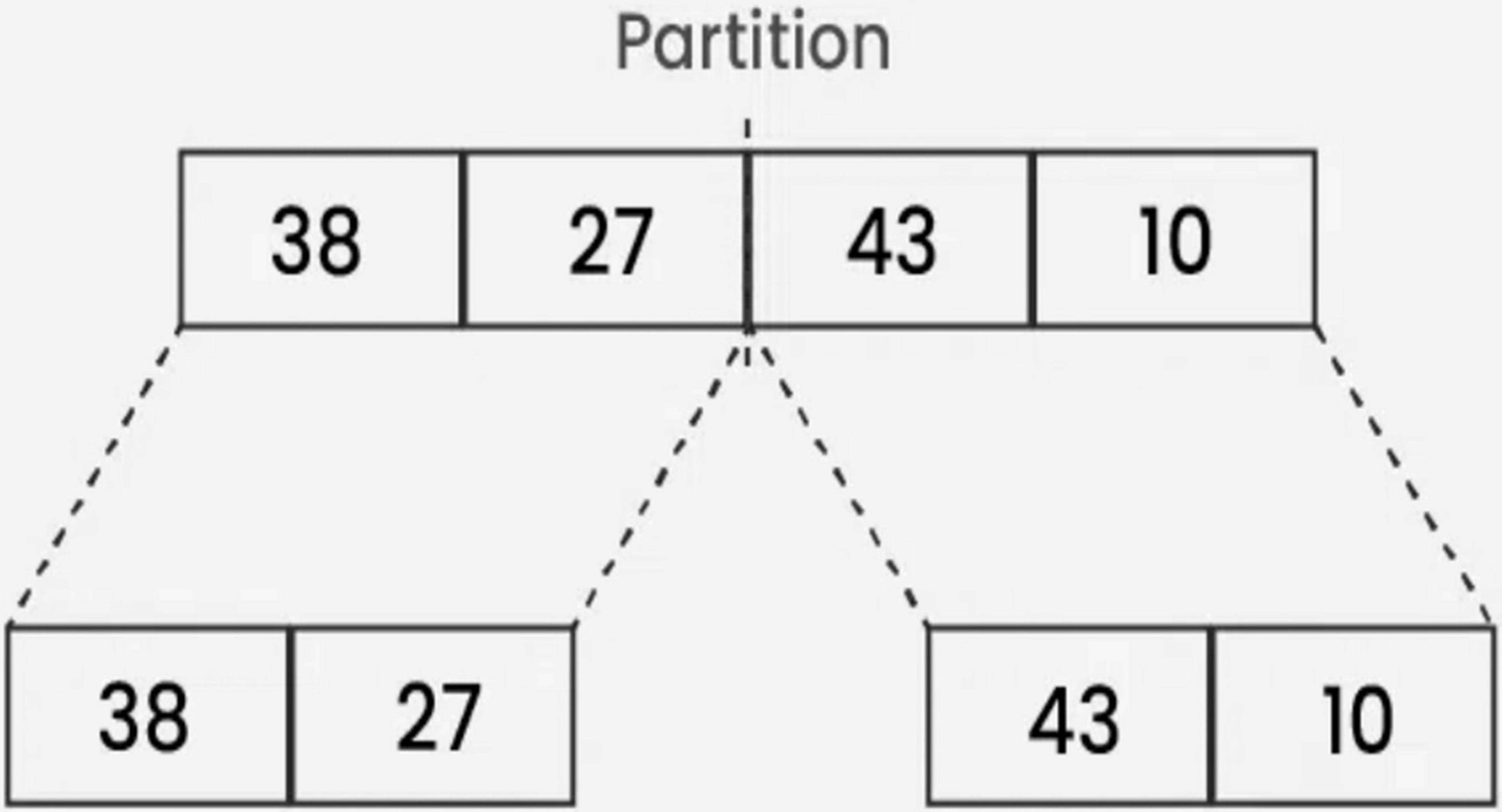
Step-by-Step Explanation

Divide: Divide the list or array recursively into two halves until it can no more be divided.

