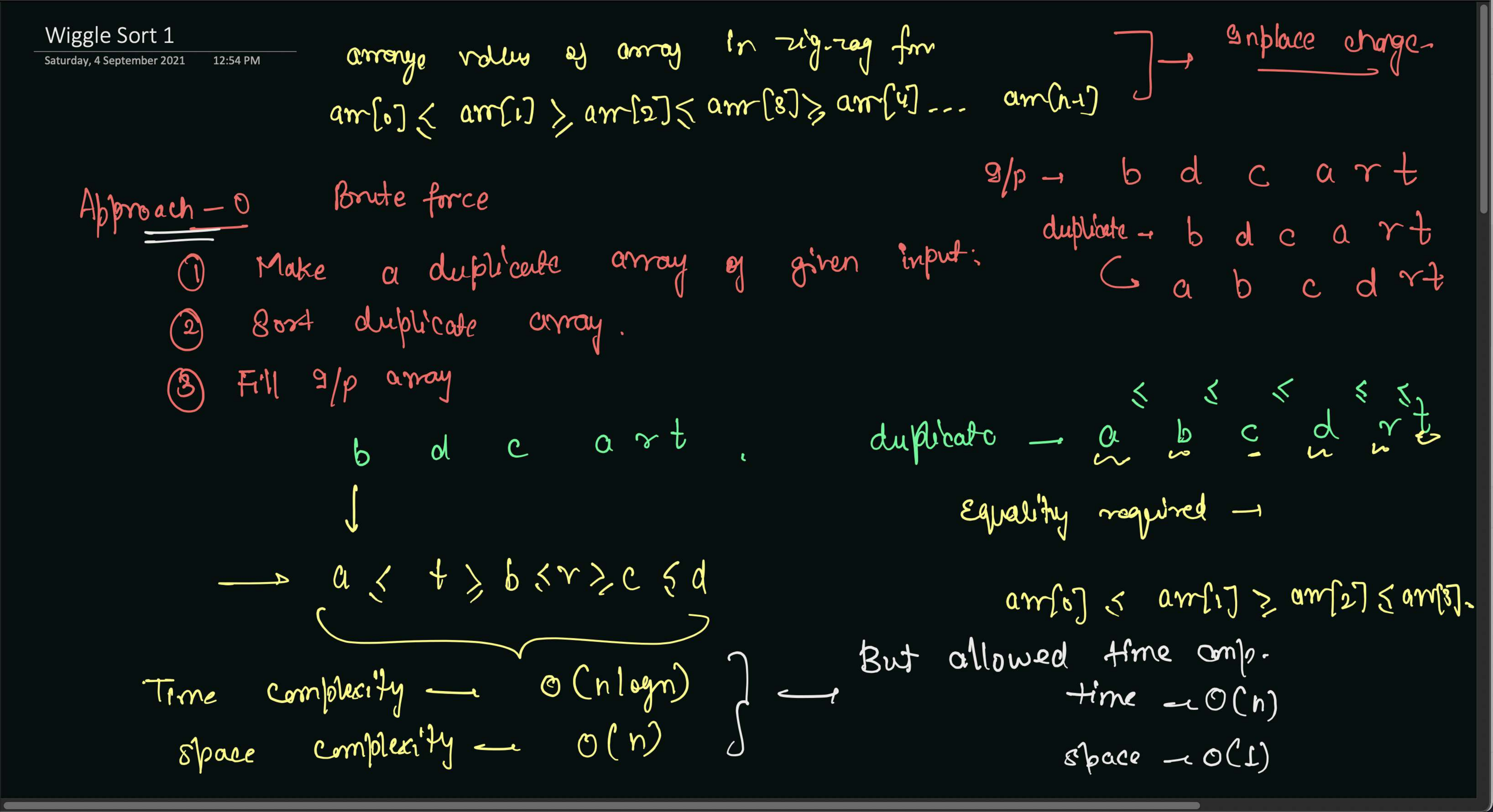


Wiggle Sort 1

Saturday, 4 September 2021 12:54 PM

arrange values of array in zig-zag form

$arr[0] \leq arr[1] \geq arr[2] \leq arr[3] \geq arr[4] \ldots arr(n-1)$] → Inplace change

I/p → b d c a r t

duplicate → b d c a r t

↳ a b c d r t

Approach - 0 Brute force

1. Make a duplicate array of given input.
2. Sort duplicate array.
3. Fill I/p array

b d c a r t .

