Rotate Array By k

Ex :- arr[] = (3, -2, 1, 4, 6, 9, 8) , k = 3

We have to rotate the array by k (3) places

let's observe initial input and final output once,

Input :- (3, -2, 1, 4, 6, 9, 8)

Output :- (6, 9, 8, 3, -2, 1, 4)

Observations

① if k = 3 → input's last 3 elements came to starting & became first 3 Elements

(Sequence also not changed)

Input :- (3, -2, 1, 4, 6, 9, 8)

Output :- (6, 9, 8, 3, -2, 1, 4)

Observations

① if k = 3 ↓ leave last 3 Elements and se...,

those elements occured after k elements in the output.

☆ In reverse Order

* So... figure out a way to bring last k Elements to first and first (n-k) Elements to back (as reversed)

↓

Our answer array is Set

Algorithm :

1. Reverse whole array
2. Reverse first k-Elements
3. Reverse the elements, which are after k elements

Ex: arr[] = (3, -2, 1, 4, 6, 9, 8) , k = 3

⇓

Step 1: (8, 9, 6, 4, 1, -2, 3) — ① Reverse whole array

⇓

Step 2: (6, 9, 8, 4, 1, -2, 3) — ② Reverse first k-Elements

⇓

Step 3: (6, 9, 8, 3, -2, 1, 4) — ③ Reverse the elements, which are after k elements

This is our final array

* Create a reverse function
  - which takes array and (L, R) → Ranges
    - L, R → from where to where array need to be reversed

* Call the reverse function for 3 times
  1. reverse (arr, 0, n-1) → whole array reversing
  2. reverse (arr, 0, k-1) → first k-Elements
  3. reverse (arr, k, n-1) → last Elements from k

Equilibrium Index

Sum of all elements before i^{th} idx = Sum of all elements after i^{th} idx

Ex : $arr[\,] = (\underbrace{-2, 0, 1, 2}_{\text{left Sum}}, \overset{\text{Pivot index / Equilibrium index}}{3}, \underbrace{0, 1}_{\text{rightSum}})$

$\quad\quad\quad

Case:-

if (i == 0)

↳ Similarly, there will be no left sum to check with right sum

So, need to handle it seperately

check :-

$$\{ \underbrace{PS(n-1)}_{\text{right sum}} - \underbrace{PS(0)}_{\text{leftsum}} == 0 \} \longrightarrow$$

if true, we got one Equilibrium index

Product of Array Except itself

Given nums[n], return an array answer, such that answer[i] is equal to the product of all elements of nums except itself (nums[i])

Ex:- nums[] = (1, 2, 3, 4)

⇓

answer[] = (24, 12, 8, 6)

* We have to multiply every index value other than current idx for each index

Algorithm

① Initialize a "Suffix" variable → Suffix = 1

② for (n-1 → 1)

↳ Do, prefixproduct (i) = prefixproduct [i-1] * Suffix

Suffix = Suffix * nums [i]

What's going here?

↳ let's breakdown

① At starting, create a prefixproduct and suffix product of our array

arr[] : (1, 2, 3, 4)

prefix product

PP[] : (1, 2, 6, 24)

SP[] : (1, 24, 12, 4)

suffix product

arr[] : (1, 2, 3, 4)

prefix product

PP[] : (1, 2, 6, 24)

SP[] : (1, 24, 12, 4)

Suffix product

At every index, between (1 to n-2)

See → PP[i-1] × SP[i+1]

= 2 × 4 = 8

Check our answer array, for index we having 3 in input array we will have 8 as answer

for (i==0) → SP(i+1)

for (i==n-1) ↔ PP(i-1)

Creating 2 arrays (Prefix Product and Suffix Product) 🙃

Bad Bro.... Optimize it!!!

Now understand this,

Algorithm

① Initialize a "Suffix" variable → Suffix = 1

② for (n-1 → 1)

Do, prefixproduct[i] = prefixproduct[i-1] * Suffix

Suffix = Suffix * nums[i]

We don't really need 2 arrays, we can do this with one prefixproduct and a variable

(which stores required Suffix product)

* Create a suffix variable
  ↳ initializes to 1

* On every iteration update suffix → By suffix * arr[i]
  ↳ goes from Back (n-2 to 1)

* Make ans for every index as
  ↳ PP[i] = PP[i-1] * Suffix

  Suffix → Sp[i+1] → previously it is

* At last pp[] is our answer

Dry run :

arr[] : (1, 2, 3, 4)

Prefix product [] : (1, 2, 6, 24) , Suffix = 1

i	prefixproduct [i]	Suffix
3	PP(i-1) = 6 Suffix = 1 PP(i-1) * Suffix = 6	Suffix = Suffix * arr[i] = 1 * 4 = 4
2	2 * 4	4 * arr[2] = 12
1	1 * 12	12 * arr[1] = 24
0	24	

at (i==0) → prefixproduct [i] = Suffix

Trapping Rain Water

42. Trapping Rain Water

Solved

Hard | Topics | Companies

Given `n` non-negative integers representing an elevation map where the width of each bar is `1`, compute how much water it can trap after raining.

Example 1:

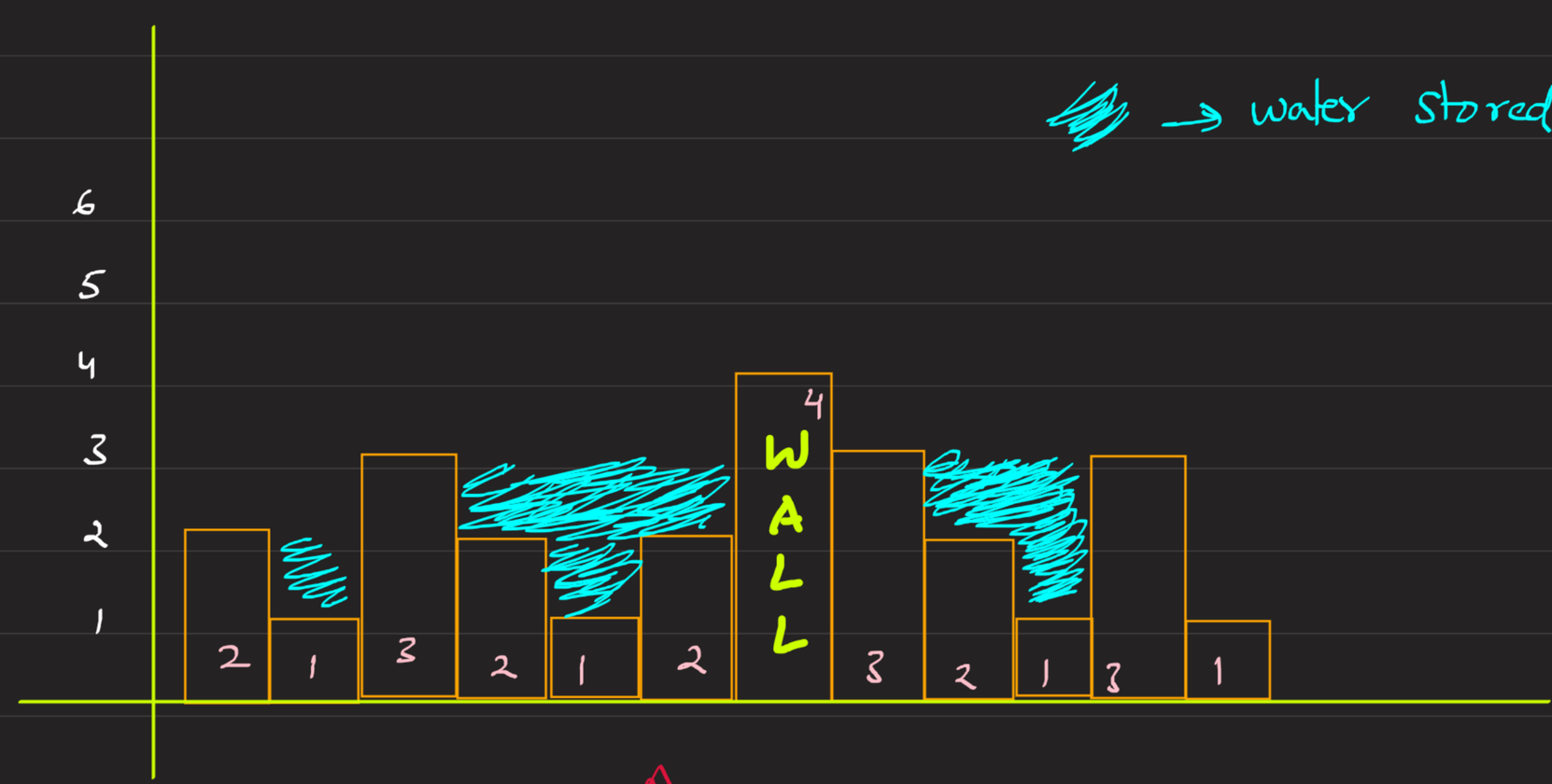
```
Input: height = [0,1,0,2,1,0,1,3,2,1,2,1]
Output: 6
Explanation: The above elevation map (black section) is represented by array [0,1,0,2,1,0,1,3,2,1,2,1]. In this case, 6 units of rain water (blue section) are being trapped.
```