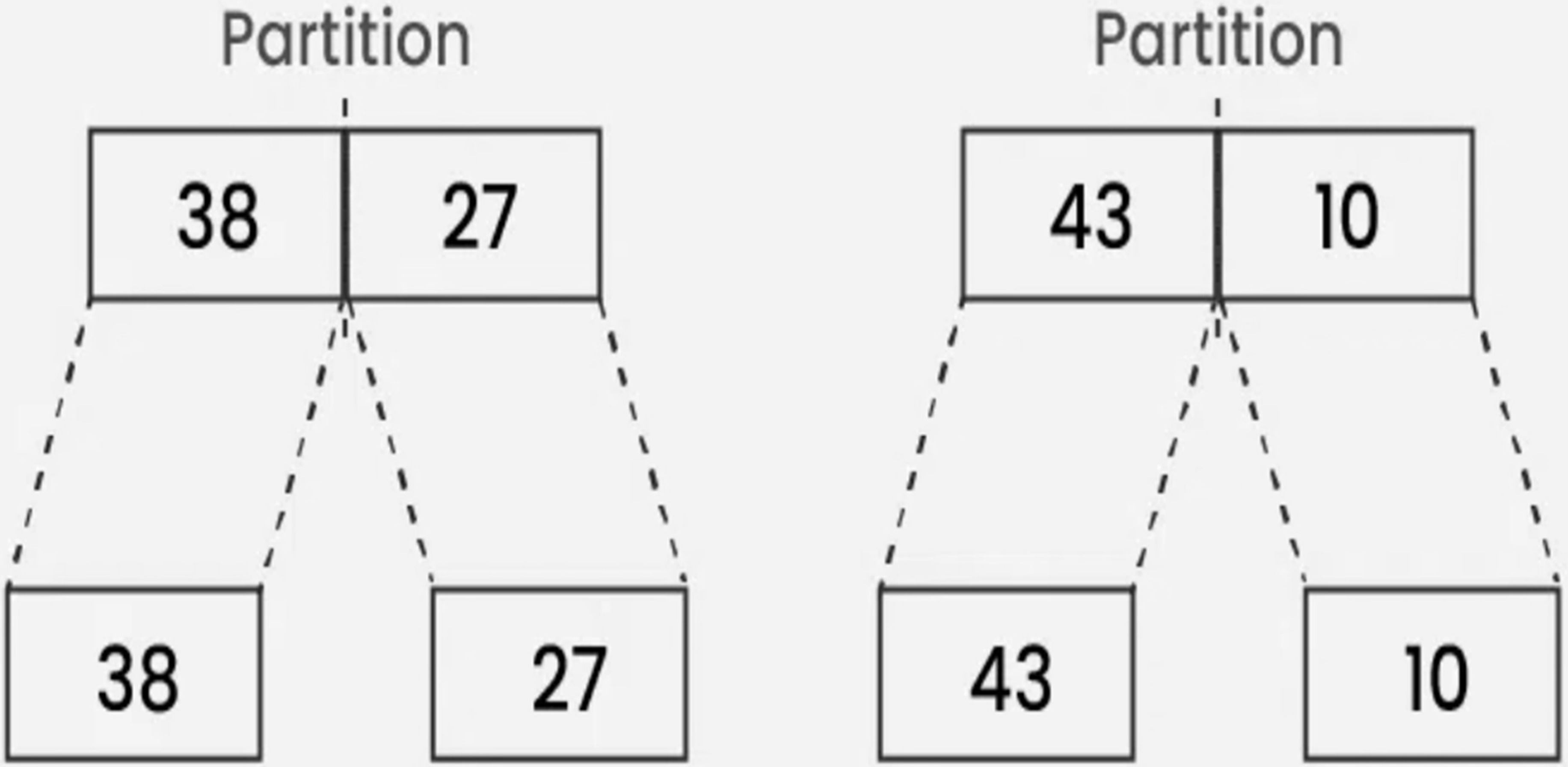
Conquer: Each subarray is sorted individually using the merge sort algorithm.

Merge: The sorted subarrays are merged back together in sorted order. The process continues until all elements from both subarrays have been merged.