↓

→ $a \leq t \geq b \leq r \geq c \leq d$

duplicate → $\overset{\leq}{a}$ $\overset{\leq}{b}$ $\overset{\leq}{c}$ $\overset{\leq}{d}$ $\overset{\leq}{r}$ t

Equality required →

$arr[0] \leq arr[1] \geq arr[2] \leq arr[3]$.

Time complexity → O(n log n)

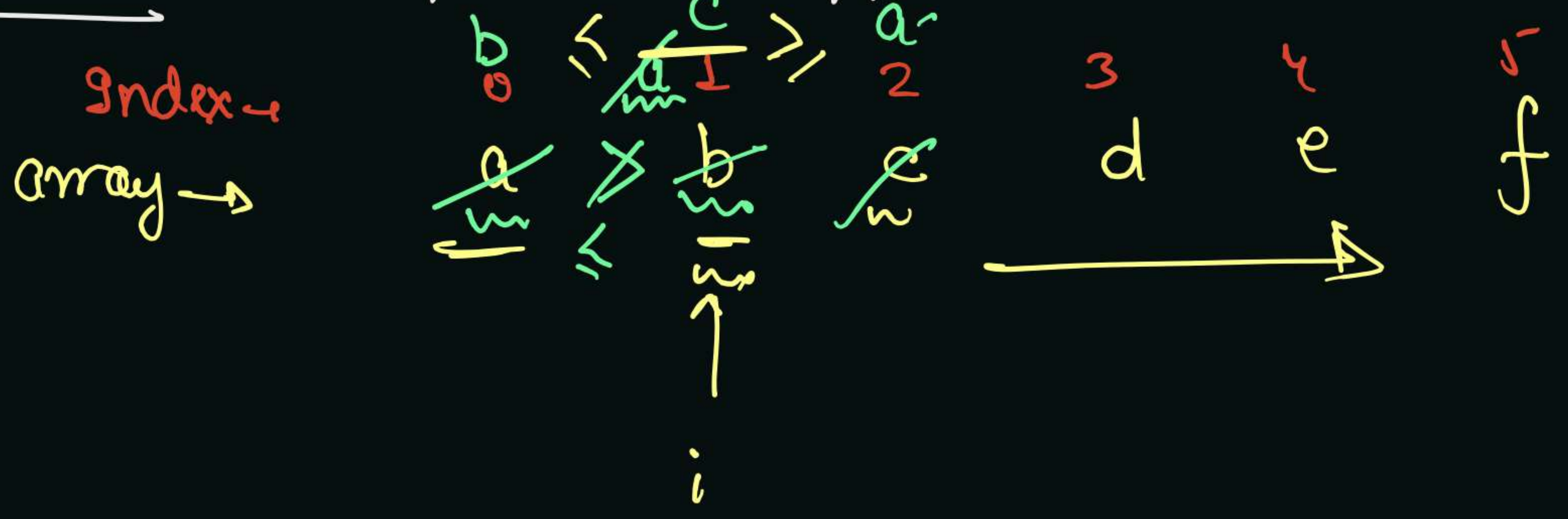
space complexity → O(n)

← But allowed time comp.

time → O(n)

space → O(1)

Approach-1 - optimised approach-



$a > b$

$c > a$

possible Equalities

# odd Index Elements are greater than equal to previous and next Element

odd Index

```
if(arr[i] < arr[i+1]) {
    swap (arr, i, i+1);
}
```

$arr[i] \geq arr[i+1]$

nothing to do

Even Index

```
if ( arr[i+1] < arr[i]){
    swap(arr, i, i+1);
}
```

$arr[i+1] \geq arr[i]$

nothing to do

How previous equalities is unchanged.

→ $a > b$

→ $c > a$.