Example 2:

```
Input: height = [4,2,0,3,2,5]
Output: 9
```

arr[] : (2, 1, 3, 2, 1, 2, 4, 3, 2, 1, 3, 1)

→ water stored (trapped)

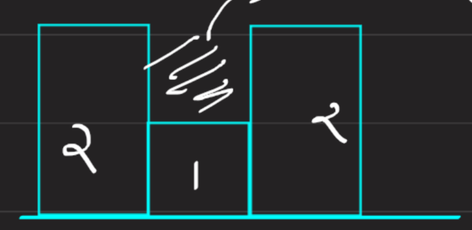


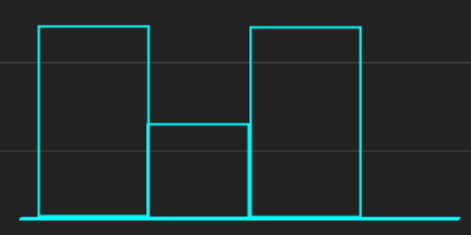
Array values

* Calculate amount on each wall and add them → ans

let's observe some cases

① 1 unit of water

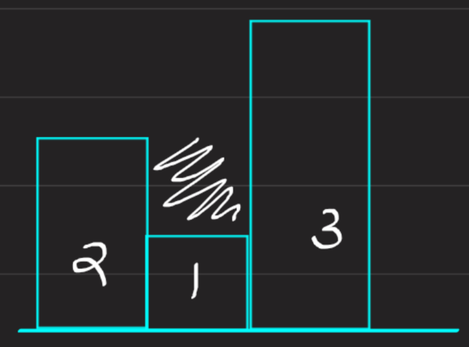




for this wall,

leftBound = 2 (left greatest index)
RightBound = 2 (Right greatest index)

myBound = min(leftBound, rightBound)
= min(2,2)
= 2

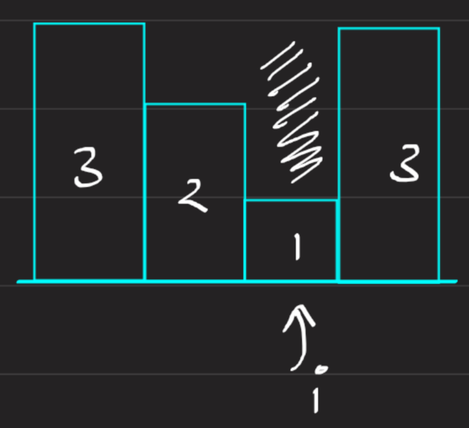


left Bound = 2
right Bound = 3

my limit = min(2, 3) = 2

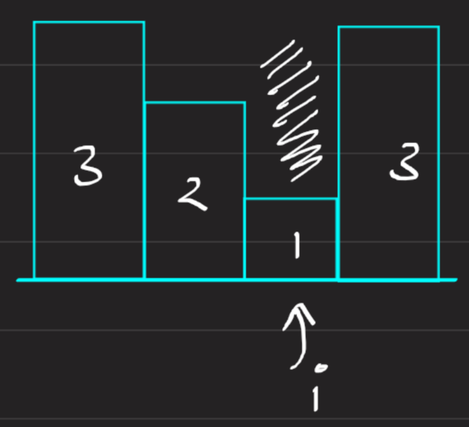
the amount of water i can store = mylimit − my wall size

= 2 − 1 = 1



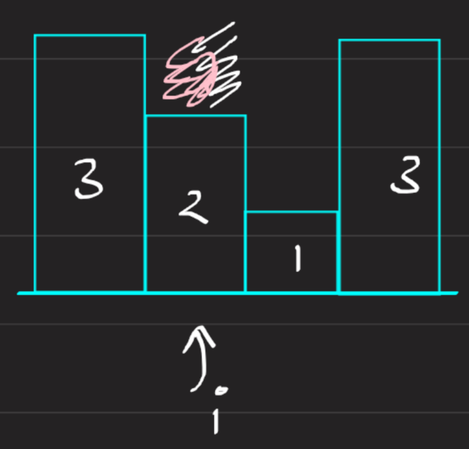
for ith wall → leftBound = 3
rightBound = 3

myAns = Min(lB, rB) − wall
= 3 − 1 = 2 units



for ith wall → leftBound = 3
rightBound = 3

myAns = Min(lB, rB) − wall
= 3 − 1 = 2 units



for ith wall → leftBound = 3
rightBound = 3

myAns = Min(lB, rB) − wall
= 3 − 2 = 1 units

* Therefore, we somehow, need to track of which is the largest building to my left and to my right,

if that's done ⟶ Hard tag becomes Easy 😄

Which store left max height's of building

arr[] : (2, 1, 3, 2, 1, 2, 4, 3, 2, 1, 3, 1)

Prefix Max [] : (2, 2, 3, 3, 3, 3, 4, 4, 4, 4, 4, 4)

Suffix Max [] : (4, 4, 4, 4, 4, 4, 4, 3, 3, 3, 3, 1)

At i^{th} wall,

$rb = Pmax(i-1)$

$lb = Smax(i+1)$

⟶ myBound / mylimit = min(rb, lb)

Amount of water ⟹ ans = ans + (myBound - size of wall)

SubArrays

* for printing all the subarrays

```
for(int sp=0; sp< N ; sp++)
{
    for(int ep=sp; ep< N ; ep++)
    {
        for(int i=sp ; i<=ep ; i++)
        {
            print (arr[i]);
        }
    }
}
```

Starting point

ending point

Subarray with Starting point and ending point

* if you want to print sum of each subarray

① first Create a prefixSum array → psum[ ]

Array

6	9	3	2	1
0	1	2	3	4

if sp = 1, ep = 4

6 | 9 3 2 1

psum[sp-1]

```
for( int sp =0 ; sp< n ; sp++)
{   for(int ep =sp; ep<n; ep++)
    {   if (sp==0)   print (psum[ep])
        else  print (psum[ep] - psum[sp-1])
    }
}
```

This is nothing but psum[ep] - psum[sp-1]

* Run a loop of sp → (0 to n)

ep → (sp to n)

// [sp, ep] → Our valid subarray range

To get sum of all subarrays

Contribution Technique

arr[] : (5, 3, -1, 8)

5			
5 3	3		
5 3 -1	3 -1	-1	
5 3 -1 8	3 -1 8	-1 8	8

Subarrays starting with 5

3 -1 8

Value	Occurance
arr[0] = 5	4 times ⇒ 5 * 4 = 20
arr[1] = 3	6 times → 3 * 6 = 18
arr[2] = -1	6 times → -1 * 6 = -6
arr[3] = 8	4 times → 8 * 4 = 32

Ans = 20 + 18 + (-6) + 32 = 64

Generalizing the approach of value and their occurences

arr[4] : (5, 3, -1, 8)

x_1 x_2 x_3 x_4 (no. of times appeared)

$\therefore$ ans = (arr[0] * x_1) + (arr[1] * x_2) + (arr[2] * x_3) . . . .

6 9 3 Then how to find $x_0, x_1, \ldots$ (no of occurances of each element)

for example

* A subarray which includes (−1) is having :

arr[] : (3, −2, 4, −1, 2, 6)

sp	ep
0	3
1	4
2	5
3	

similarly ← endpoint

(sp : starting point
ep : ending point)

sp=0 (3, −2, 4, −1, 2, 6)

(3, −2, 4, −1, 2, 6) sp=1

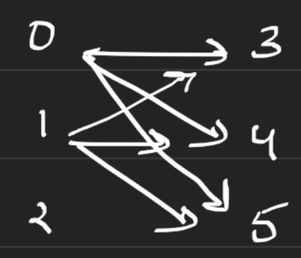
(3, −2, 4, −1, 2, 6) sp=2

(3, −2, 4, −1, 2, 6) sp=3

sp	ep

(sp : starting point
ep : ending point)

3 combinations with 1



3 Combinations with '0'

3 / 4 — 3

Likewise, we total have (4) * (3) combinations (occurences)

∴ if i is '3' → Valid starting points = (i+1) = 4 (index)

Valid ending points = (n−i)

∴ ans = (i+1) * (n−i) → if you want x_1 = (1+1) * (4−1) = 2*3 = 6 for (n=4)

// Pseudo Code

```
int ans =0;
for (int i=0 ; i<n; i++)
{ int occ = (i+1)*(n-i);
  ans += (occ* arr[i]) ;
}
```

x_0
x_1
x_2
x_n

Max Subarray Sum

↳ Kadane's Algorithm

// Problem Statement

Calculate maximum subarray sum

arr[] : (3, 2, -6, 8, 2, 9, 4)