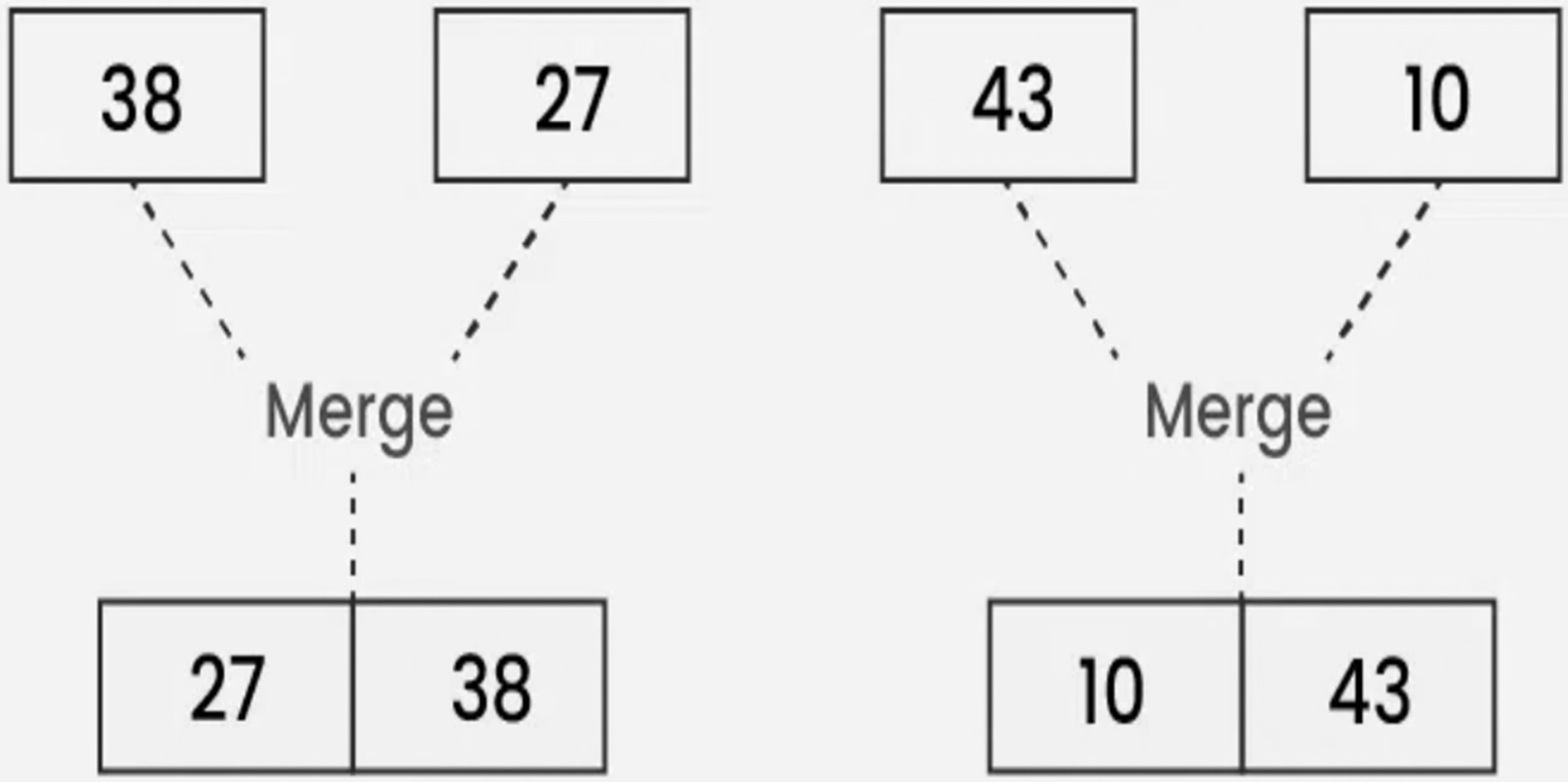
Step 1 | Splitting the Array into two equal halves