$b \leq a < c$ problematic equality

$a > b$ and $c > a$ → $b < a < c$

→ $c > b$ $b < c$

$$\begin{array}{ccccccccccc} 0 & & 1 & & 2 & & 3 & & 4 & & 5 \\ 3 & \le & 5 & \ge & 1 & \le & 6 & \ge & 2 & \le & 4 \end{array} \longrightarrow arr[0] \le arr[1] \ge arr[2] \le arr[3] \ldots$$

$$arr[0] \le arr[1] \ge arr[2] \le arr[3] \ge arr[4] \le arr[5] \longrightarrow ||$$

odd Index

```
if( arr[i]< arr[i+1]) {
    swap(arr, i, i+1);
} else {
    // Nothing todo
}
```

Even Index

```
if(arr[i]> arr[i+1]){
    swap(arr, i, i+1)
} else {
    // Nothing to do
}
```

Reverse sorted and contain duplicates. →

$$\begin{array}{ccccccccccccccccccc} 0 & & 1 & & 2 & & 3 & & 4 & & 5 & & 6 & & 7 & & 8 & & 9 \\ 4 & \le & 5 & \ge & 3 & \le & 4 & \ge & 2 & \le & 3 & \ge & 1 & \le & 1 & \ge & 1 & \le & 1 \end{array}$$

$$\longrightarrow \quad 4 \le 5 \ge 3 \le 4 \ge 2 \le 3 \ge 1 \le 1 \ge 1 \le 1$$

Wiggle sort 2

Saturday, 4 September 2021 7:17 PM

time → O(nlogn)

Space → O(n)

arr[0] < arr[1] > arr[2] < arr[3] - - - - -

Steps:

① sort →

② place first and last, first ++, last -- ✗

after sorting → 1 3 4 5 5 5 6 7 8 9

wiggle sort → 1 9 3 8 4 7 5 6 5 5

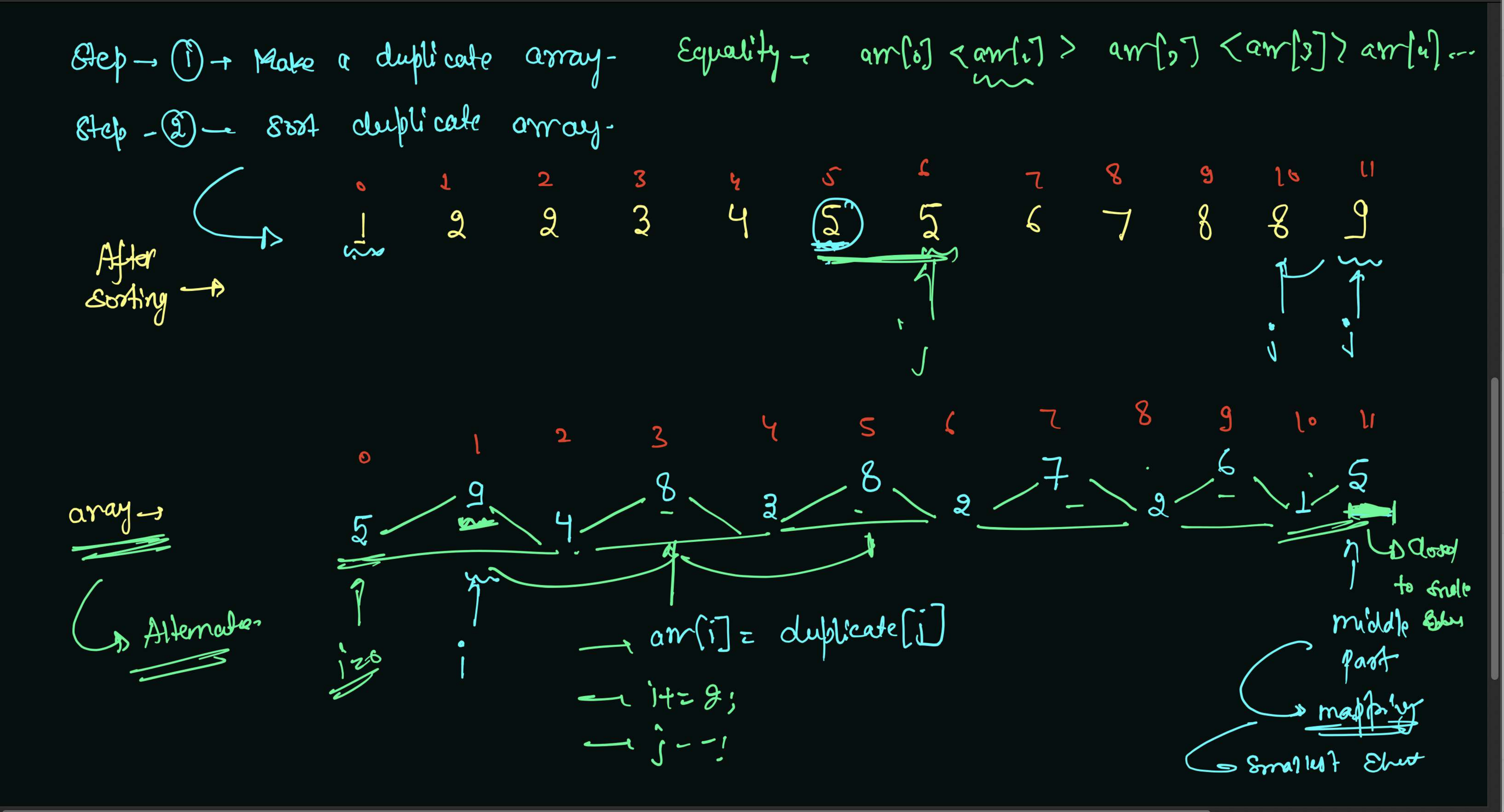
same Element

1 < 9 > 3 < 5 > 4 < 7 > 5 < 6 > 5 < 8 ✓

✗

What Not work ?? →

① sort, ② make left and Right pointer, ③ Place in Res



Step → ① → Make a duplicate array-   Equality → arr[0] < arr[1] > arr[2] < arr[3] > arr[4] ....

Step - ② → Sort duplicate array-

After sorting →

0	1	2	3	4	5	6	7	8	9	10	11
1	2	2	3	4	5	5	6	7	8	8	9

array →

0	1	2	3	4	5	6	7	8	9	10	11
5	9	4	8	3	8	2	7	2	6	1	5

Alternate

→ arr[i] = duplicate[j]

→ i+=2;

→ j--!

Closed to find middle elem part

mapping

Smallest Elem

Number of Subarray With Bounded Maximum

Saturday, 4 September 2021 7:56 PM

left → 2 right 3

array → 2 1 4 3

max of subarray

Total no. of Subarray = $\frac{n(n+1)}{2}$

$= \frac{\cancel{4}^{2} \times 5}{\cancel{2}} = 10$

all possible subarray →

2 → 2 ✓

2 1 → 2 ✓

2 1 4 → 4 ×

2 1 4 3 → 4 ×

1 → 1 ×

1 4 → 4 ×

1 4 3 → 4 ×

4 → 4 ×

4 3 → 4 ×

3 → 3 ✓

No. of subarrays such that Max of subarray is in Range [left, Right] included

Answer = 3

[2], [2,1], [3]

3

array → 2 4 3 | 10 | 6 1 7 | 9 | 3 1 4 8 | 11 | 3

j

i 6 1 7 → 7

Continuous

Significance of j → no. of possible subarray Ending at j

left → 2

Right → 8

~~lo~~ = 3 , hi = 6

new Element

[3 1 4 5]

i j

prev. count

Break Point

Ending at j →

previous count → prev

consider array part from i to j-1

↳ count

overall count ⇒ count

↳ total no. of subarray having max in Range.

Case-I → Left ≤ arr[i] ≤ Right , arr[i] = 5

[3, 1, 4] 5 → 4, 14, 314

0 1 2 3

5, 45, 145, 3145 ⇒ j-i+1

DP → Divide array into k-partition

count += prev.count

no. of subarray

Case-II → arr[i] < left

arr[i] = 1

[3, 1, 4]

last prev..

