Merge Sort

Course on Sorting and Searching

Rohit Mazumder • Lesson 2 • Feb 19, 2021

Agenda

1. Recap: The sorting problem
2. Divide & Conquer - The Motivation
3. Divide & Conquer - Steps
4. Merge Sort - Explanation of Algorithm
5. The Merge Step
6. Merge Sort - PseudoCode
7. Merge Sort - Implementation in JAVA
8. Time Complexity Analysis
9. Auxiliary Space requirement
10. Applications/Classical Problems based on Merge Sort
11. Assessment
12. Practice Problems/HW

What is the sorting problem?

Arranging elements in a specific order.

Commonly used orders are:

- Ascending order
- Descending order

Target: Sort an array in ascending order

Merge-Sort

Merge-Sort is based on the concept of divide and conquer.

Divide & Conquer - Motivation

Question:

Count the number of ones in the array [1, 3, 4, 1, 1, 5, 1] ⟶ 4 ones

Question:

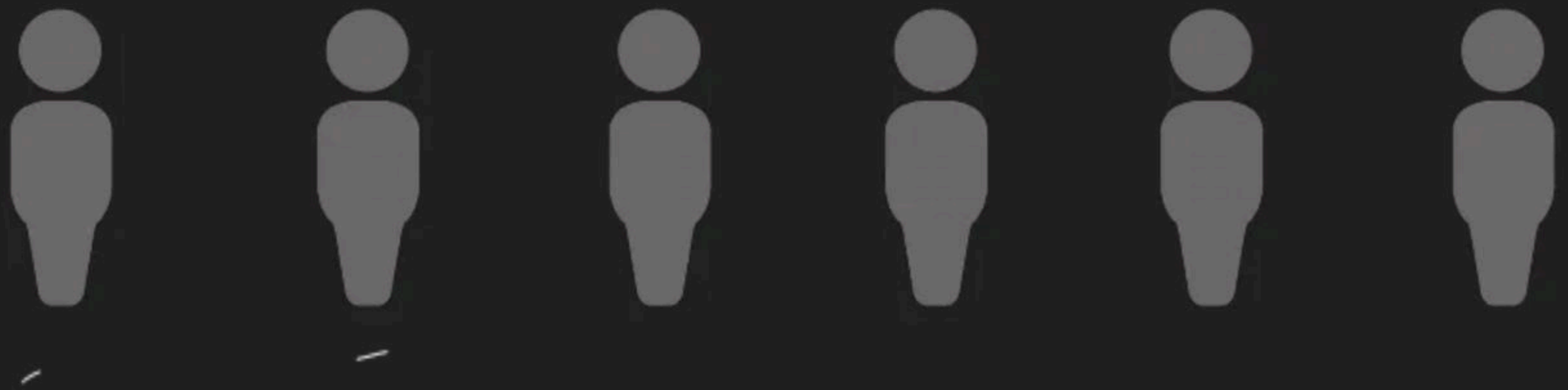
Count the number of ones in the array:

[1, 3, 4, 1, 1, 5, 1, 6, 5, 16, 11, 1, 70, 1, 1, 23, 1, 70, 70, 70, 115, 45, 1, 45, 1, 3, 4, 1, 12, 34, 65, 67, 1, 45, 1, 1, 1, 6, 1, 7, 8, 9, 0, 12, 43, 67, 99, 34, 65, 77, 12, 09, 38, 65, 66, 1, 6, 6, 1, 7]

Question:

Count the number of ones in the array:

[1, 3, 4, 1, 1, 5, 1, 6, 5, 16, 11, 1, 70, 1, 1, 23, 1, 70, 70, 70, 115, 45, 1, 45, 1, 3, 4, 1, 12, 34, 65, 67, 1, 45, 1, 1, 1, 6, 1, 7, 8, 9, 0, 12, 43, 67, 99, 34, 65, 77, 12, 09, 38, 65, 66, 1, 6, 6, 1, 7]

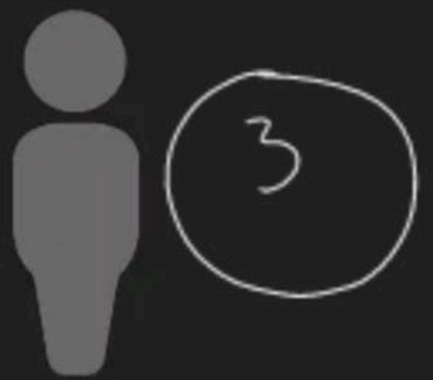


Question:

4 + 4 + 3 + 5 + 0 + 2 = 18

[1, 3, 4, 1, 1, 5, 1, 6, 5, 16]

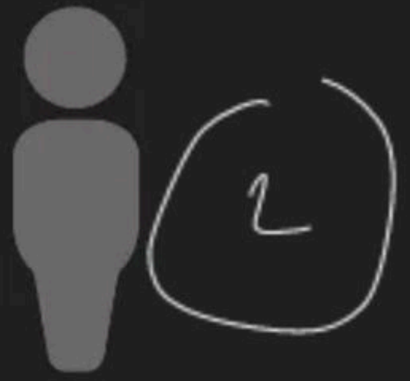
[11, 1, 70, 1, 1, 23, 1, 70, 70, 70]



[115, 45, 1, 45, 1, 3, 4, 1, 12, 34]

[65, 67, 1, 45, 1, 1, 1, 6, 1, 7]

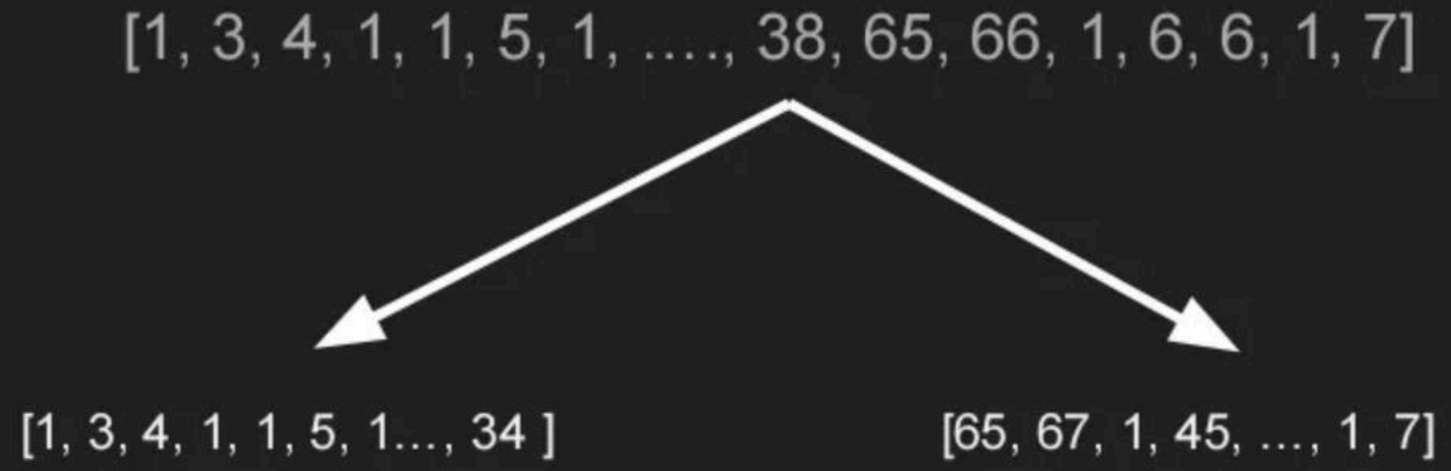
[8, 9, 0, 12, 43, 67, 99, 34, 65, 77]