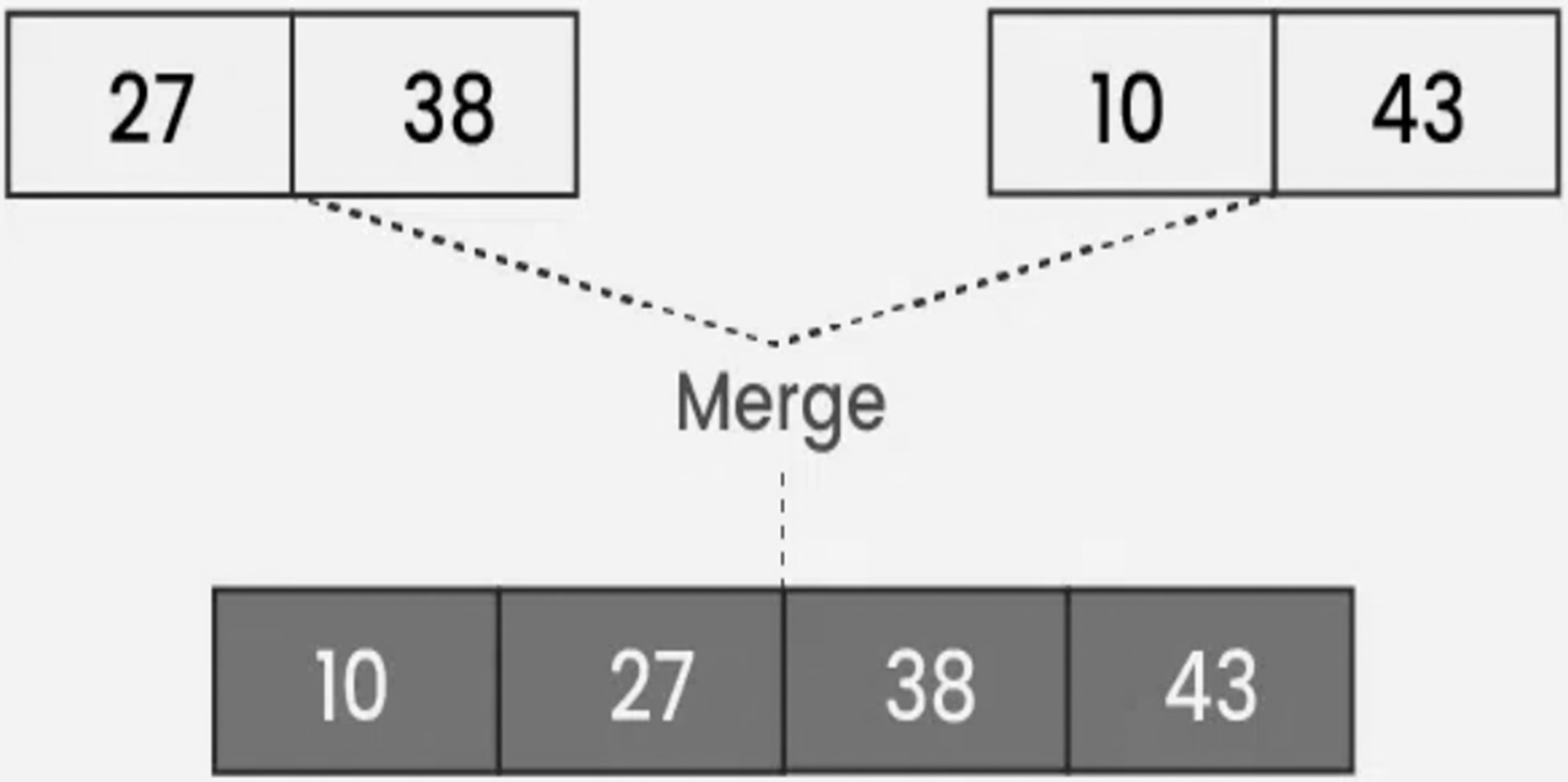
Step 2 | Splitting the subarrays into two halves