↳ ans = 23

arr[] : (-3, 2, 4, -1, 3, -4, 3)

↳ ans = 8

// idea 1 :

generate all subarray sum's and find out maximum out of them

T.C : $O(n^2)$, SC : $O(n)$

TLE will come bro... learn this algo, we don't have other option 😤

// Kadane's Algorithm

Observations

① (+ve, +ve, +ve, +ve, +ve, +ve)

Our ans will be sum of all

② (-ve, -ve, -ve, +ve, +ve, +ve, +ve)

ans = sum of all positives

③ (+ve, -ve, +ve, -ve, +ve, +ve, +ve)

if these sum < 0 then this is our answer

Algo :

if (any time Sum become < 0)

Update Sum with Current index value

otherwise add arr[present_Index] to Sum

Update Max on every iteration

```java
class Solution {
    public int maxSubArray(int[] nums) {
        int n = nums.length;

        int sum = 0;
        int max = Integer.MIN_VALUE;
        for(int i = 0;i<n;i++){
            sum+=nums[i];
            max = Math.max(sum,max);
            if(sum<0)sum = 0;
        }
        return max;
    }
}
```

increase Sum → Update the max

(Case 3)

whenever the Sum becomes negative our answer will be on right side of the index

So make Sum to 0 and go right

① (+ve, −ve, +ve, −ve, +ve, +ve, +ve)

if these Sum < 0

then this is our answer

Rotate Matrix By 90° (clockwise)

Example 1:

1	2	3
4	5	6
7	8	9

⟹

7	4	1
8	5	2
9	6	3

```
Input: matrix = [[1,2,3],[4,5,6],[7,8,9]]
Output: [[7,4,1],[8,5,2],[9,6,3]]
```

Example 2:

5	1	9	11
2	4	8	10
13	3	6	7
15	14	12	16

⟹

15	13	2	5
14	3	4	1
12	6	8	9
16	7	10	11

```
Input: matrix = [[5,1,9,11],[2,4,8,10],[13,3,6,7],[15,14,12,16]]
Output: [[15,13,2,5],[14,3,4,1],[12,6,8,9],[16,7,10,11]]
```

* Pre-requisites

① Inorder to do this, first learn

→ how to transpose a matrix

Transpose Matrix

	0	1	2	3
0	10	20	30	40
1	50	60	70	80
2	90	100	110	120
3	130	140	150	160

(2,0) (3,1)

⟹

10	50	90	130
20	60	100	140
30	70	110	150
40	60	120	160

(0,2) (1,3)

Observation

* The value in (2,0) ⟶ changed to (0,2)

* The value in (3,1) ⟶ changed to (1,3)

∴ Transpose is nothing but swapping of (i, j) ⟶ (j, i)

⟹ if (0,0) swap's, again (0,0) only comes
(1,1) (1,1)

→ So no problem for diagonals, they stay intact

* We need to iterate only half triangle (Either lower triangle or upper triangle)

* if we travel whole matrix

"if you are at some (x,y) → you do transpose and it turns out to be (y,x)

* And again if you are at (y,x) → you swap right, if you travel whole array

this becomes (x,y) again

* So, travel only one half

```
public void transpose(int[][] arr){
    int n = arr.length;
    for(int i = 0;i<n;i++){
        for(int j = 0;j<i;j++){
            int temp = arr[i][j];
            arr[i][j] = arr[j][i];
            arr[j][i] = temp;
        }
    }

}
```

→ One half

Rotate Matrix By 90°

Input

10	20	30	40
50	60	70	80
90	100	110	120
130	140	150	160

⟹

Output

130	90	50	10
140	100	60	20
150	110	70	30
160	120	80	40

// Approach

Input

10	20	30	40
50	60	70	80
90	100	110	120
130	140	150	160

Output

130	90	50	10
140	100	60	20
150	110	70	30
160	120	80	40

Both are Same

transpose

10	50	90	130
20	60	100	140
30	70	110	150
40	80	120	160

reverse every row

130	90	50	10
140	100	60	20
150	110	70	30
160	120	80	40

* So, inorder to rotate this matrix by 90° (clockwise)

→ we need to transpose first
→ then reverse (each row)

```java
public void reverse(int[][] arr){
    int n = arr.length;
    for(int i = 0;i<n;i++){
        for(int j = 0;j<n/2;j++){
            int temp = arr[i][j];
            arr[i][j] = arr[i][n-j-1];
            arr[i][n-j-1] = temp;
        }
    }
}
```

if arr[] = (1, 2, 3, 4)

↓

if you want to reverse this

(1, 2, 3, 4) → Swap
j n-j-1

(1, 2, 3, 4) → Swap
j n-j-1

* So, you iterate for half only...

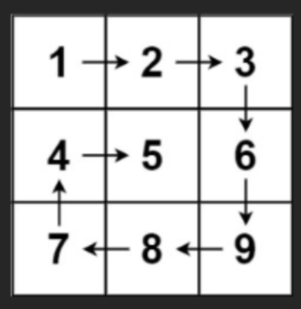
Spiral Matrix

54. Spiral Matrix

Medium | Topics | Companies | Hint

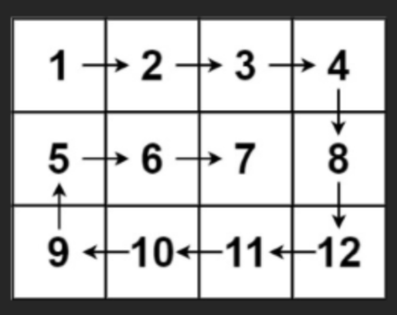
Given an `m x n` `matrix`, return *all elements of the* `matrix` *in spiral order.*

Example 1:




```
Input: matrix = [[1,2,3],[4,5,6],[7,8,9]]
Output: [1,2,3,6,9,8,7,4,5]
```

Example 2:




```
Input: matrix = [[1,2,3,4],[5,6,7,8],[9,10,11,12]]
Output: [1,2,3,4,8,12,11,10,9,5,6,7]
```

In order to learn about Spiral matrix clearly...

* It is necessary to think about Boundary traversal

output:

1 2 3 4 5 6 12 18 24 30

36 35 34 33 32 31 25 19 13 7

1	2	3	4	5	6
7	8	9	10	11	12
13	14	15	16	17	18
19	20	21	22	23	24
25	26	27	28	29	30
31	32	33	34	35	36

reverse

reverse

if it is (6x6)

in output, 5 no's from 0th row

5 no's from last col

5 no's from last row (in reverse)

5 no's from 0th col (in reverse)

name these circled portions → suppose, blue box

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
21	22	23	24	25

→ yellow Box

red box

(5x5)

pink box

* if it is (nxn) size;