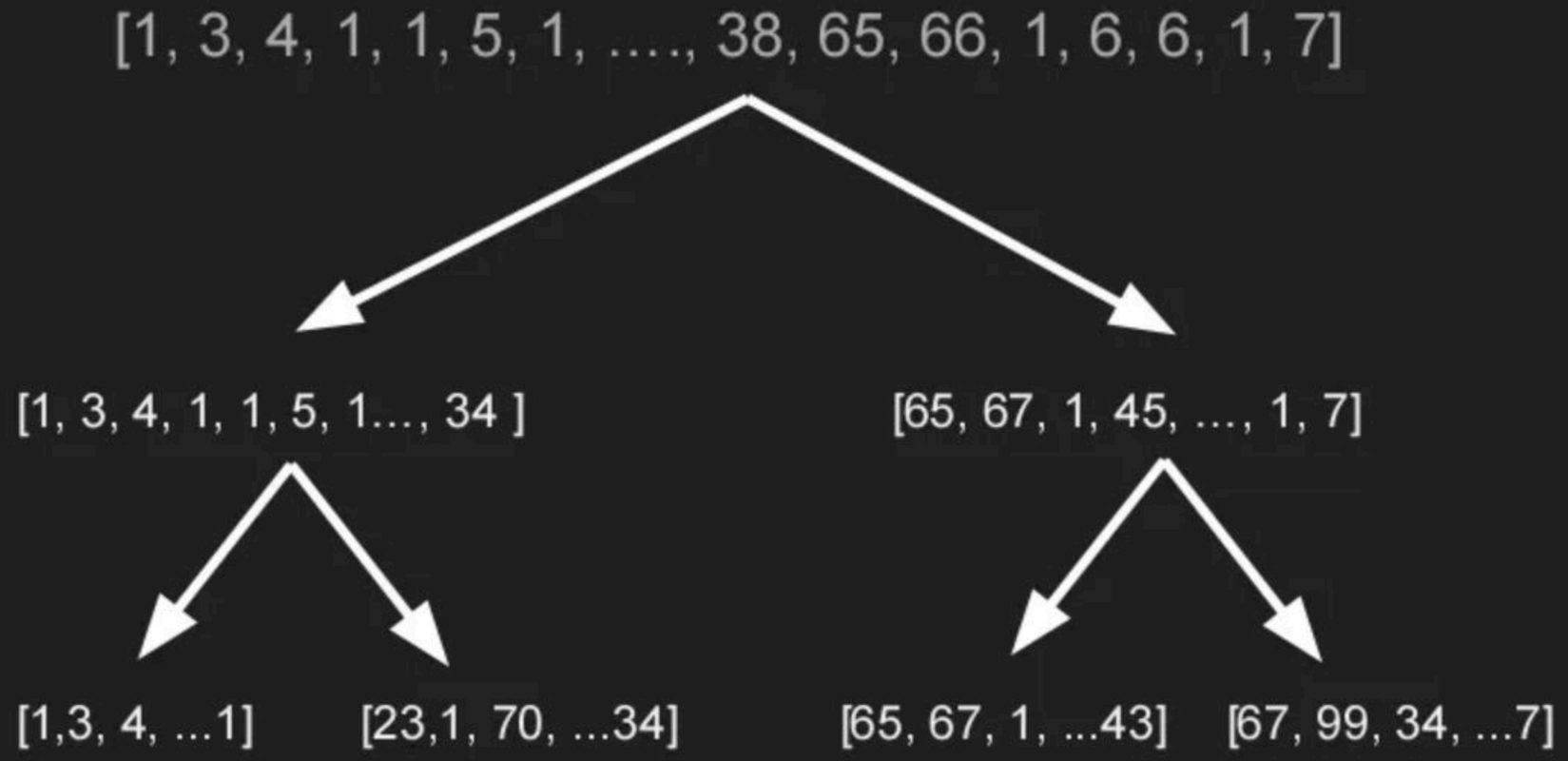


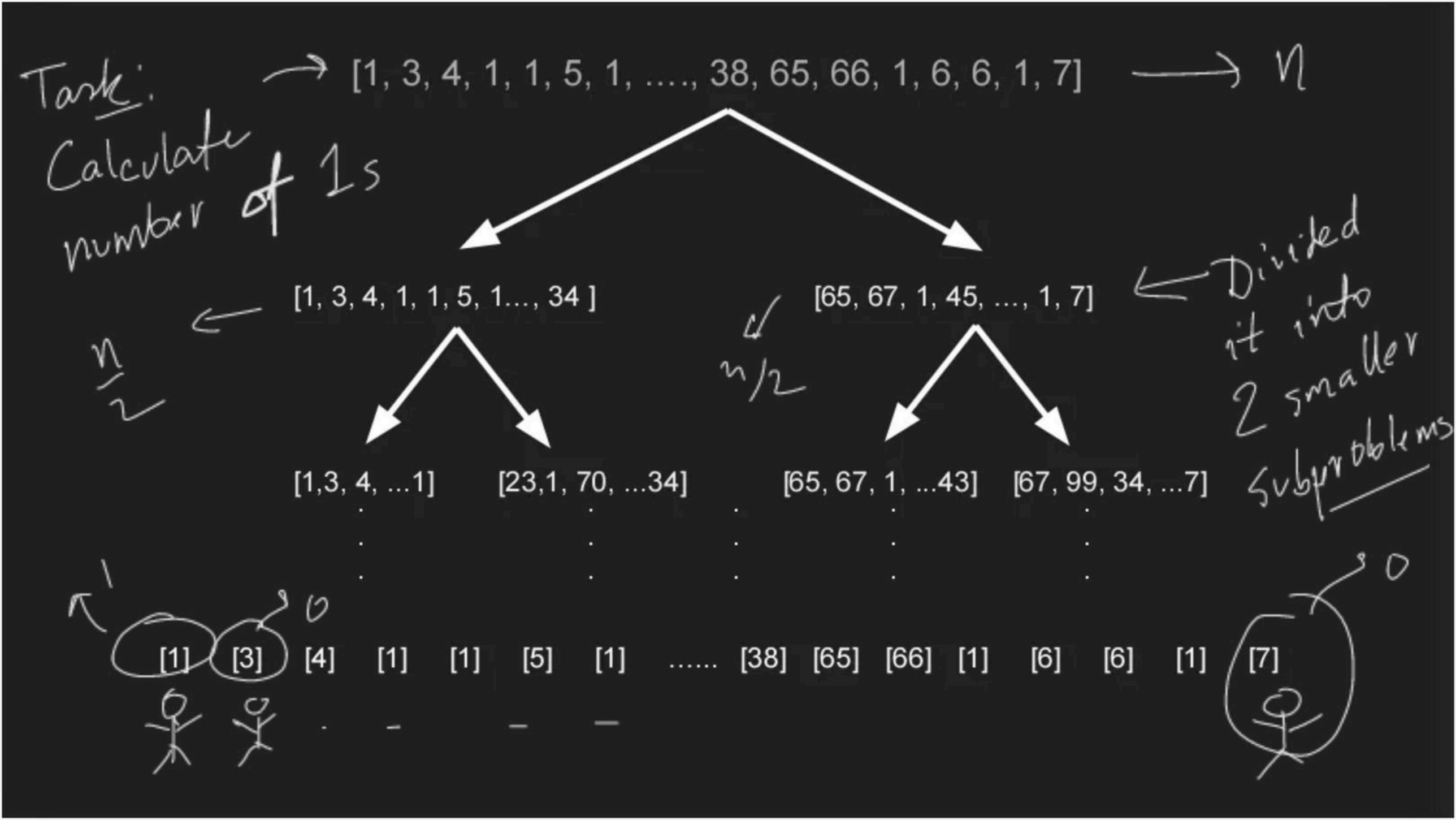
[12, 09, 38, 65, 66, 1, 6, 6, 1, 7]

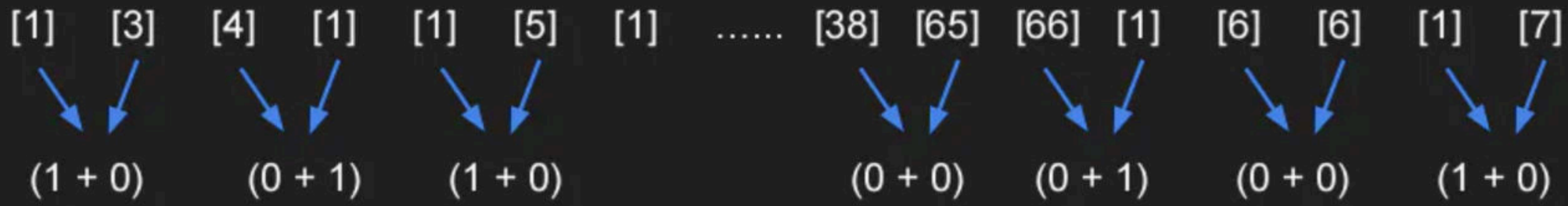
- The more people we employ, the easier it is to the tackle the task.
- In an ideal scenario, we would love to employ as many people as the number of elements in the array!