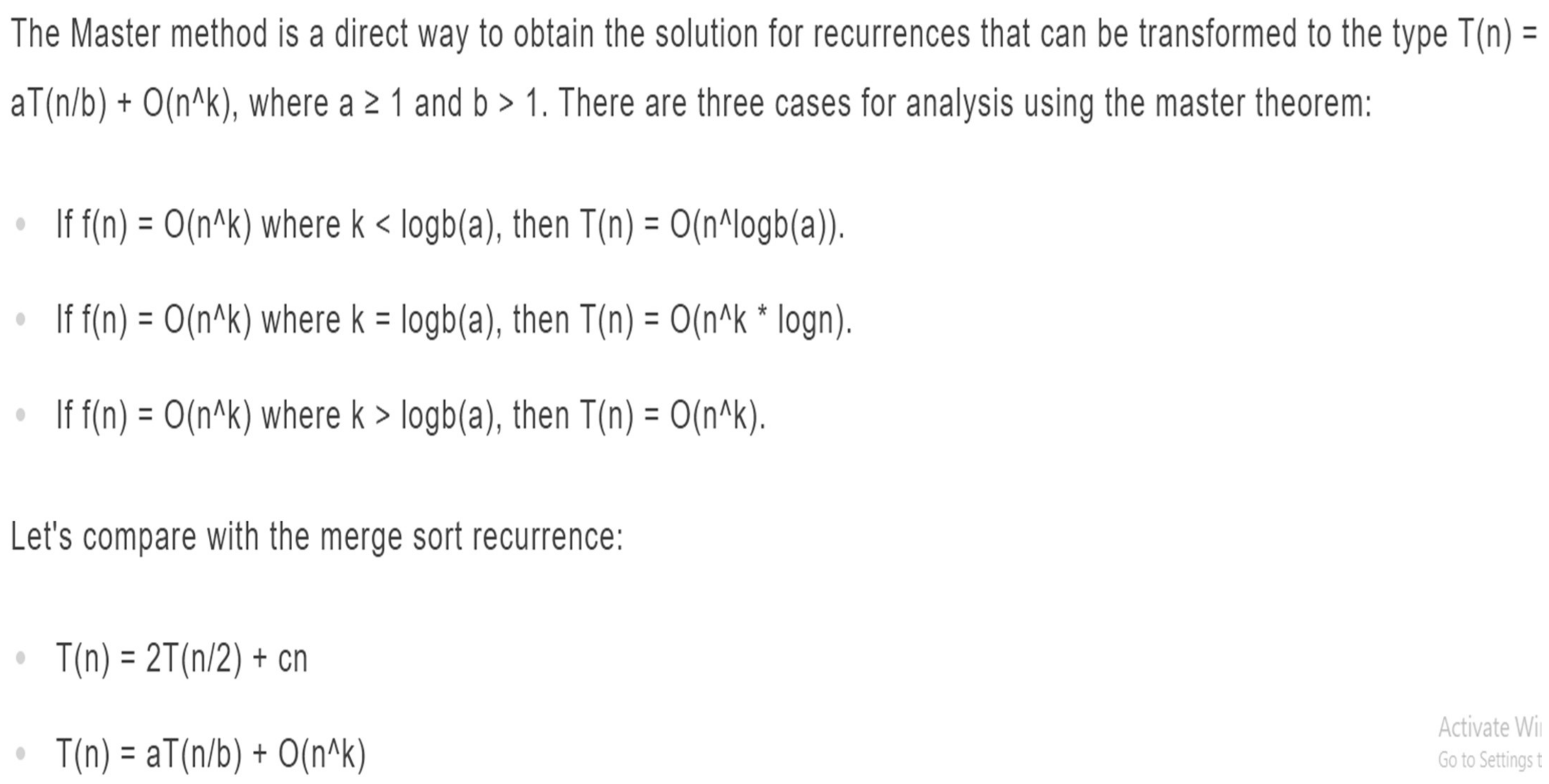
Step 3 | Merging unit length cells into sorted subarrays



Step 4 Merging sorted subarrays into the sorted array



Time Complexity ?