(n-1) no's from 0th row

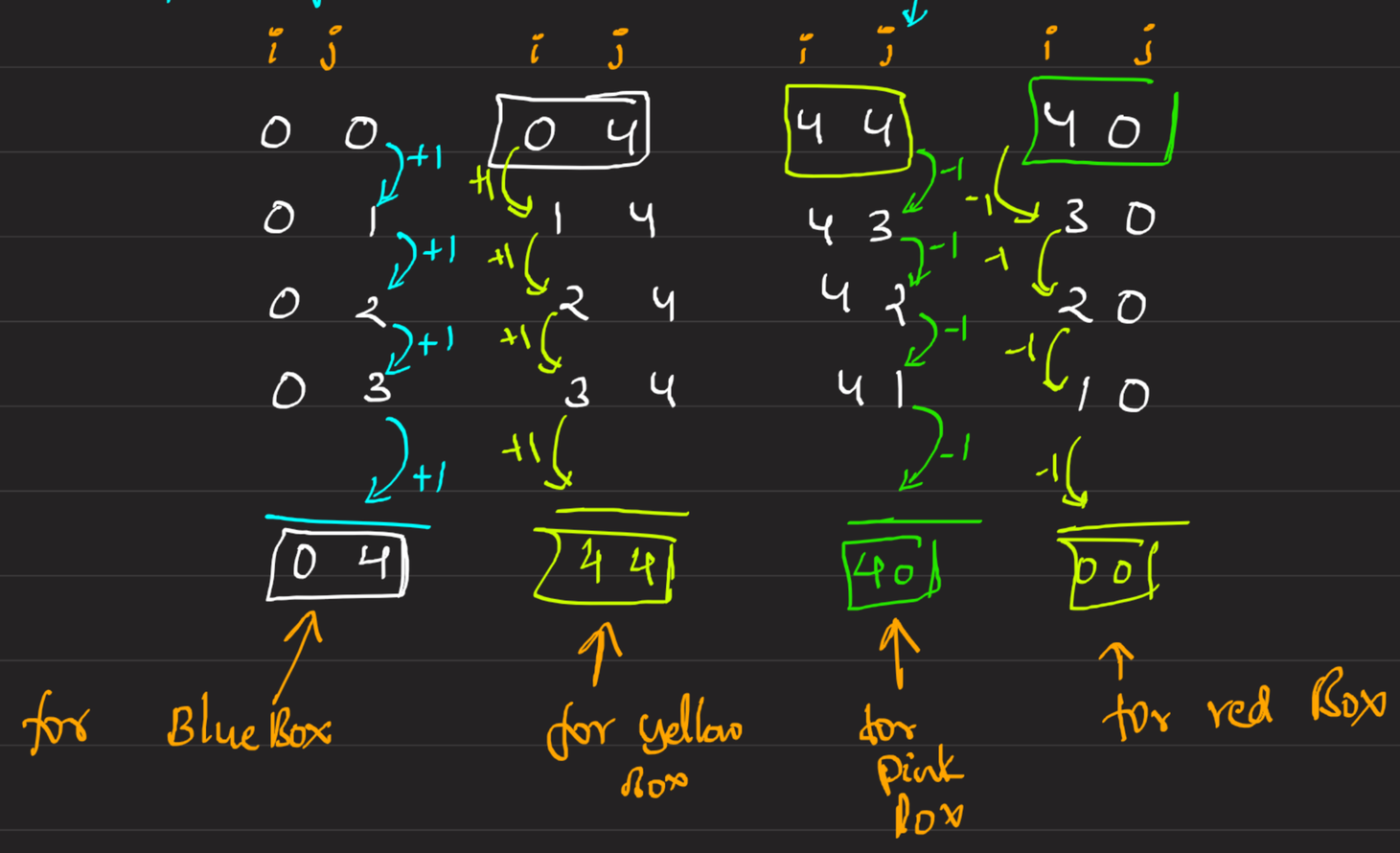
(n-1) no's from last col

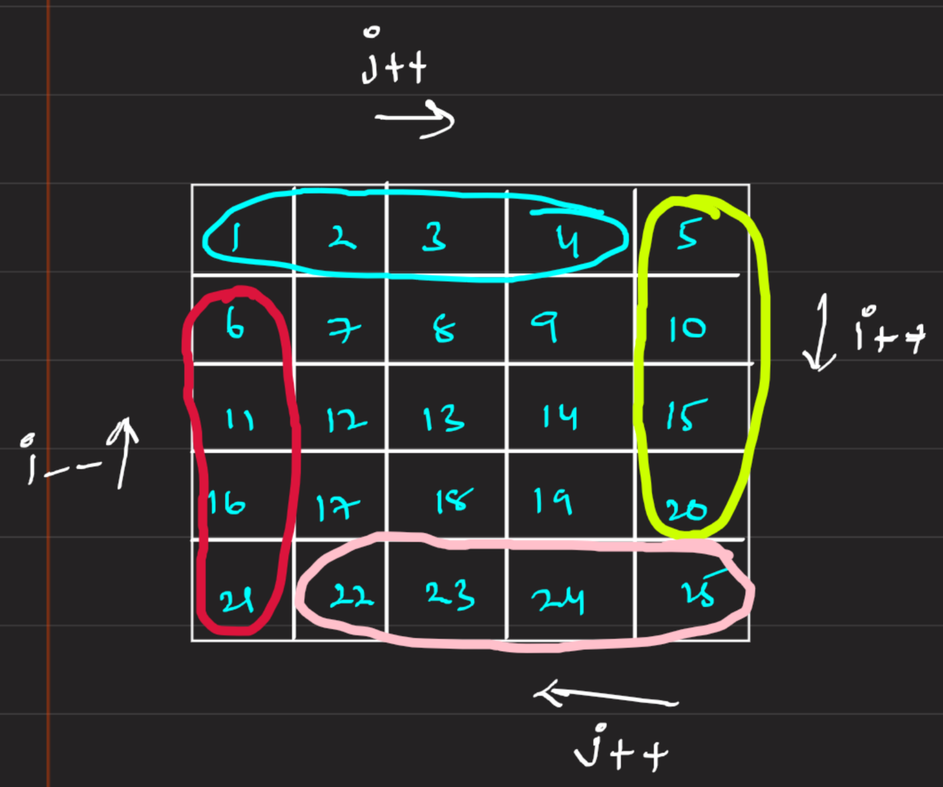
(n-1) no's from last row (in reverse)

(n-1) no's from 0th col in reverse

Observation

printing is done as follows :-





for Blue Box

```
for (int k=0 ; k<(n-1) ; k++)
{
    print (arr[i][j]);
    j++;
}
```

run's for same (n-1) times

for yellow Box

```
for (int k=0 ; k<(n-1) ; k++)
{
    print (arr[i][j]);
    i++;
}
```

for pink Box

```
for (int k=0 ; k<(n-1) ; k++)
{
    print (arr[i][j]);
    j--;
}
```

for red Box

```
for (int k=0 ; k<(n-1) ; k++)
{
    print (arr[i][j]);
    i--;
}
```

Now.... Use this Boundary traversal and Construct Spiral traversal...

direction in every 5 elements we print

	i	j	steps	
first Boundary traversal starts at	0	0	$n-1$	→ for(6x6)
	+1	+1	−2	
Second time from 8	1	1	$n-3$	→ we reduce by 2
	+1	+1		
	2	2	$n-5$	

∴ if you at steps = 5, for (6x6) and do boundary traversal and reduce steps by 2 and again "

till $\boxed{\text{Steps} > 0}$ → we get Our Spiral matrix

```
int i=0, j=0, Steps = n-1;
while (Steps > 0)
{
    // Blue loop
    // yellow loop
    // pink loop
    // red loop

    i++;
    j++;
    step -= 2
}
```

But, But, But....

In our example, we took (n x n) matrix, that's why in every new Boundary traversal, Steps are decreasing by 2

But what if rows & col's are different?

* if (nxn) matrix → then steps are same in rows & col, so we use only one variable to track changes

* if (mxn)

to track row changes → rowsteps (m-1)

→ colsteps (n-1)

* How to update rowsteps, colsteps ?

→ After completing a full Boundary traversal, reduce (rsteps by 2, csteps by 2) & don't forget to increment (i & j)

i j :

i	j	rsteps	csteps
0	0	4	0

1
5
15
20
25

1	2	3
7	8	9
13	14	15
19	20	25
27	28	29

i	j	rsteps	csteps
0	0	4	2
+1	+1	-2	-2
1	1	2	0

In such cases, Either rsteps Or csteps will become zero

* We have to print all the elements of remaining row / column

* refer to code for clear grasp

```
if(rsteps == 0){
    for(int k = 0;k <= csteps;k++){
        ans.add(matrix[i][j]);
        j++;
    }
}else if(csteps == 0){
    for(int k = 0;k <= rsteps;k++){
        ans.add(matrix[i][j]);
        i++;
    }
}
```

→ if rsteps are zero traverse through csteps & print

because we are traversing column → increment j

First Missing Positive Integer

Ex:-

is missing +ve

arr[5] → (3, -2, 1, 2, 7) → ans = 4

arr(7) → (-8, 2, 6, 4, -7, 1, 3) → ans = 5

arr(6) → (2, 1, 6, 4, 3, 5) → ans = 7

arr(5) → (-4, 8, 3, -1, 0) → ans = 1

// Basic - idea

Sort the array & traverse upto last Element

T.C : $O(n\log n) + O(n) \approx O(n\log n)$

Interviewer :- I'm not ok with this Time Complexity, can you optimize it further ?

Me : No, I'm happy with this 😭

Anyway

In Order to think of an Optimal approach, let's look some Observations

① arr[7] :

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

Possible min Ans → 1

Possible max ans → 8

So, for array size = 7 → Ans Range (1, 8)

what if it is ∃n → Ans Range (1, n+1)

Answer range $(1, n+1)$

$\{1, n\}$ — we can see them in our array

$\{n+1\}$ — we can't see this at all,

So, when to return $n+1$ as answer?

$\Rightarrow$ If each corresponding index has its $(+1)$ value, then we can return $(n+1)$

* If our given array is sorted, let's see?

	0	1	2	3	4	5	6	7
arr[8] :	1	2	3	4	5	6	7	8

Each index has its $(+1)$ value at place

* Our main aim is to convert given array into above pattern, from that we can easily get the first missing value by $O(n)$

	0	1	2	3	4	5	6	7
arr[8] :	1	2	3	9	5	6	7	8

if after all possible arrangement we set all indexes perfectly and still some indexes are not in perfect place it means some positive value is missing.