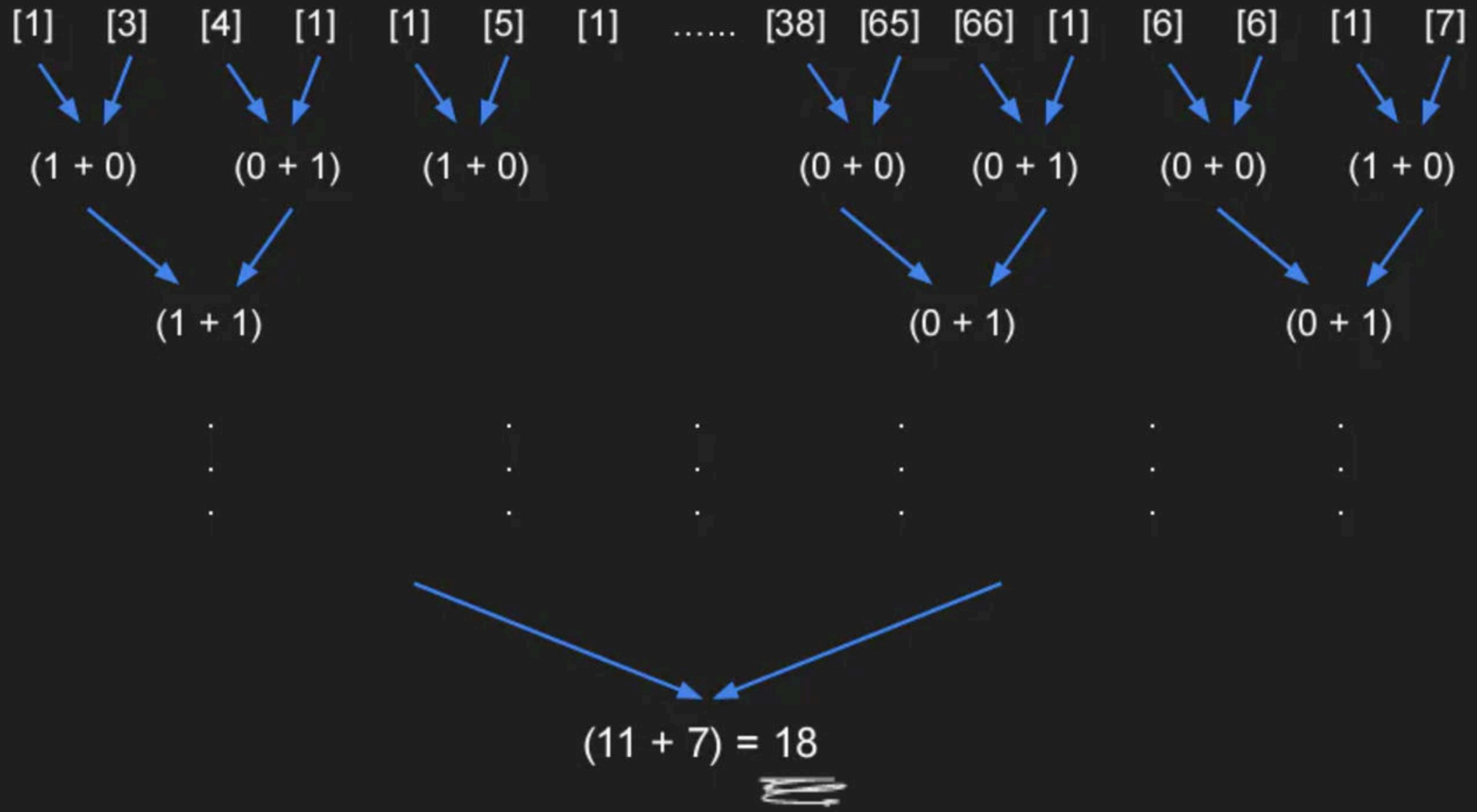
[1, 3, 4, 1, 1, 5, 1, …., 38, 65, 66, 1, 6, 6, 1, 7]


[1, 3, 4, 1, 1, 5, 1, …., 38, 65, 66, 1, 6, 6, 1, 7]
[1, 3, 4, 1, 1, 5, 1…, 34]
[65, 67, 1, 45, …, 1, 7]


[1, 3, 4, 1, 1, 5, 1, …., 38, 65, 66, 1, 6, 6, 1, 7]
[1, 3, 4, 1, 1, 5, 1..., 34]
[65, 67, 1, 45, ..., 1, 7]
[1,3, 4, ...1]
[23,1, 70, ...34]
[65, 67, 1, ...43]
[67, 99, 34, ...7]

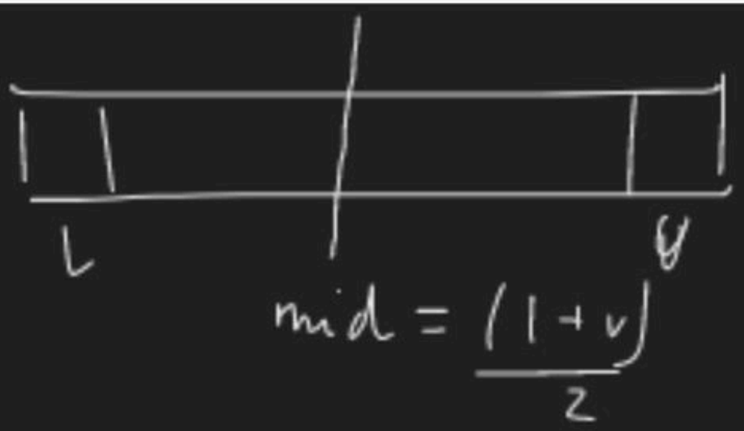

Task:
Calculate number of 1s
[1, 3, 4, 1, 1, 5, 1, …., 38, 65, 66, 1, 6, 6, 1, 7]
n
[1, 3, 4, 1, 1, 5, 1..., 34]
[65, 67, 1, 45, ..., 1, 7]
n/2
n/2
Divided it into 2 smaller subproblems
[1,3, 4, ...1]
[23,1, 70, ...34]
[65, 67, 1, ...43]
[67, 99, 34, ...7]
1
0
[1] [3] [4] [1] [1] [5] [1] ..... [38] [65] [66] [1] [6] [6] [1] [7]
0


[1] [3] [4] [1] [1] [5] [1] ..... [38] [65] [66] [1] [6] [6] [1] [7]
(1 + 0) (0 + 1) (1 + 0) (0 + 0) (0 + 1) (0 + 0) (1 + 0)


[1] [3] [4] [1] [1] [5] [1] ..... [38] [65] [66] [1] [6] [6] [1] [7]
(1 + 0) (0 + 1) (1 + 0) (0 + 0) (0 + 1) (0 + 0) (1 + 0)
(1 + 1) (0 + 1) (0 + 1)
(11 + 7) = 18

Divide & Conquer - Steps

- Divide a problem into smaller, simpler, and similar sub-problems,
- Conquer subproblems by calling recursively until they are trivial enough to be solved.
- Merge the solution to the sub-problems to find out the answer to the actual problem.




```
/* Returns the number of ones in A in the range l to u */
int numberOfOnes(int[] A, int l, int u){
    if(l < u) {
        int numberOfOnesInLeft = numberOfOnes(A, l, (l + u)/2);
        int numberOfOnesInRight = numberOfOnes(A, (l + u)/2 + 1, u);
        return numberOfOnesInLeft + numberOfOnesInRight;
    }
}
```

T(n)

T(n/2)

T(n/2)

O(1)

numberOfOnes(A, 0, A.length - 1);