4, 14, 314

Same count

1 ×

41, 141, 3141

3 sub..

count += prev.count

[3, 6]

2

3 5 0 11

prev count = 2

2 ×

12 ×

112 ×

0 11 2 ×

3 5 0 11 2 2

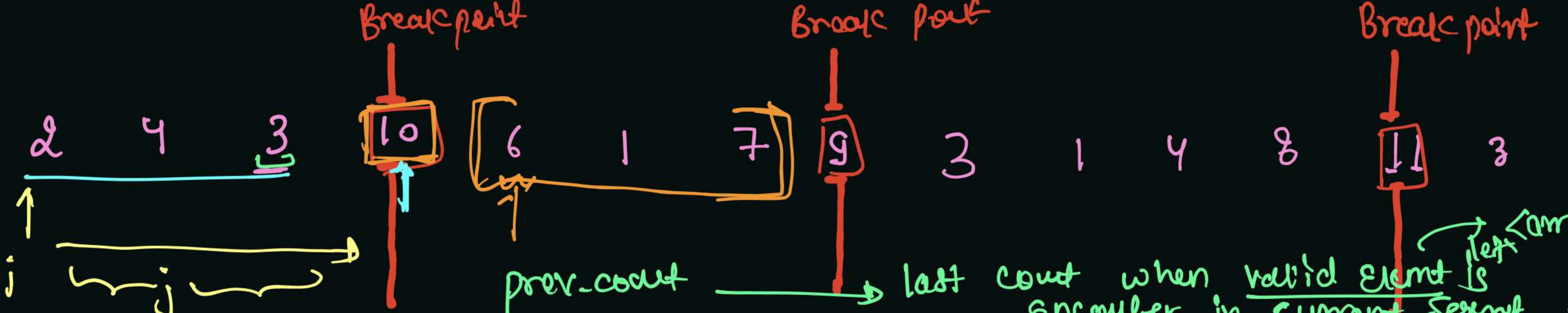
Case III → arr[i] > Right

i = j + 1,

j ++,

array →

~~[3, 8]~~



i → index After break point

j → Running index After i

prev_count → last count when valid Element is encounter in current segment

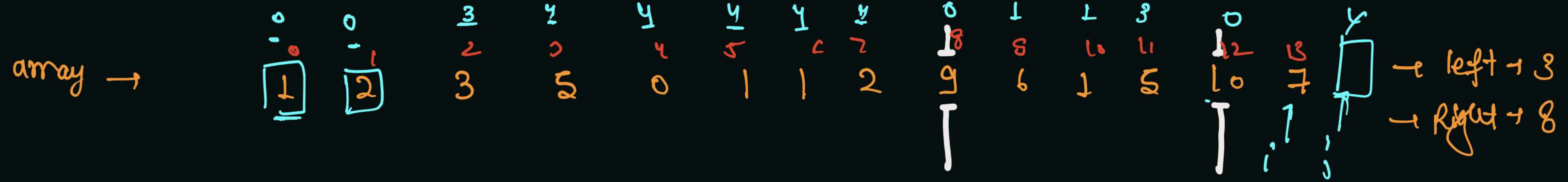
left <= arr[i] <= R

overall count → complete array. no. of valid subarr

j is travelling and add count of subarray in overall count

arr[i] > Right [3, 8]	left <= arr[i] <= Right. [3, 8]	arr[i] < left [3, 8]
[3, 1, 4, 10]	[3, 1, 4, 5]	[3, 1, 4, 2]
arr[j] = 10	arr[j] = 5	arr[j] = 2
i index j i	i j	i j

Prev_count = 0;
i = j+1;
j++;
	No. of possible subarray Ending at j
	5
	4 5
	1 4 5
	3 1 4 5
	prev_count = j-i+1
	overall-count += prev-count
		overall_count += prev_count
		No. of subarrays Ending at j
		2 ✗
		4 2
		1 4 2
		3 1 4 2
		so 3 5 0 1 1 2
		3 5 0
		3 5 0 1
		2
		1 2
		1 1 2
		0 1 1 2
		5 0 1 1 2 ✓
		3 5 0 1 1 2 ✓



prev_count = ~~0~~ ~~2~~. ~~4~~ ~~0~~ ~~1~~ ~~2~~ ~~0~~ 1

overall_count = ~~0~~ ~~3~~ ~~7~~ ~~11~~ ~~15~~ ~~19~~ ~~23~~ ~~24~~ ~~25~~ ~~28~~ 29