In above example, at 3rd index $\rightarrow$ 4 needs to be there

But 4 not present $\rightarrow$ so first missing element is 4

if before 3rd index, any index faces same issue, then that will be our ans.

as they are acting first missing

So, how to map like →

	0	1	2	3	4	5	6	7
arr[8] :	1	2	3	4	5	6	7	8

By Simple Swapping

	0	1	2	3	4	5	6	7
arr [8] :	4	2	-7	6	9	1	-8	3

arr [0] = 4 , but 4 need to be at index = 3

So, Swap 0th idx & 3rd idx

	0	1	2	3	4	5	6	7
arr [8] :	6	2	-7	4	9	1	-8	3

arr [0] = 6 , but 6 need to be at index = 5

So, Swap 0th idx & 5th idx

	0	1	2	3	4	5	6	7
arr [8] :	1	2	-7	4	9	6	-8	3

Now idx 0 is perfect with its (i+1) value So, increment i

Dry run

arr[0] = 4 ⟶ Actual index for 4 = 3 (Swap (0, 3))

arr[0] = 6 ⟶ Actual index for 6 = 5 (Swap (0, 5))

arr[0] = 1 ⟶ Actual index for 1 = 0 perfect → i++

arr[1] = 2 ⟶ Actual index for 2 = 1 i++

arr[2] = -7 ⟶ Irrelevant ✗ i++

arr[3] = 4 ⟶ Actual index for 4 = 3 i++

arr[ ] = 9 ⟶ Irrelevant i++

⋮

arr[7] = 3 ⟶ Actual index for 3 = 2 (Swap (7, 2))

Another key observation

	0	1	2	3	4
arr[] :	4	1	3	3	2

arr(0) = 4 → Swap (0, 3)

	0	1	2	3	4
arr[] :	~~4~~ 3	1	3	~~3~~ 4	2

arr(0) = 3 → Swap (0, 2)

	0	1	2	3	4
arr[] :	~~4~~ ~~3~~ 3	1	~~3~~ 3	~~3~~ 4	2

↓

now → Same loop runs again n again

We will go to a infinite loop

So, if you about to Swap → check if swapping values are Same or not

* if Same → ignore (i++)
* else → do swap

```
public int firstMissingPositive(int[] nums) {
    int n = nums.length;

    int i = 0;
    while(i<n){
        if(nums[i]<1 || nums[i]>n || nums[i] == i+1){
            i++;
        }
        else{
            int idx = nums[i]-1;
            if(nums[i] == nums[idx])i++;
            else{
                int temp = nums[i];
                nums[i] = nums[idx];
                nums[idx] = temp;
            }
        }

    }
    for(i = 0;i<n;i++){
        if(nums[i]!=i+1)return i+1;
    }
    return n+1;
}
```

Out of range

already set (Perfect placed)

→ if swapping index values are Same, don't swap

else, swap

traverse through the array, & check which index is not perfectly placed

if all are perfect, our answer is (n+1)

Majority Element - I

A majority element is
↳ A element with frequency $> \frac{N}{2}$ → where 'N' is size of array

arr[6] : (1, 2, 1, 6, 1, 1) → freq of '1' = $\left(4 > \frac{6}{2}\right)$

So, 1 is a majority Element

arr[9] : (3, 4, 4, 8, 4, 9, 4, 3, 4) → freq of '4' = $\left(5 > \frac{9}{2}\right)$

So, 4 is a majority Element

Note :

There always exists a majority element, so code accordingly

// idea - 1

use nested loops : $O(n^2)$

// idea - 2

use hashmap to Count : TC : $O(n)$, SC : $O(n)$

As Time Complexity → $O(n)$ is good, but we have an algorithm that can reduce space to $O(1)$

Boyer Moore's Voting Algorithm

If we cancel out 2 distinct elements one by one, the uncancelled (or) remaining element is going to be our majority element

Ex : arr[11] : (3, 3, 4, 6, 1, 3, 2, 5, 3, 3, 3)
(indexes: 0 1 2 3 4 5 6 7 8 9 10)

if a value that is different from Box value, we reduce box frequency count

Value, frequency

i = 0 : 3 - 1 (value 3, occuring 1 time)

i = 1 : 3 - 2

i = 2 : 3 - 1 (Cancel one '3' with 4)

i = 3 : 3 - 0 (Cancel one 3 with 6)

i = 4 : 1 - 1

i = 5 : 1 - 0 (Cancel one 1 with 3)

i = 6 : 2 - 1

i = 7 : 2 - 0 (Cancel one 2 with 5)

i = 8 : 3 - 1

i = 9 : 3 - 2

i = 10 : 3 - 3

```java
public int majorityElement(int[] nums) {
    int n = nums.length;
    int freq = 1;
    int val = nums[0];

    for(int i = 1;i<n;i++){
        if(freq == 0){
            val = nums[i];
            freq = 1;
        }
        else if(val == nums[i])freq++;
        else freq--;
    }
    return val;
}
```

} → At Start, whatever is there in 0th index, store it

} → whenever freq goes to 0, update with new value,

→ if same as box value comes, increase freq

if different value comes, decrement count (distinct from box value)

* for understand, see at (i==3) & (i==4)

from (i==3) → the freq become '0', so in (i==4) we assigned 1 as our new majority element with freq = 1

Majority element -02

N is size of array

A majority Element is having a $\left(\text{frequency} > \frac{N}{3}\right)$

* We use same approach - "Boyer's moore voting algo"

(But with additional Components)

* previously if two distinct elements occured, we cancelled them
* Now, if three distinct elements occurs, we will cancel them

Let's split the solution in 2 phases

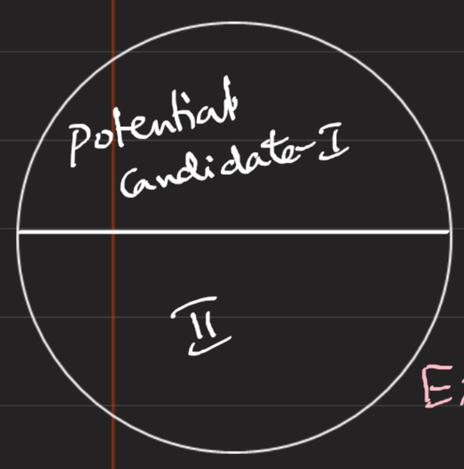
1. Identifying potential candidates
2. Verifying the candidates

1 Identifying potential candidates

* Use 2 variables to store potential Candidates & their counts
* Traverse the array
    1. If one counter is zero, update it with current element and set freq to 1
    2. If current element matches with one of the candidates increment its count
    3. Otherwise, decrement both candidates counts

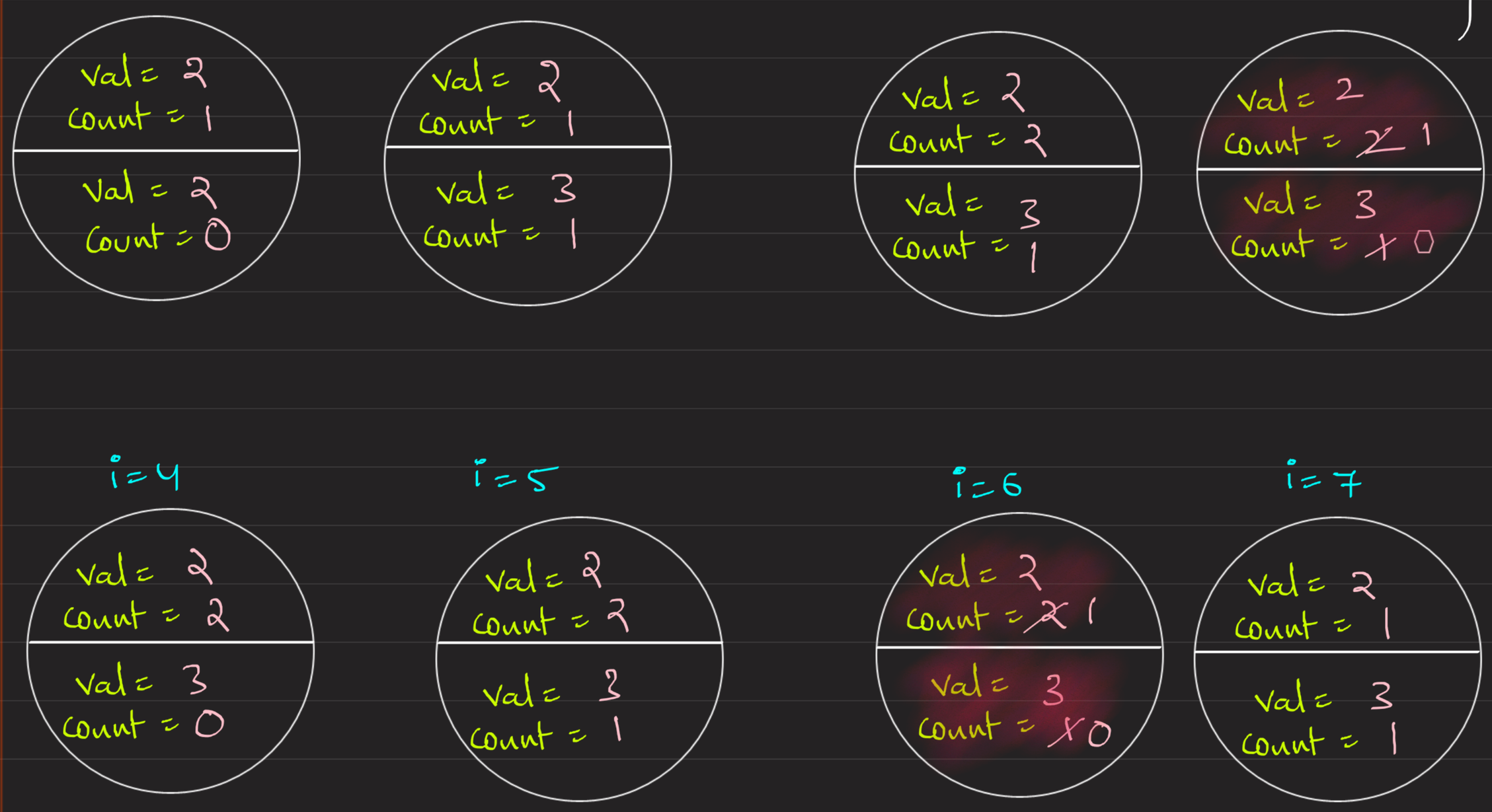
② Verifying the candidates

* Initialize count for the assumed two potential candidates
* Traverse array to count every candidate Occurance
* Check if any of those 2 candidates count is $> \frac{n}{3}$