Let's say numberOfOnes(1) → T(n)

where n = length from l to u

$$T(n) = 2T\left(\frac{n}{2}\right) + O(1) \leftarrow \text{Recurrence Relation.}$$

$$= 2T\left(\frac{n}{2}\right) + c$$

$$= 2\left[2T\left(\frac{n}{4}\right) + c\right] + c$$

$$T(n) = 4T\left(\frac{n}{4}\right) + 3c \quad \text{——— (1)}$$

$$T(n) = 4T\left(\frac{n}{4}\right) + 3c$$

$$T(n) = 4\left(2T\left(\frac{n}{8}\right) + c\right) + 3c$$

$$= 8T\left(\frac{n}{8}\right) + 7c \quad \text{---} \quad (2)$$

Similarly,

$$\boxed{T(n) = nT\left(\frac{n}{n}\right) + (n-1)c} \quad \text{---} \quad (3)$$

$$T(n) = n\,T\left(\frac{n}{n}\right) + (n-1)\,c$$

$$= n\,T(1) + (n-1)\,c$$

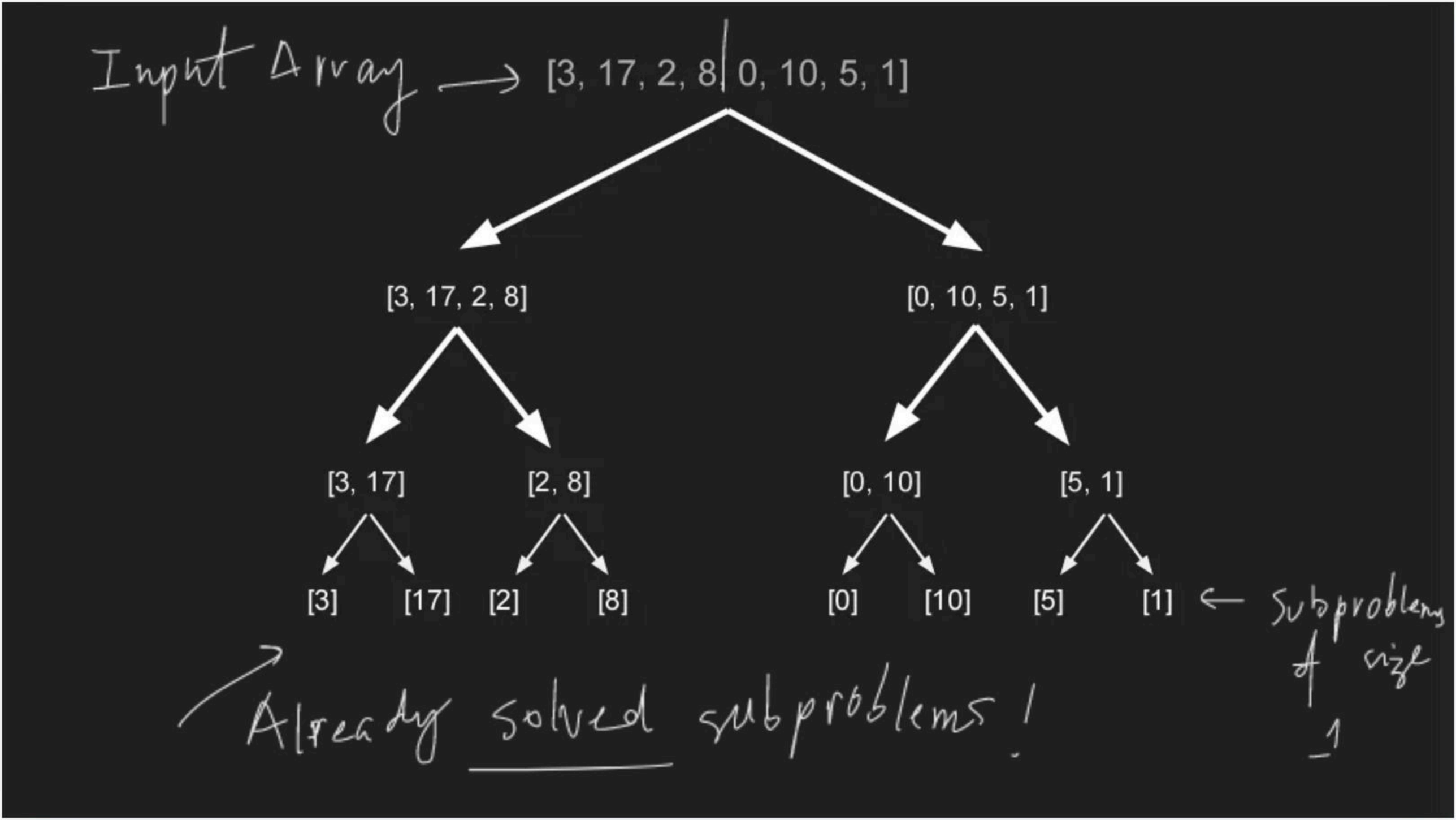
$$T(n) = nk + (n-1)\,c$$

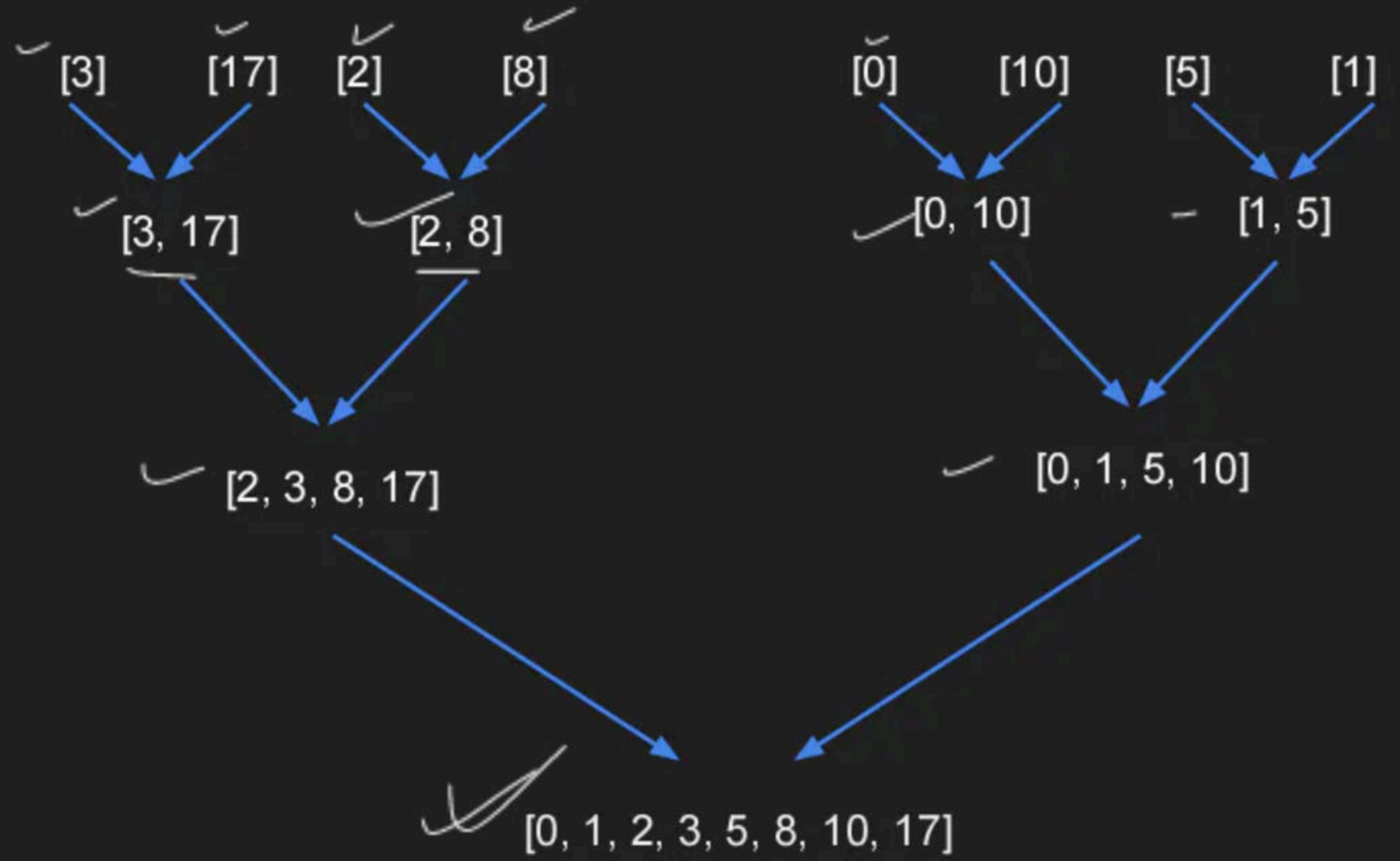
$$T(n) = O(n)$$ ← Time Complexity

Divide and Conquer - Examples:

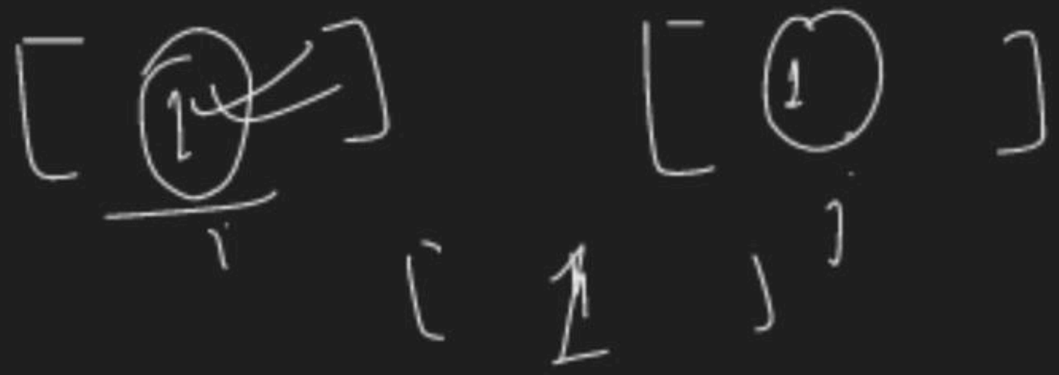
1. Merge Sort
2. Quick Sort
3. Binary Search

How do these steps apply merge sort?