Index 8

6

Index 9

6 1

Index 11

5
1 5
6 1 5

Index 13

7

[3, 8]

Index 2

3
2 3
1 2 3

Index 3

5
3 5
2 3 5
1 2 3 5

Index 4

5 0
3 5 0
2 3 5 0
1 2 3 5 0

Index 5

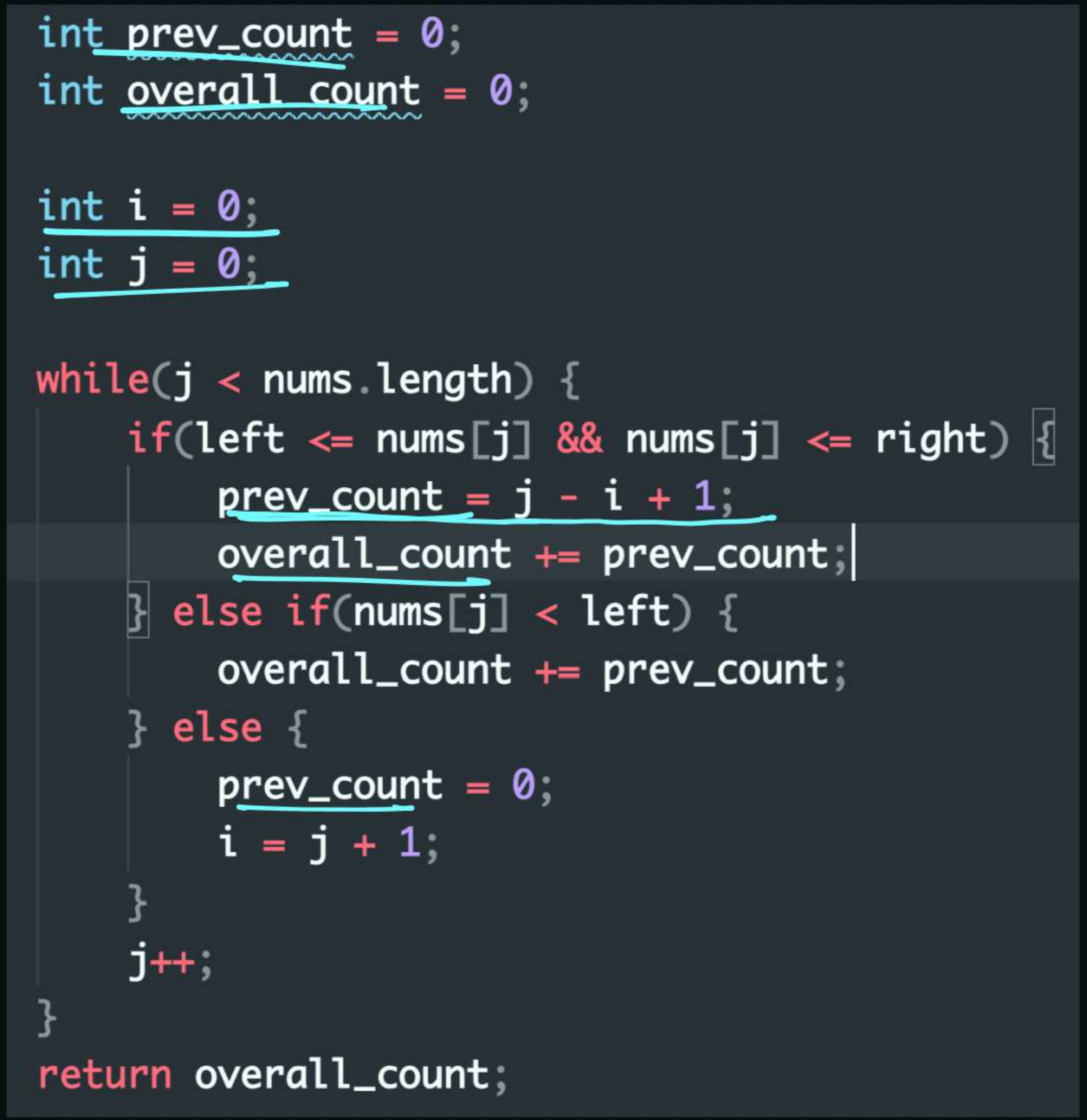
5 0 1
3 5 0 1
2 3 5 0 1
1 2 3 5 0 1

Index 6

5 0 1 1
3 5 0 1 1
2 3 5 0 1 1
1 2 3 5 0 1 1

Index 7

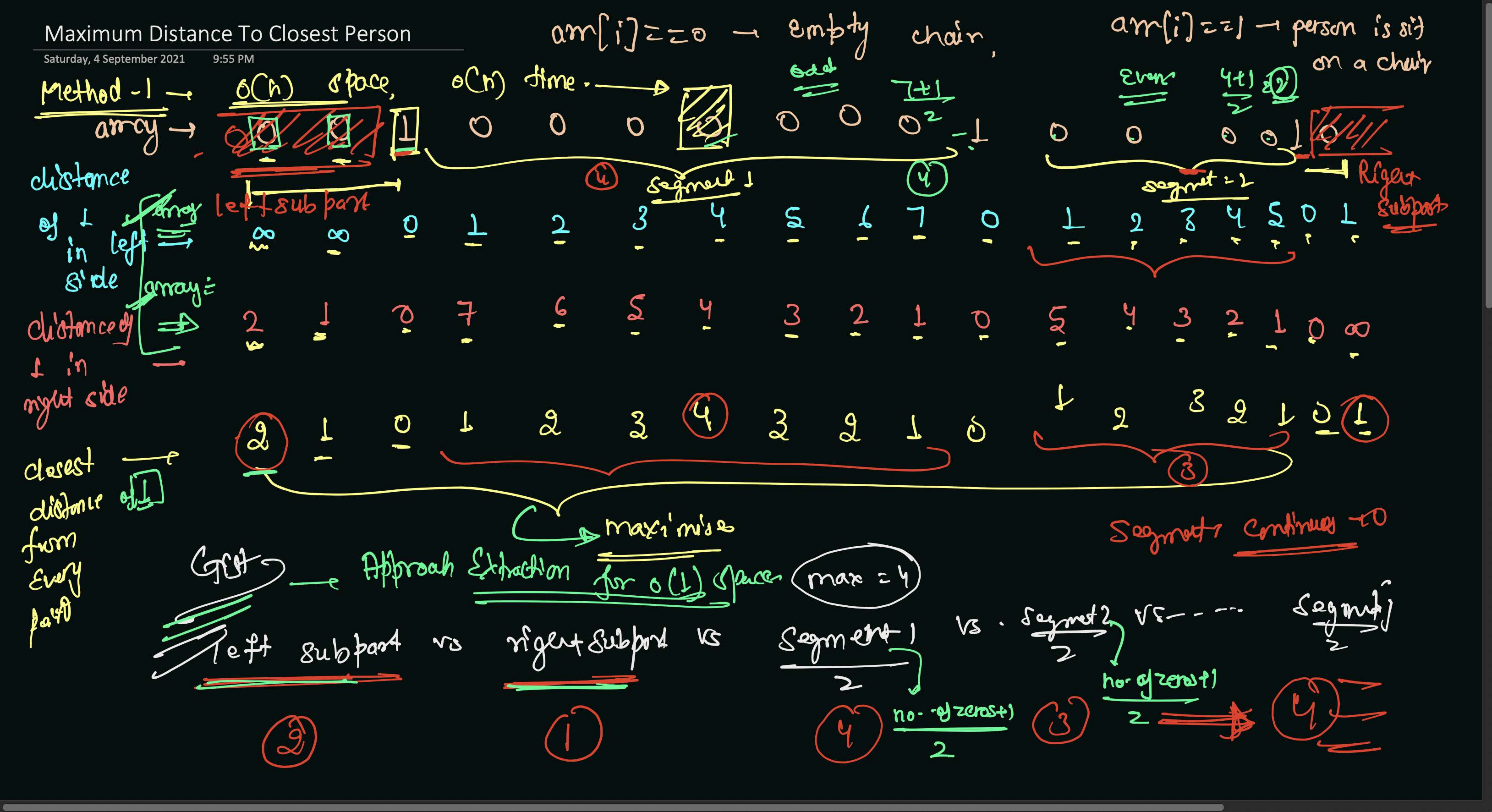
5 0 1 1 2
3 5 0 1 1 2
2 3 5 0 1 1 2
1 2 3 5 0 1 1 2



```
int prev_count = 0;
int overall_count = 0;

int i = 0;
int j = 0;

while(j < nums.length) {
    if(left <= nums[j] && nums[j] <= right) {
        prev_count = j - i + 1;
        overall_count += prev_count;
    } else if(nums[j] < left) {
        overall_count += prev_count;
    } else {
        prev_count = 0;
        i = j + 1;
    }
    j++;
}
return overall_count;
```