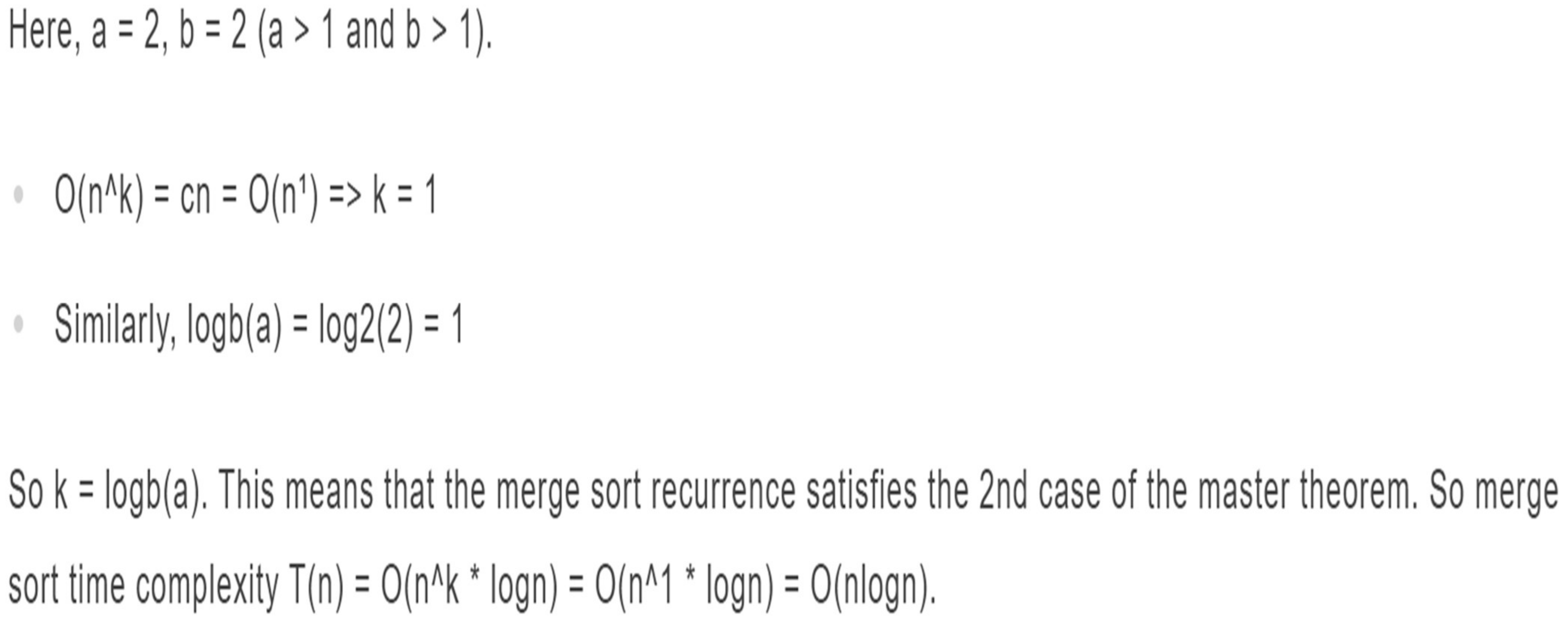
The Master method is a direct way to obtain the solution for recurrences that can be transformed to the type T(n) = aT(n/b) + O(n^k), where a ≥ 1 and b > 1. There are three cases for analysis using the master theorem:

- If f(n) = O(n^k) where k < logb(a), then T(n) = O(n^logb(a)).
- If f(n) = O(n^k) where k = logb(a), then T(n) = O(n^k * logn).
- If f(n) = O(n^k) where k > logb(a), then T(n) = O(n^k).

Let's compare with the merge sort recurrence:

- T(n) = 2T(n/2) + cn
- T(n) = aT(n/b) + O(n^k)

Time Complexity ?



Here, a = 2, b = 2 (a > 1 and b > 1).

- O(n^k) = cn = O(n^1) => k = 1
- Similarly, logb(a) = log2(2) = 1

So k = logb(a). This means that the merge sort recurrence satisfies the 2nd case of the master theorem. So merge sort time complexity T(n) = O(n^k * logn) = O(n^1 * logn) = O(nlogn).

Another Way!!!