Input Array → [3, 17, 2, 8 | 0, 10, 5, 1]
[3, 17, 2, 8]
[0, 10, 5, 1]
[3, 17]
[2, 8]
[0, 10]
[5, 1]
[3]
[17]
[2]
[8]
[0]
[10]
[5]
[1]
← Subproblems of size 1
Already solved subproblems!


[3]
[17]
[2]
[8]
[0]
[10]
[5]
[1]
[3, 17]
[2, 8]
[0, 10]
[1, 5]
[2, 3, 8, 17]
[0, 1, 5, 10]
[0, 1, 2, 3, 5, 8, 10, 17]

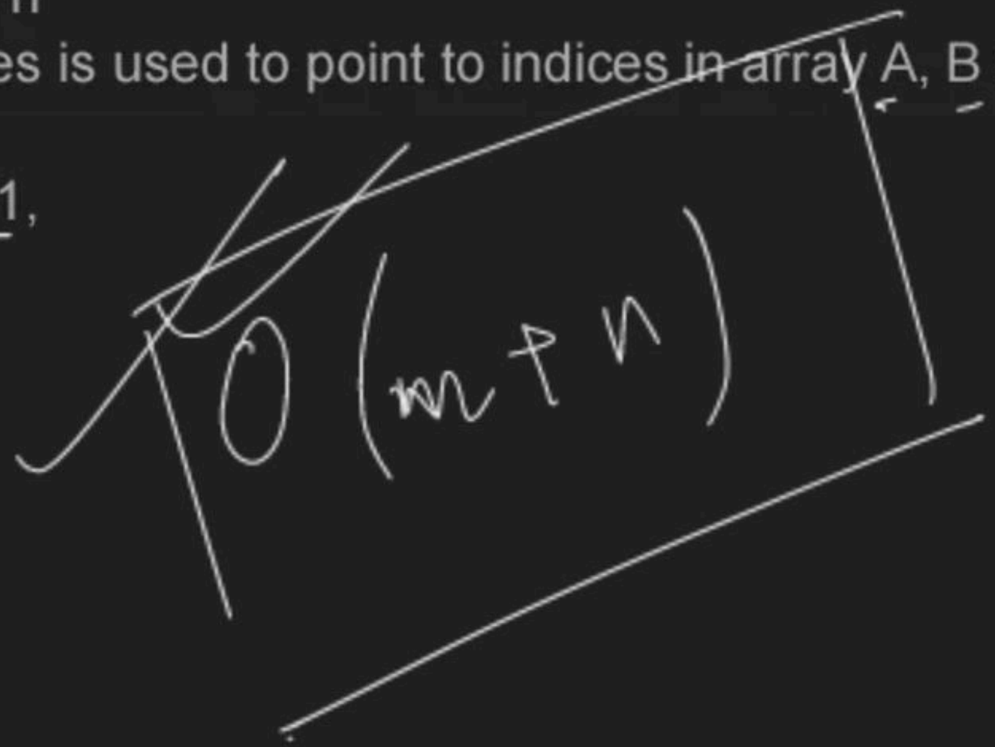
The Merge Step



Target: Merge two sorted arrays into one.

Let A and B be two sorted array. We want to merge them to get the sorted array C as output.
Assume size(A) = m, size(B) = n, size(C) = m + n
Step 1: Initialise i = 0, j = 0, k = 0, these variables is used to point to indices in array A, B and C respectively.
Step 2: Repeat the loop for k = 0 to k = m + n - 1,

a. If A[i] <= B[j] then:
   C[k] = A[i];
   i = i + 1;
b. else:
   C[k] = B[j];
   j = j + 1



Merging - Example

size = m

A: [1, 2, 8, 17, 19]

size = n

B: [3, 9, 10, 18, 20, 21]

C: [1, 2, 3, 8, 9, 10, 17, 18, 19, 20, 21]

size = m + n

$O(\frac{n}{2}+n) = O(2n) = O(n)$

Merge Sort - Pseudo-code

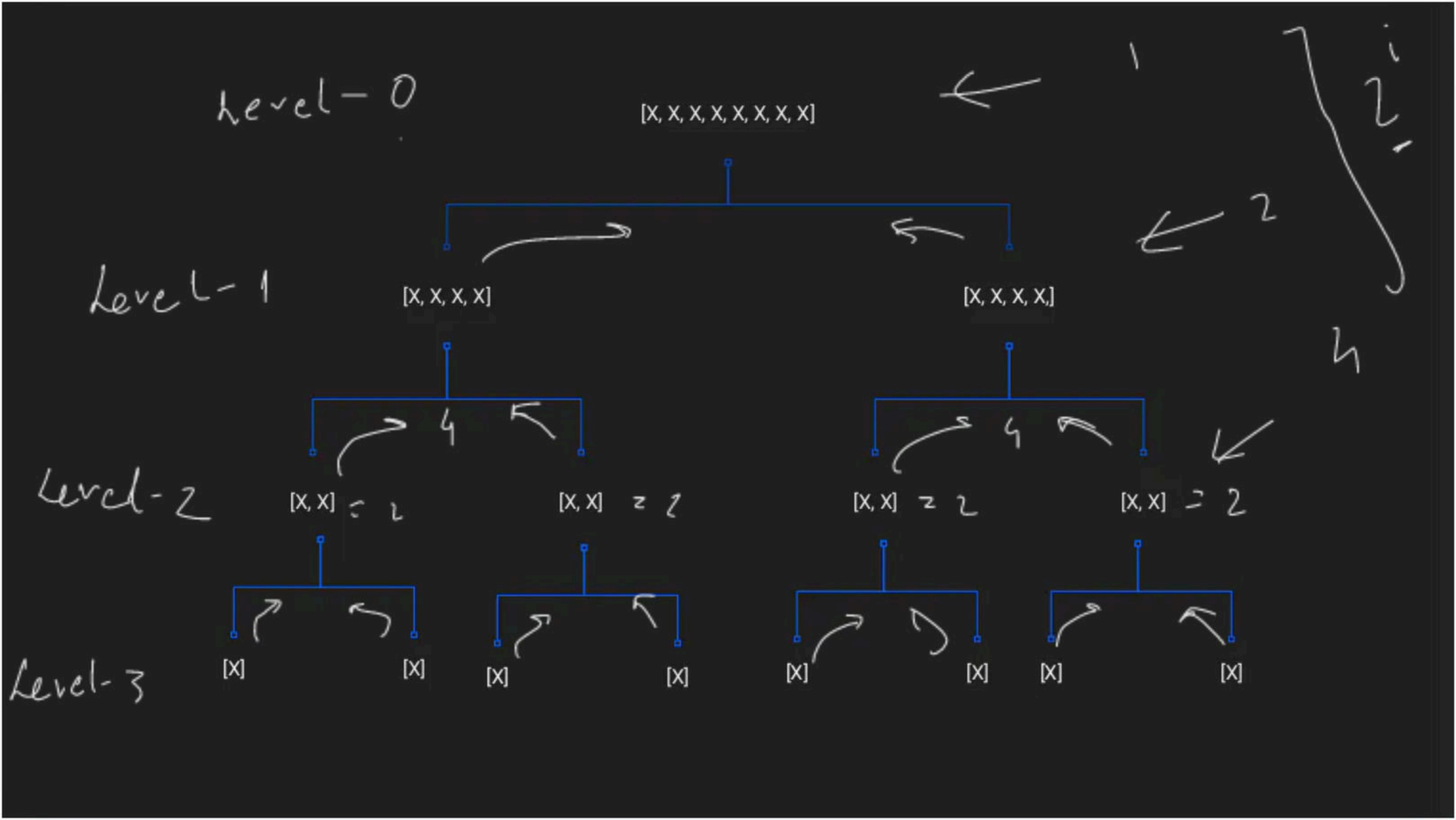
size = n

```
MergeSort(A, l, r):

1. if l < r:
2.     mid = floor((l + r)/2)
3.     MergeSort(A, l, mid)        → T(n/2)
4.     MergeSort(A, mid + 1, r)    → T(n/2)
5.     Merge(A, l, mid, r)         → O(n)
```