Maximum Distance To Closest Person

Saturday, 4 September 2021 9:55 PM

arr[i] == 0 → empty chair.

arr[i] == 1 → person is sit on a chair

Method - 1 → O(n) space, O(n) time. →

odd 7+1/2

Even 4+1/2 = 2

array → 0 0 1 0 0 0 0 0 0 0² 1 0 0 0 0 1 0

segment 1 (4) (4)

segment = 2

Right subpart

left sub part

distance of 1 in left side ~~array~~ ⇒ ∞ ∞ 0 1 2 3 4 5 6 7 0 1 2 3 4 5 0 1

array = distance of 1 in right side ⇒ 2 1 0 7 6 5 4 3 2 1 0 5 4 3 2 1 0 ∞

closest distance of 1 from every part → 2 1 0 1 2 3 4 3 2 1 0 1 2 3 2 1 0 1

(3)

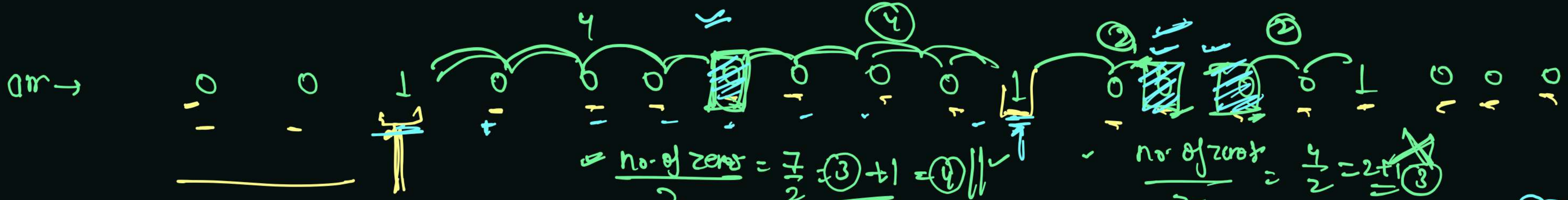
maximise

Segments continues to 0

Gist → Approach Extraction for O(1) spaces max = 4

Left subpart vs right subpart vs Segment 1 / 2 vs . Segment 2 / 2 vs ---- Segment j / 2

(2) (1) (4) no. of zeros+1 / 2 (3) no. of zeros+1 / 2 ⇒ (4)



$\frac{\text{no. of zeros}}{2} = \frac{7}{2} = 3 + 1 = 4$

$\frac{\text{no. of zeros}+1}{2} = \frac{7+1}{2} = 4$

$\frac{\text{no. of zeros}}{2} = \frac{4}{2} = 2 + 1 = 3$

$\frac{\text{no. of zeros}+1}{2} = \frac{4+1}{2} = 2$

2

2 vs. vs $\boxed{\frac{7+1}{2}}$ 4 vs vs $\frac{4+1}{2}$ 2 vs VS 3 3

distance = ~~0~~ 2

zeros = ~~0~~ ~~1~~ ~~2~~

~~0~~ ~~1~~ ~~2~~ ~~3~~ ~~4~~ ~~5~~ ~~6~~ ~~7~~ 0

~~0~~ ~~1~~ ~~2~~ ~~3~~ ~~4~~ 0

~~0~~ ~~1~~ ~~2~~ 3

max = 4

$\frac{\text{no. of zeros}+1}{2}$

$\frac{\text{no. of zeros}}{2} = \frac{\text{distance}+1}{2}$

out of loop

fixes

2000