Quick Sort

- Quick sort is a highly efficient sorting algorithm and is based on partitioning of array of data into smaller arrays.
- A large array is partitioned into two arrays one of which holds values smaller than the specified value, say pivot, based on which the partition is made and another array holds values greater than the pivot value.
- Quicksort partitions an array and then calls itself recursively twice to sort the two resulting subarrays.
- This algorithm is quite efficient for large-sized data sets.

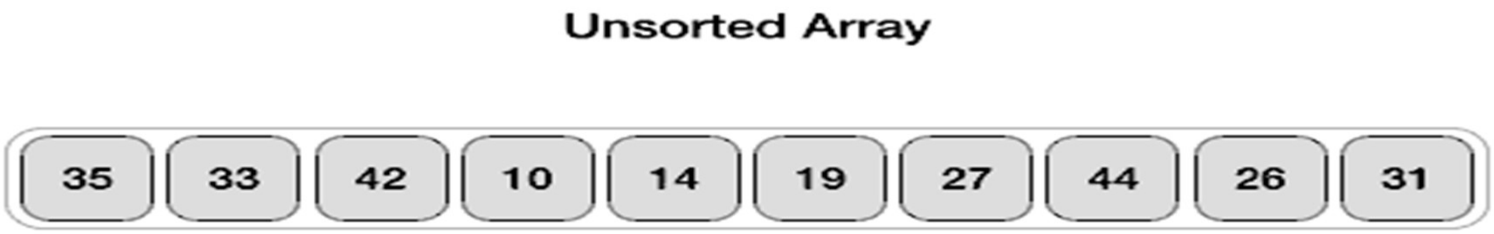
Quick Sort Pivot

- **Step 1** − Choose the highest index value has pivot
- **Step 2** − Take two variables to point left and right of the list excluding pivot
- **Step 3** − left points to the low index
- **Step 4** − right points to the high
- **Step 5** − while value at left is less than pivot move right
- **Step 6** − while value at right is greater than pivot move left
- **Step 7** − if both step 5 and step 6 does not match swap left and right
- **Step 8** − if left ≥ right, the point where they met is new pivot

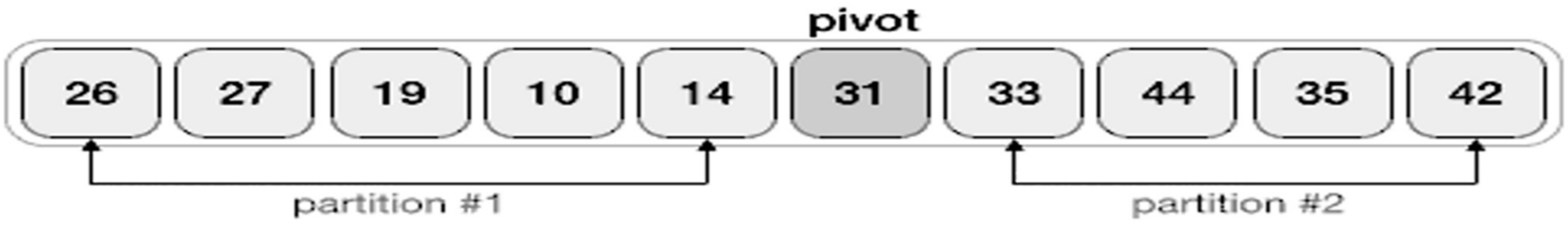
Quick Sort Algorithm

- **Step 1** – Make the right-most index value pivot
- **Step 2** – partition the array using pivot value
- **Step 3** – quicksort left partition recursively
- **Step 4** – quicksort right partition recursively