$$T(n) = 2 * T\left(\frac{n}{2}\right) + O(n)$$

Time Complexity Analysis


Level-0
[X, X, X, X, X, X, X, X]
1
2
Level-1
[X, X, X, X]
[X, X, X, X]
2
4
4
Level-2
[X, X] = 1
[X, X] = 1
[X, X] = 2
[X, X] = 2
[X]
[X]
[X]
[X]
[X]
[X]
[X]
[X]
Level-3

There are $\lceil \log_2 n \rceil + 1$ levels $0 \rightarrow \log_2 n$

In a level-i there will be 2^i subproblems

Size of each subproblem $= \frac{n}{2^i}$

Total work at a level-i $= \frac{n}{2^i} \times 2^i$

" " done in tree

Total work done in the tree

= Work done in one level * # levels

$= n * (\lceil \log_2 n \rceil + 1) = T(n)$

$$T(n) = O(n \log n)$$

In the recursion tree for an array of n elements, the number of ~~elements~~ levels are:

1. n
2. ceil($\log_2 n$) + 1
3. ceil($\sqrt{n}$) + 1
4. n^2

$n = 8$

$\lceil \log_2 8 \rceil + 1$

$= 4$

O(n)

Auxiliary Space Complexity Analysis

Merge-sort is not an in-place algo !

Is merge sort a stable algorithm?

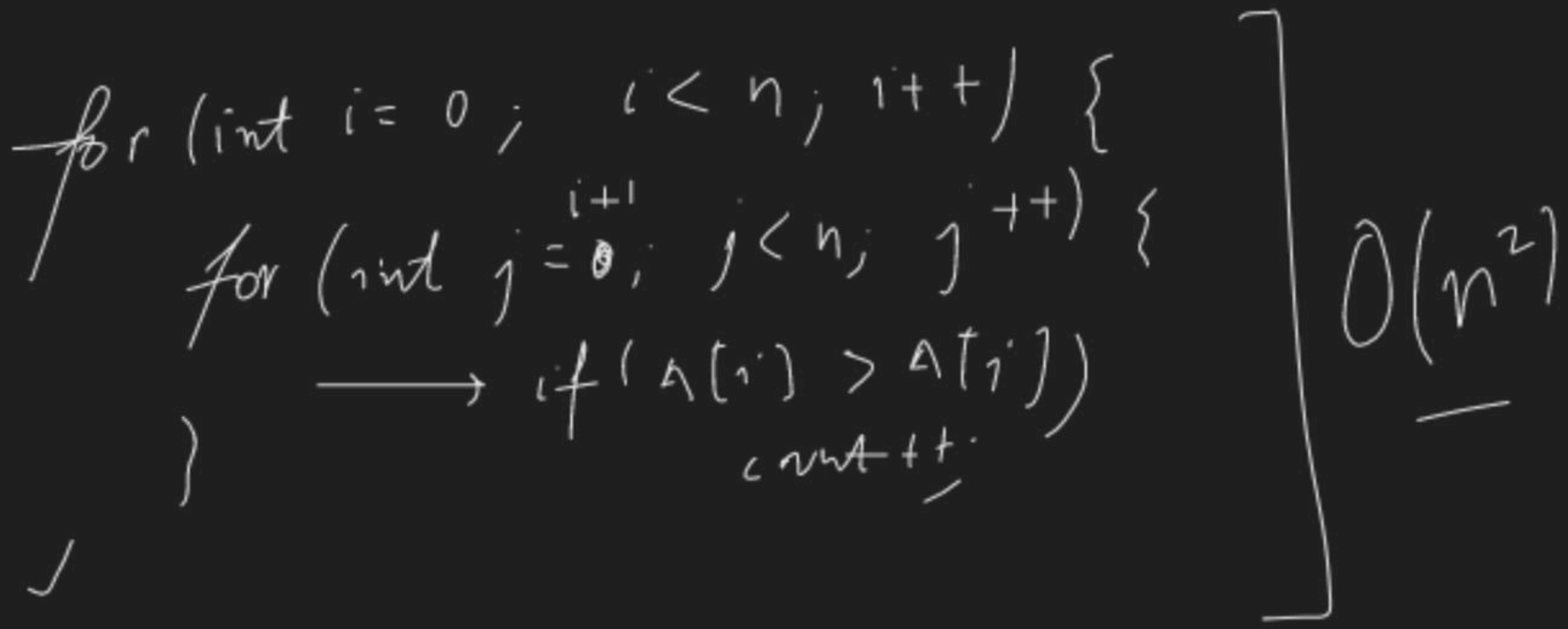
A. True

B. False

Application of Merge Sort

Number of inversions in an array

- Number of inversions refers to the number of pairs (A[i], A[j]) such that i < j and A[i] > A[j].
- It is a measure of "how much sorted is an array".



Example:

[3, 17, 2, 8, 0, 10, 5, 1]
0 1 2 3 4 5 6 7

Output:

17

[1, 2, 3, 4, 5]

(17, 2) → Inverted pair
1 2

(17, 8)

(10, 5)

(2, 0)

(2, 1

:

:

:

Divide & Conquer ?

(17, 2) (10, 5)
(12, 8) (5, 1)
(17, 16)

→ [x, x, x, x, x, x, x, …,x]

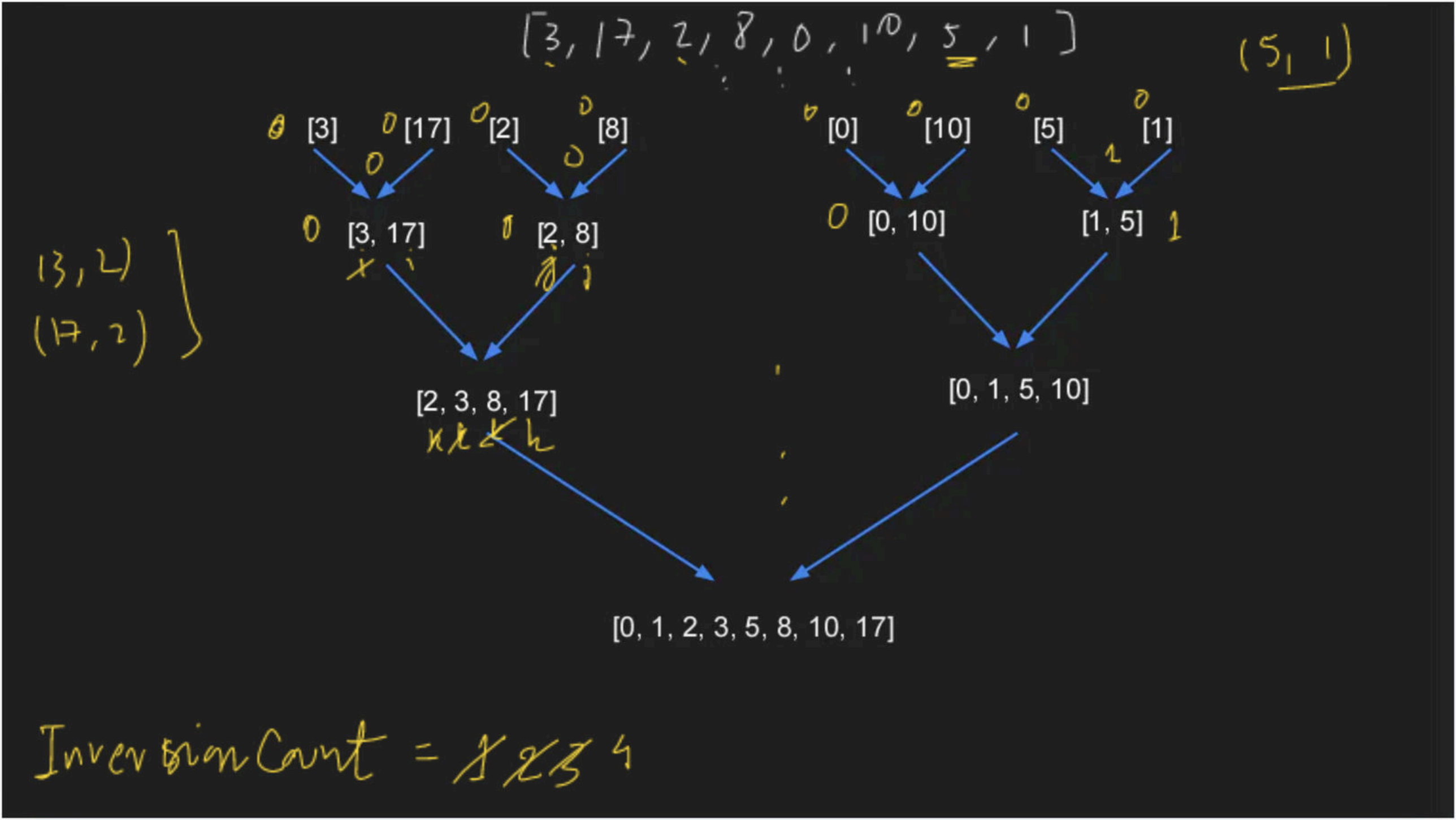
[x, x, x, x, …, x] [x, x, x, x, … , x]