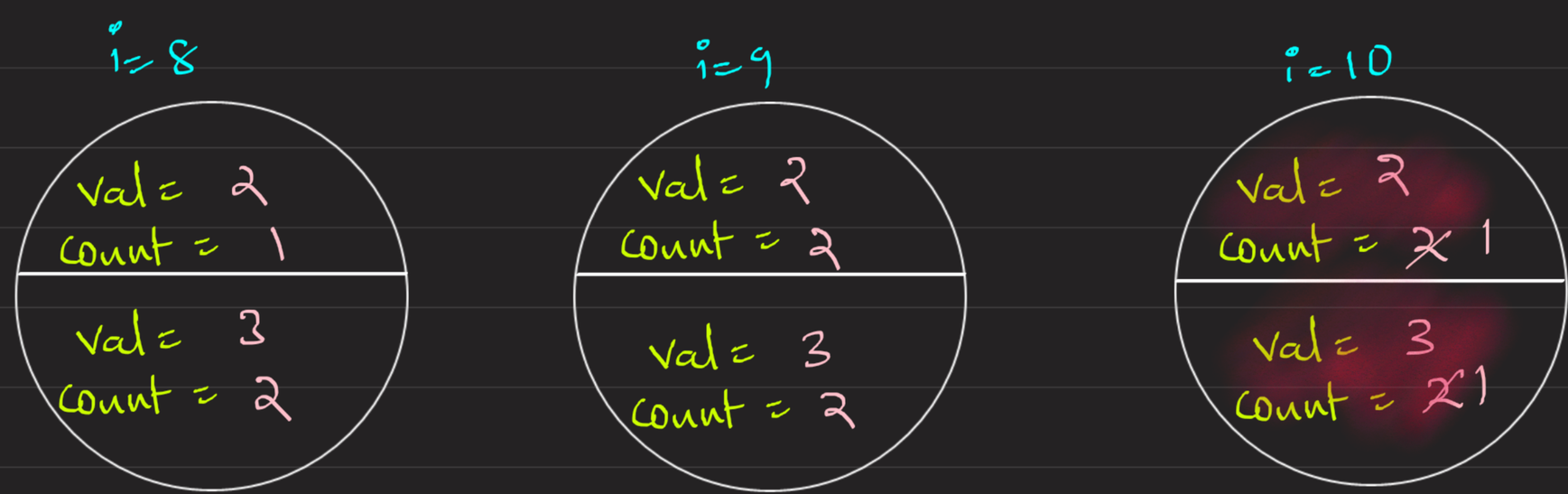


Ex: arr[11] = (2 , 3 , 2 , 4 , 2 , 3 , 5 , 3 , 3 , 2 , 4)
(indexes: 0 1 2 3 4 5 6 7 8 9 10)

if we observe 3 Candidates are distinct, we reduce count



Ex: arr[11] = (2 , 3 , 2 , 4 , 2 , 3 , 5 , 3 , 3 , 2 , 4)
(indexes: 0 1 2 3 4 5 6 7 8 9 10)



* At last, for value = 2, 3 ⇒ we are having counts > 0

So, these are possible candidates for majority elements

Hence, traverse through the array, and check whether {2, 3} are appearing ($> \frac{n}{3}$) times or not

* which ever element from {2, 3} appears ($> \frac{n}{3}$) times that element is a majority element

```
for(int i = 1;i<n;i++){
    if(nums[i] == val1)cnt1++;
    else if(nums[i] == val2)cnt2++;
    else if(cnt1 == 0){
        val1 = nums[i];
        cnt1++;
    }
    else if(cnt2 == 0){
        val2 = nums[i];
        cnt2++;
    }
    else if(val1 != nums[i] && val2 != nums[i]){
        cnt1--;
        cnt2--;
    }
}

int candidate1 = 0;
int candidate2 = 0;

for(int i = 0;i<n;i++){
    if(val1 == nums[i])candidate1++;
    else if(val2 == nums[i])candidate2++;
}
```

→ phase-I (as mentioned starting)

→ phase-II

Check which candidate is $> \frac{N}{3}$ & add it to your answer

Next Greater Element - III

556. Next Greater Element III

Solved

Medium | Topics | Companies

Given a positive integer `n`, find *the smallest integer which has exactly the same digits existing in the integer* `n` *and is greater in value than* `n`. If no such positive integer exists, return `-1`.

Note that the returned integer should fit in **32-bit integer**, if there is a valid answer but it does not fit in **32-bit integer**, return `-1`.

Example 1:

```
Input: n = 12
Output: 21
```

Example 2:

```
Input: n = 21
Output: -1
```

Constraints:

- $1 <= n <= 2^{31} - 1$

we have to use the characters in the given n, and have to make a bigger number than the input,

if Can't form → return (-1)

Ex: (9, 7, 5, 1, 3) → (9, 7, 5, 3, 1) is next bigger number

//idea -1

Iterate from n+1 till you get the answer & Check for every integer if occurances of digits Same or not

Suppose, 38 → is our given array ('3', '8')

Iterate from 39 to .... (83), 84

↳ when you get answer return it which is greater > 38 & occurances are Same

//idea - 2 (Optimal)

let us take an example:

12145 → 12154 ∴ 12154 > 12145

If we want the next greater element of this number, we Swap last 2 digits

For your Info :)

```
char arr[] = (n+"").toCharArray();
int m = arr.length;
```

→ Convert give integer n to char array

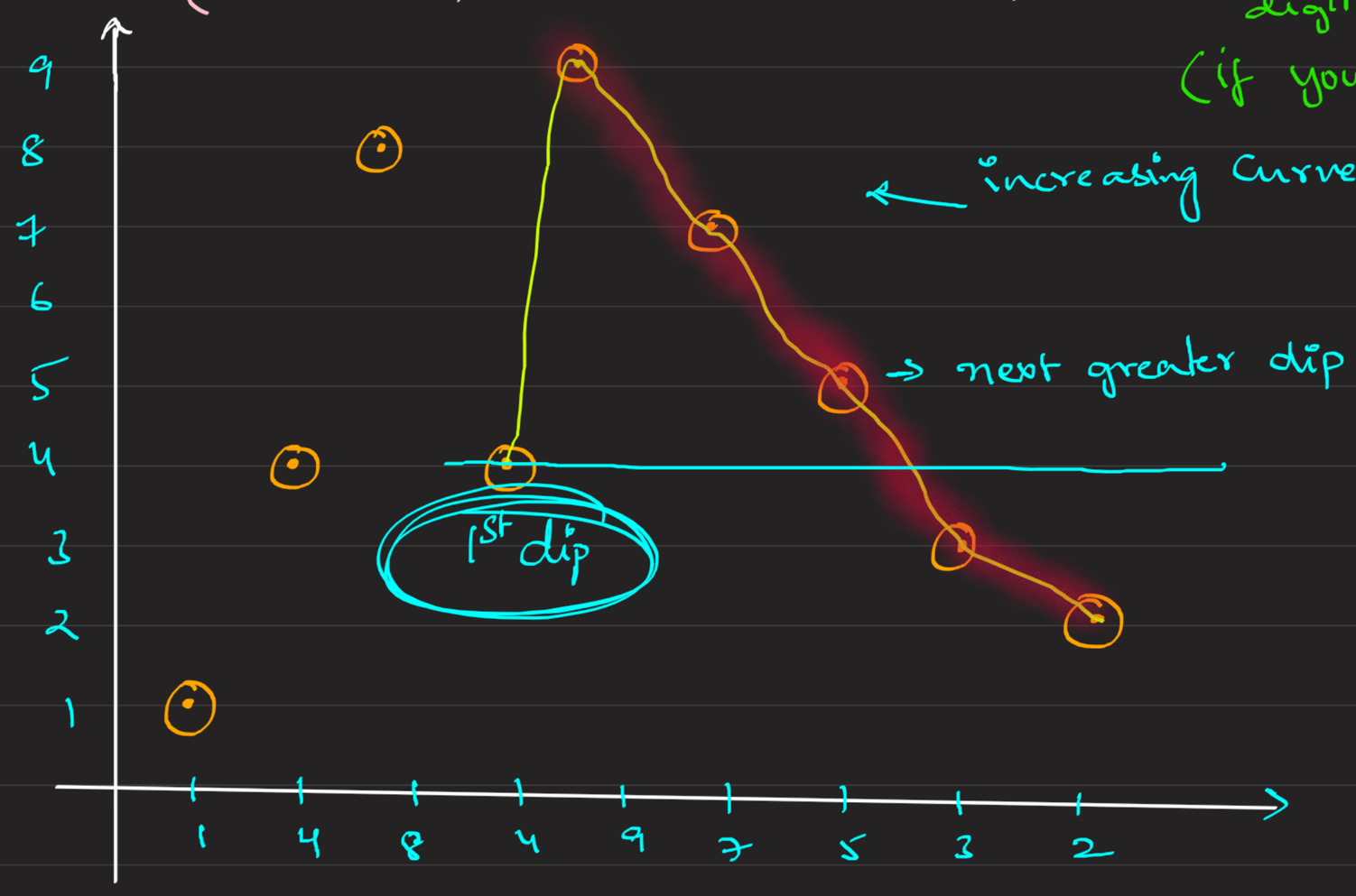
An observation here is,

As we want to do minimum increment, we target smaller place value

and swap them

Ex: (1, 4, 8, 4, 9, 7, 5, 3, 2)

(if you swap any of the digits in this range) your number will get lower



* So, the most important point is ⇒ look for 1st dip & swap it with the next greater number in right side portion

first dip

Ex: (1, 4, 8, 4, 9, 7, 5, 3, 2)

look for next greater element for 4 in right portion

Ex: 1 4 8 4 9 7 5 3 2

9 > 4 ✓

So, 9 may be our answer

1 4 8 4 9 7 5 3 2

7 < 9 & 7 > 4

So, 7 might be our best answer

1 4 8 4 9 7 5 3 2

5 < 7 & 5 > 4

→ Same 5 also

1 4 8 4 9 7 5 3 2

(3 < 4, so before 3, which is 5 is next greater element than 4)

swap(4, 5)

1 4 8 5 9 7 4 3 2

Sort this portion after swap

Our ans will be ready

Max Chunks to make array Sorted -I

769. Max Chunks To Make Sorted

Solved

Medium | Topics | Companies | Hint

You are given an integer array `arr` of length `n` that represents a permutation of the integers in the range `[0, n - 1]`.

We split `arr` into some number of **chunks** (i.e., partitions), and individually sort each chunk. After concatenating them, the result should equal the sorted array.

Return *the largest number of chunks we can make to sort the array.*

Example 1:

```
Input: arr = [4,3,2,1,0]
Output: 1
Explanation:
Splitting into two or more chunks will not return the required result.
For example, splitting into [4, 3], [2, 1, 0] will result in [3, 4, 0,
1, 2], which isn't sorted.
```

Example 2:

```
Input: arr = [1,0,2,3,4]
Output: 4
Explanation:
We can split into two chunks, such as [1, 0], [2, 3, 4].
However, splitting into [1, 0], [2], [3], [4] is the highest number of
chunks possible.
```

Ex :- $\overset{0}{(1}, \overset{1}{0}, \overset{2}{2}, \overset{3}{3}, \overset{4}{4})$

The key understanding is, if we sort the elements → The obtained answer will be our indexes

$\overset{0}{(1}, \overset{1}{0}, \overset{2}{2}, \overset{3}{3}, \overset{4}{4}) \xrightarrow{\text{Sort}} \overset{0}{(0}, \overset{1}{1}, \overset{2}{2}, \overset{3}{3}, \overset{4}{4})$ ← indexes

↳ values

* So, the question will be a permutation of indexes only, it won't go beyond range of indexes

So, what are chunks here ? → "partitions"

arr[5] : (1, 0, | 2, | 3, | 4)

↓

if we split the array into 4 parts and sort each part individually, the whole array will be sorted

∴ Ans is 4 chunks

for same example:

(0, 1, | 2, 3, 4)

we can split as ↑ aswell, if you sort those 2 individual chunks then also we get a sorted whole array,

but, we need to maximize chunks

arr[9] : (2, 0, 1 | 4, 3, | 6, 7, 5 | 8)

↓ sort individually

(1, 0, 2 | 3, 4, | 5, 6, 7, | 8) ✓ Array is sorted

① ② ③ ④

Ans = 4 chunks

How ?

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

* if you want to make chunks, look for maximum index of the chunks

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

All indexes and values are bound to be in this chunk only

i	arr[i]	maxValue
		$-\infty$
0	2	2
1	0	2
2	1	2 → in this case, index & maxvalue upto this point are equal, so make a chunk here
3	4	~~2~~ 4
4	3	4 → chunk again
5	6	~~4~~ 5
6	7	~~5~~ 6
7	5	~~6~~ 7 → chunk again
8	8	~~7~~ 8 → chunk again

* So, simply if you look for (maxValue & IndexPosition), you can make chunks

Note :: If already a number is placed in its perfect index, chunk++

Max chunks to make array sorted - II

768. Max Chunks To Make Sorted II

Solved

Hard | Topics | Companies | Hint

You are given an integer array `arr`.

We split `arr` into some number of **chunks** (i.e., partitions), and individually sort each chunk. After concatenating them, the result should equal the sorted array.

Return *the largest number of chunks we can make to sort the array.*

Example 1:

```
Input: arr = [5,4,3,2,1]
Output: 1
Explanation:
Splitting into two or more chunks will not return the required result.
For example, splitting into [5, 4], [3, 2, 1] will result in [4, 5, 1, 2, 3], which isn't sorted.
```

Example 2:

```
Input: arr = [2,1,3,4,4]
Output: 4
Explanation:
We can split into two chunks, such as [2, 1], [3, 4, 4].
However, splitting into [2, 1], [3], [4], [4] is the highest number of chunks possible.
```

Everything is same as type - I, problem

But, array index values are not permutation of indexes,

like → we can have any value, it can't bound ~~i~~ index max

Ex : (23, 10, 18, 27, 35, 48, 26, 52, 50, 64, 68) → arr[]

if you take this array, there is nothing to do with indexes and array values like previous problem

Intuition :

arr : (23, 10, 18, 27, 35, 48, 26, 52, 50, 64, 68)

if you consider this, max of this chunk = 23

min of this portion = 26

$\boxed{23 < 26}$

So, if you sort portion I, II seperately, they won't merge

Because,

in portion - I (max = 23)

in portion - II (min = 26)

if portion - II get sorted, individually, the first element will be 26 and that definitly lies after max element of portion - I

This is the main logic :

So, at which ever index, the max element (upti'll that index) is less than the min element (from that next index to n-1)

↓

We can make chunk

How do we keep track of which is max & particular index and which is min from back till that before index ??

we do prefixMax & suffixMin

Ex:

arr[] : (23, 10, 18, 27, 35, 48, 26, 52, 50, 64, 68)

prefixMax[] : (23, 23, 23, 27, 35, 48, 48, 52, 52, 64, 68)

Suffix Min[] : (10, 10, 18, 26, 26, 26, 26, 50, 50, 64, 68)

23 <= 26 ✓ → So we chunk

Result : prefixMax[i] <= suffixMin[i+1]

i	prefixMax[i]	suffixMin[i-1]	Result	chunk
0	23	10	False	0
1	23	18	False	0
2	23	26	True	~~0~~ 1
3	27	26	False	1
4	35	26	False	1
5	48	26	False	1
6	48	50	True	~~1~~ 2
7	52	50	False	2
8	52	64	True	~~2~~ 3
9	64	68	False	3
10	–	–	–	–

At last, no need to check, as there won't have suffixMin[i+1] for 10th index as it is last

Return ans with +1, because we don't count last chunk

Max Product Subarray

152. Maximum Product Subarray

Solved

Medium | Topics | Companies

Given an integer array `nums`, find a subarray that has the largest product, and return *the product*.

The test cases are generated so that the answer will fit in a **32-bit** integer.

Example 1:

```
Input: nums = [2,3,-2,4]
Output: 6
Explanation: [2,3] has the largest product 6.
```

Example 2:

```
Input: nums = [-2,0,-1]
Output: 0
Explanation: The result cannot be 2, because [-2,-1] is not a subarray.
```

Constraints:

- $1 <= nums.length <= 2 * 10^4$
- $-10 <= nums[i] <= 10$
- The product of any prefix or suffix of `nums` is **guaranteed** to fit in a **32-bit** integer.