Quick Sort

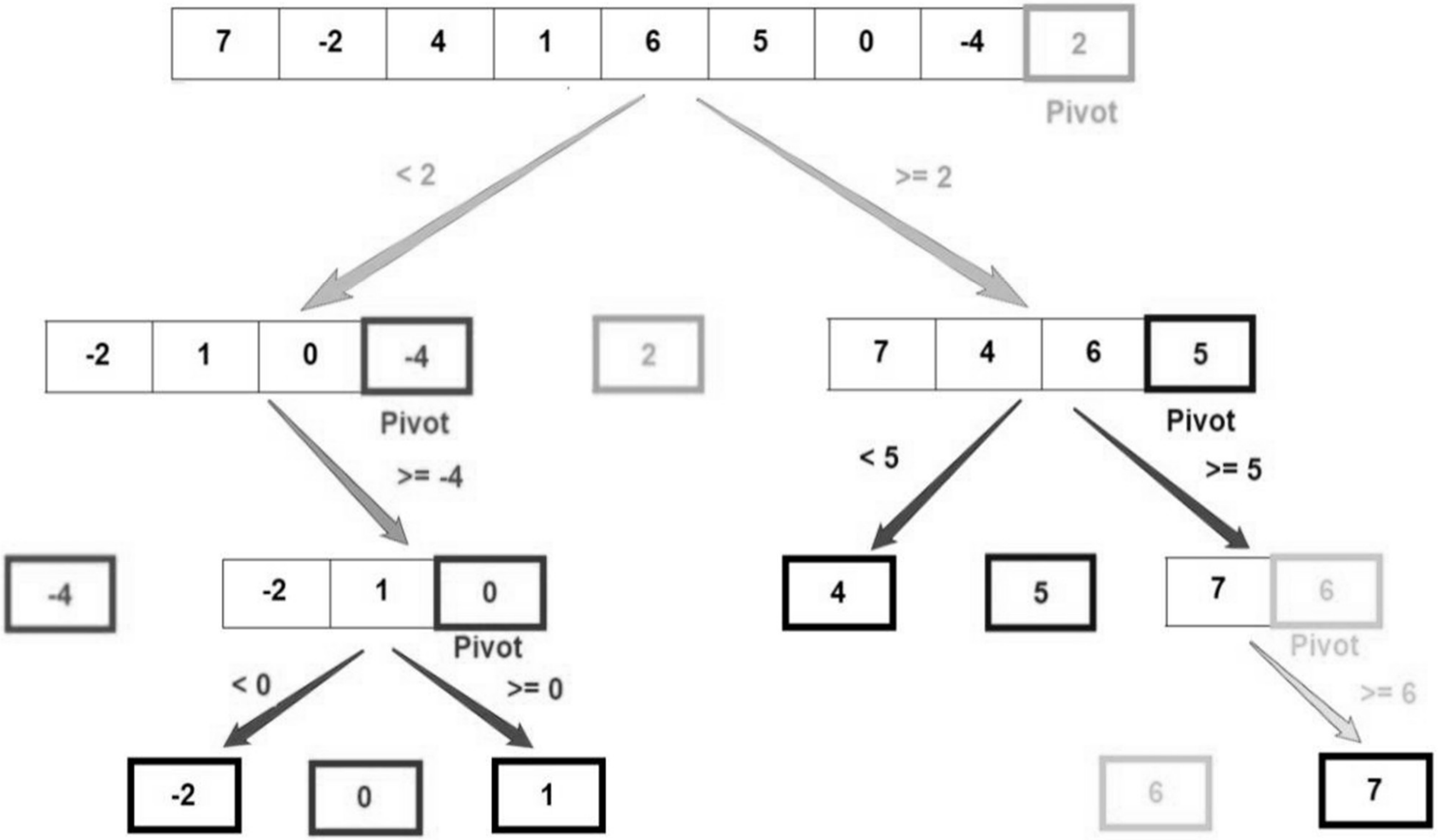


Quick Sort



lo			hi	Pivot		lo		hi	Pivot
26	27	19	10	14	31	33	44	35	42
	lo	hi					lo	hi	
10	27	19	26	14	31	33	44	35	42
	lo & hi							lo & hi	
10	27	19	26	14	31	33	35	44	42
10	14	19	26	27	31	33	35	42	44

Quick Sort


7
-2
4
1
6
5
0
-4
2
Pivot
< 2
>= 2
-2
1
0
-4
Pivot
2
7
4
6
5
Pivot
>= -4
< 5
>= 5
-4
-2
1
0
Pivot
4
5
7
6
Pivot
< 0
>= 0
>= 6
-2
0
1
6
7

HOMEWORK

Estimate the time complexities of Merge sort Algorithm (Best and Average) ???

Thank You