Total inversions =

Inversions_in_left_half + Inversions_in_right_half +
Inversions_formed_by_one_element_in_the_left_and_other_in_the_right

[3, 17, 2, 8, | 0, 10, 5, 1]

[3, 17, 2, 8] [0, 10, 5, 1]


[3, 17, 2, 8, 0, 10, 5, 1]
(5, 1)
[3]
[17]
[2]
[8]
[0]
[10]
[5]
[1]
[3, 17]
[2, 8]
[0, 10]
[1, 5]
(3, 2)
(17, 2)
[2, 3, 8, 17]
[0, 1, 5, 10]
[0, 1, 2, 3, 5, 8, 10, 17]
Inversion Count = 4

Claim:
If (x, y) forms an inversion, i.e. y forms an inversion with x, then y will also form an inversion with every number after x.
OR,
If x occurs at an index *i* in the first array, then y will form mid - i + 1 inversions with the first array.

(3, 2) (5, 2)
(3, 2)

l mid mid+1 r

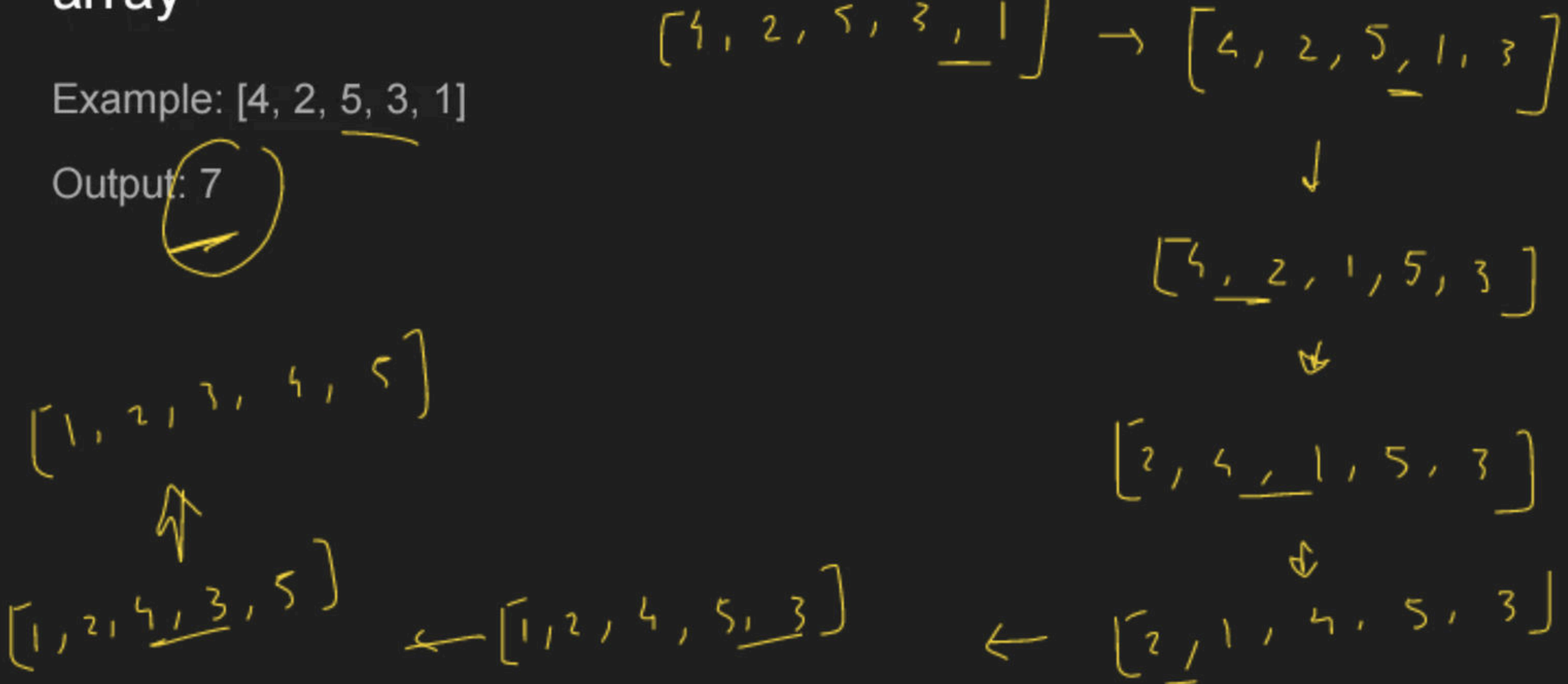
[1, 3, 5, 7] [2, 4, 6, 8]

Since 2 forms an inversion with 3, i.e., (3, 2) is an inversion => 2 will also form an inversion with all numbers after 3 (i.e. 2 forms an inversion with 3, 5 and 7)
So the total number of inversions formed by 2 is (3 - 1 + 1) = 3

Minimum number of adjacent swaps required to sort an array

Example: [4, 2, 5, 3, 1]

Output: 7

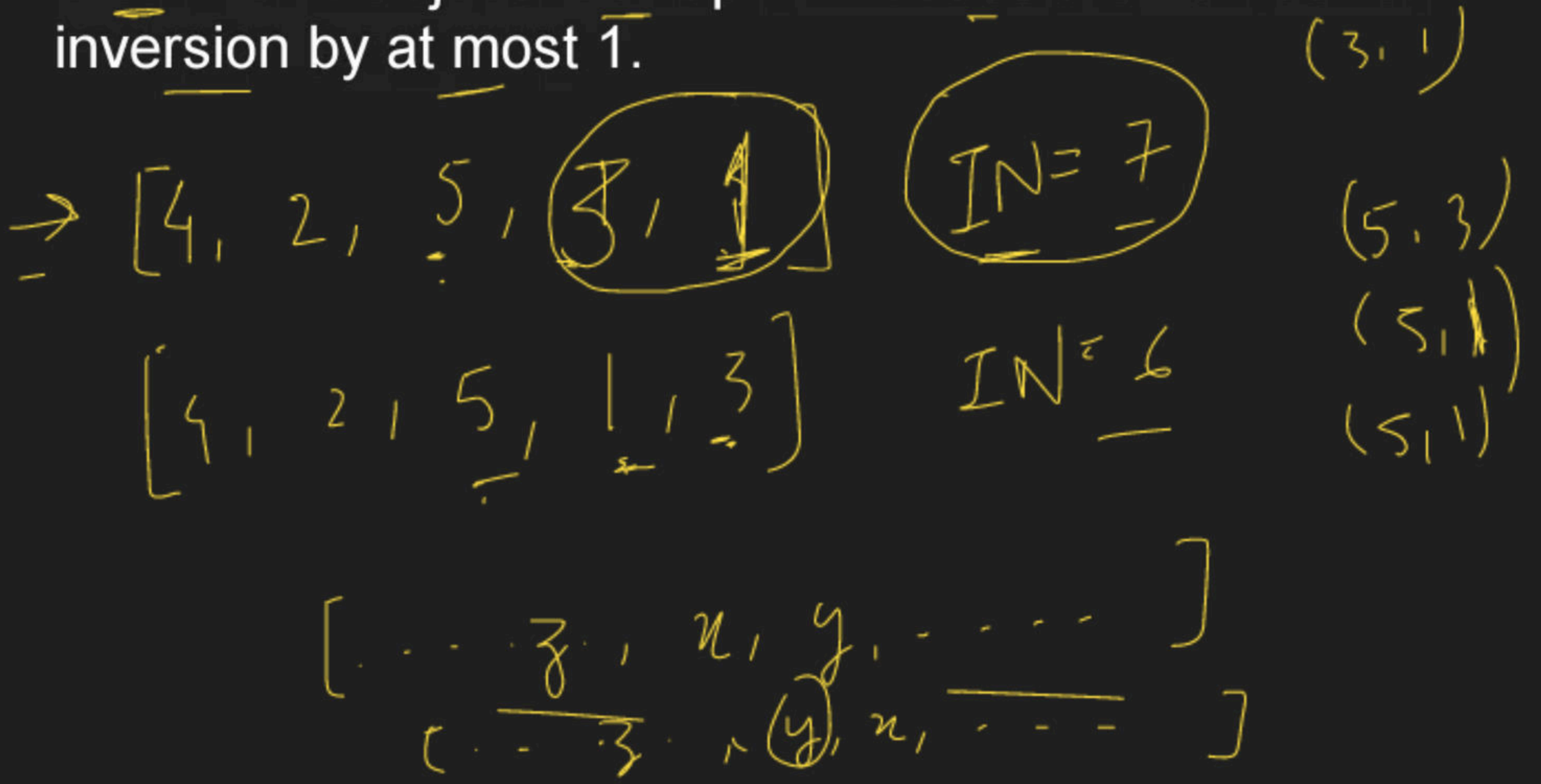


Solution :

Answer = Number of inversions in an array

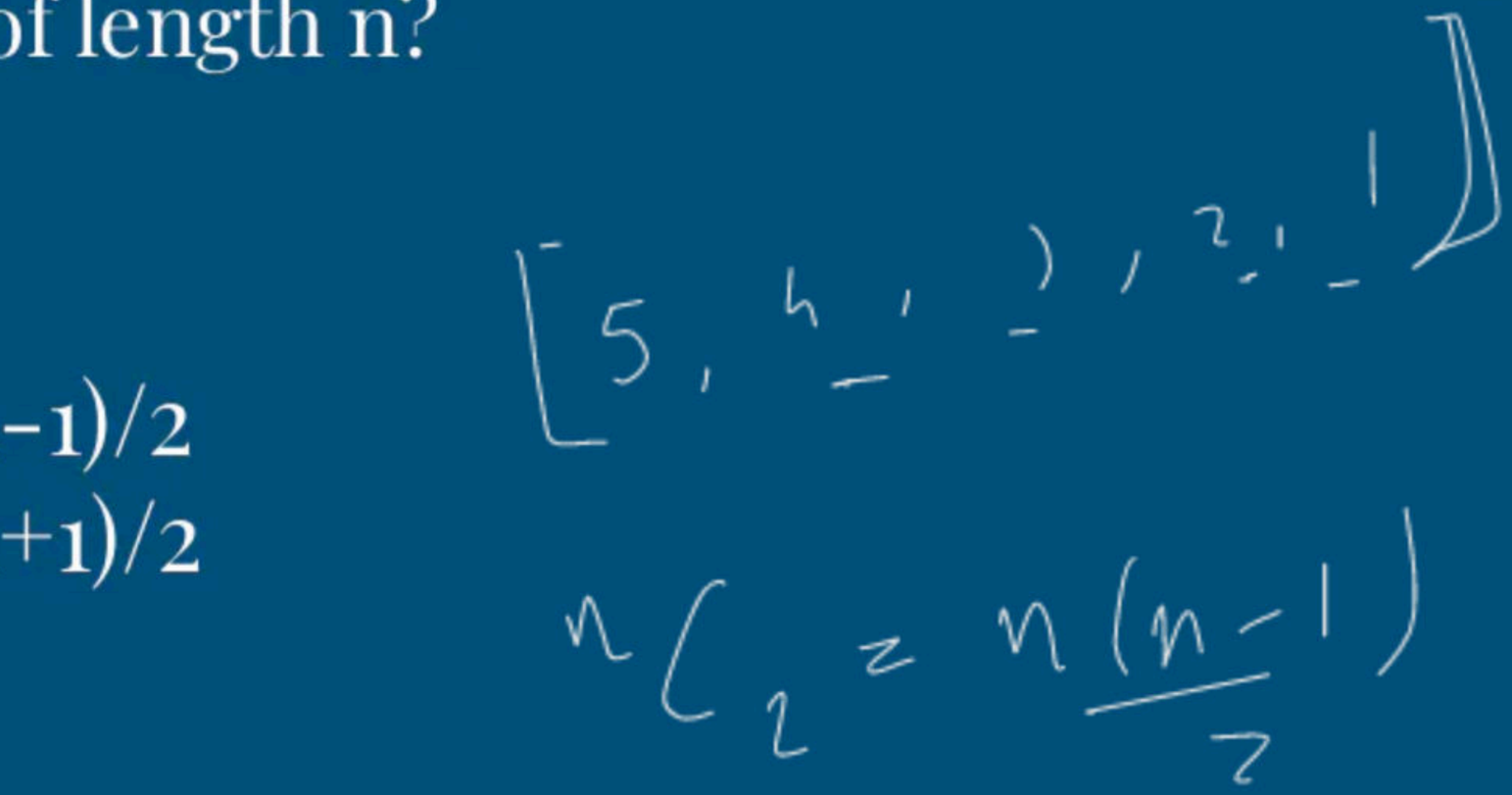
[1, 2, 3, 4, 5] ⇒ IN = 0

Claim: One adjacent swap will reduce the number of inversion by at most 1.



1. What is the maximum number of inversions possible in an array of length n?

A. n^2
B. n*(n-1)/2
C. n*(n+1)/2
D. n+1



A. n^2

B. $n*(n-1)/2$

C. $n*(n+1)/2$

D. $n+1$

Solution : Worst case the array is sorted in decreasing order.

2. Which of the following mentioned algorithms works on the divide and conquer paradigm?

A. Merge Sort
B. Selection Sort
C. Bubble Sort
D. Insertion Sort

A. Merge Sort
B. Selection Sort
C. Bubble Sort
D. Insertion Sort

Solution : Merge Sort works on divide and conquer as discussed in class.

3. You have to sort 1GB of data with only 100MB of main memory available. Which sorting technique will be most appropriate?

A. Bubble Sort
B. Merge Sort
C. Selection Sort
D. Insertion Sort

A. Bubble Sort
B. Merge Sort
C. Selection Sort
D. Insertion Sort

Solution : Merge sort works for external sorting

4. What is the total number of levels in the recursion tree while using merge sort on an array of 8 elements?

A. 1
B. 2
C. 3
~~D.~~ 4

$\lceil \log_2 8 \rceil + 1$

$= 3 + 1 = 4$

A. 1
B. 2
C. 3
D. 4 ✓

Solution : Level -1 => 8

Level-2 => 4+4

Level-3 => 2+2+2+2

Level-4 => 1+1+1+1+1+1+1+1

5. What is the worst case time complexity of merge sort?

A. O(n)
B. O(nlogn)
C. $O(n^2)$
D. $O(\sqrt{n})$

A. O(n)

B. O(nlogn)

C. $O(n^2)$

D. $O(\sqrt{n})$

Solution : Derived from recurrence relation

$$T(n) = 2*T(n/2)+O(n)$$

6. Which of the following algorithms works best for sorting arrays?

A. Bubble Sort
B. Selection Sort
C. Merge Sort
D. Insertion Sort

A. Bubble Sort
B. Selection Sort
C. Merge Sort
D. Insertion Sort

Solution : Worst case for merge sort is O(nlogn) while for others it is $O(n^2)$

Practice/HW

1. Given an array of length n containing both positive and negative elements, find a way to segregate the positive and negative elements together such that the relative order of the similar parity elements remains the same.
2. https://codeforces.com/contest/580/problem/B