Question is pretty straight forward,

* which ever subarray gives you max product return it

Observations :

what if ,

① all are +ve's
↳ product of whole array is our ans

② Even negatives , rest all are positives
↳ product of whole array is our ans

③ Odd negatives , rest all are positives

arr : (2, 3, -2, 4)

↓

The possible answer will be of left side of '-2' or right side of '-2'

We can't include -2 in our answer

So, answer can be from (2, 3) or (4)

↓ ↓

$2 * 3 = 6$ 4

↓

max = 6

if more negatives

Ex: arr[] : (2, 3, 4, -5, 6, -2, 3, -1, 4, 3)

* if we try to remove one negative ⇒ we can make the count of negatives as even

then, there will be no problem will come in product

So, from which side you will remove that negative ???

1. from front
2. from back
3. from middle

1. from front :

arr[] : (2, 3, 4, ~~-5~~, 6, -2, 3, -1, 4, 3)

possible arrays : (2, 3, 4) (6, -2, 3, -1, 4, 3)

↑ max product

2. from back :

arr[] : (2, 3, 4, -5, 6, -2, 3, ~~-1~~, 4, 3)

possible arrays : (2, 3, 4, -5, 6, -2, 3) (4, 3)

↑ max product

3. from middle :

arr[] : (2, 3, 4, -5, 6, ~~-2~~, 3, -1, 4, 3)

possible arrays : (2, 3, 4, -5, 6) (3, -1, 4, 3)

↑ negative ans ↑ negative ans → so, no use

arr[] : (1, 2, -1, -3, 4, -2, 6, 2, -3, -5, -3, 2, 4, 6)

from middle

arr[] : (1, 2, -1, -3, 4, ~~-2~~, 6, 2, -3, -5, -3, 2, 4, 6)

possible arrays : (1, 2, -1, -3, 4) , (6, 2, -3, -5, -3, 2, 4, 6)

+ve ans — Ok

-ve ans — Not Ok

So, don't prefer going for removing negative from middle

go with prefix or suffix

Observation 4

if array contains zero

↳ Split the array, don't mix 0 with answer, move On

Ex:- arr[] : (2, 3, -2, 4)

i	prefixproduct[i]	Suffixproduct[n-i-1] (from last)	max
0	2	4	~~-∞~~ 4
1	$\boxed{2 * 3 = 6}$	$4 * -2 = -8$	~~4~~ 6
2	$6 * -2 = -12$	$-8 * 3 = -24$	6 (dont update)
3	$-12 * 4 = -48$	$-24 * 2 = -48$	6

```
public int maxProduct(int[] nums) {
    double pp = 1,sp = 1,max = Integer.MIN_VALUE;;
    int n = nums.length;
    for(int i = 0;i<n;i++){
        if(pp == 0)pp = 1;
        if(sp == 0)sp = 1;

        pp = pp*nums[i];
        sp = sp*nums[n-i-1];

        max = Math.max(max,Math.max(pp,sp));
    }
    return (int)max;
}
```

if any of them becomes zero, reset

Came from prefix

So, in this case we are neglecting suffix

Two Sum

167. Two Sum II - Input Array Is Sorted Solved

Medium | Topics | Companies

Given a **1-indexed** array of integers `numbers` that is already ***sorted in non-decreasing order***, find two numbers such that they add up to a specific `target` number. Let these two numbers be `numbers[index1]` and `numbers[index2]` where `1 <= index1 < index2 <= numbers.length`.

Return *the indices of the two numbers,* `index1` *and* `index2`*,* ***added by one*** *as an integer array* `[index1, index2]` *of length 2.*

The tests are generated such that there is **exactly one solution**. You **may not** use the same element twice.

Your solution must use only constant extra space.

Example 1:

```
Input: numbers = [2,7,11,15], target = 9
Output: [1,2]
Explanation: The sum of 2 and 7 is 9. Therefore, index1 = 1, index2 =
2. We return [1, 2].
```

Example 2:

```
Input: numbers = [2,3,4], target = 6
Output: [1,3]
Explanation: The sum of 2 and 4 is 6. Therefore index1 = 1, index2 = 3.
We return [1, 3].
```

Example 3:

```
Input: numbers = [-1,0], target = -1
Output: [1,2]
```

return any 2 indices, which results the sum of both indices equals to the given target

2 pointer's technique

If you are required to find elements that are satisfying certain constraints or conditions → you use 2 pointers technique

Most commonly used problems:

1. Two Sum
2. Three Sum, four sum, closest three sum....
3. Container with most water
4. Sort n Searching
5. String manipulations
6. Palindrome problems
7. linkedlist problems
8. Sliding window problems

So, as our array is <u>sorted</u>, we can make observations out of it

arr: (2, 7, 11, 15)

If you set pointer → i at 0, j at n-1 ⇒ (2, 7, 11, 15)

i j

* if you sum up arr[i] & arr[j] → you will get some 'x' ans right?

let us assume, (Sum we got == X)

Case - I

If (sum < target)

it means our sum need to be increased to match target value

for that, you have to increment i, as your array is sorted, if you move i forward, the value definetly increase

Case - II

If (sum > target)

it means our sum need to be decreased to match target value

for that, you have to decrement j, as your array is sorted, if you move j backward, the value definetly decreases

Case - III

if (sum == target)

↳ return the index values of (i, j)

	i	j	sum	target	operation
tar = 18 { 2, 7, 11, 15 } (0, 1, 2, 3)	0	3	17	< 18	i++
tar = 18 { 2, 7, 11, 15 } (0, 1, 2, 3)	1	3	21	> 18	j--
tar = 18 { 2, 7, 11, 15 } (0, 1, 2, 3)	1	2	18	== 18	✓ (return {1, 2})

Three Sum

15. 3Sum Solved

Medium | Topics | Companies | Hint

Given an integer array nums, return all the triplets `[nums[i], nums[j], nums[k]]` such that `i != j`, `i != k`, and `j != k`, and `nums[i] + nums[j] + nums[k] == 0`.

Notice that the solution set must not contain duplicate triplets.

Example 1:

```
Input: nums = [-1,0,1,2,-1,-4]
Output: [[-1,-1,2],[-1,0,1]]
Explanation:
nums[0] + nums[1] + nums[2] = (-1) + 0 + 1 = 0.
nums[1] + nums[2] + nums[4] = 0 + 1 + (-1) = 0.
nums[0] + nums[3] + nums[4] = (-1) + 2 + (-1) = 0.
The distinct triplets are [-1,0,1] and [-1,-1,2].
Notice that the order of the output and the order of the triplets does
not matter.
```

Example 2:

```
Input: nums = [0,1,1]
Output: []
Explanation: The only possible triplet does not sum up to 0.
```

Example 3:

```
Input: nums = [0,0,0]
Output: [[0,0,0]]
Explanation: The only possible triplet sums up to 0.
```

* if we add 3 elements index values, that will be equal to 0, then
  ↓
  if this happens, store such index values triplet and return all triplets

* No duplicates allowed

Suppose : (-1, 0, 1, 2, 2, -1, 4)

ans1 = (-1, 0, 1, 2, 2, -1, 4) (-1, 0, 1, 2, 2, -1, 4) = ans2

Both answers results in zero, but both triplets are same, so don't allow triplets

Simply, for that to happen...

1. Sort the elements
2. Skip duplicates

* And, if we have a sorted array, and we are looking for a certain condition on index values
  ↓ we can use two pointers

nums[i] + nums[j] + nums[k] == 0

nums[] : (-1, 0, 1, 2, -1, 4)

↓ sort

(-1, -1, 0, 1, 2, 4)

placing pointers :

(-1, -1, 0, 1, 2, 4)

↑ i ↑ j ↑ k

* We start with i → traverse from 0 to n-2

* for every i → we have (j, k)
  ↳ j starts from i+1
  k starts from n-1

Now, apply 2 Sum ideology, b/w (j, k) pointers

* for every (j, k) look up for nums[i] + nums[j] + nums[k] == 0 ?

if (sum < 0) → j++ {Sum need to increase, so increment j, as array sorted value will increase

if (sum > 0) → k-- {Sum need to decrease}

if (sum == 0) → Add triplet & move j, k

* when (j, k) crosses, move (i)

arr : (-1, -1, 0, 1, 2, 4)
0 1 2 3 4 5

↑ i ↑ j ↑ k

i	j	k	Sum	operation
0	1	5	2 > 0	k--
0	1	4	0 == 0	Add triplet
// Skip all	k = 2's	&	j = -1's	to ignore duplicates
0	2	3	0 == 0	Add triplet

arr : (-1, -1, 0, 1, 2, 4)
↑ i ↑ j ↑ k

arr : (-1, -1, 0, 1, 2, 4)
↑ i ↑ j ↑ k

arr : (-1, -1, 0, 1, 2, 4)
↑ i ↑ j ↑ k

Because, with i=0, you can't form any other triplets with same values of (j, k)

if (j & k Crosses)
↓
increment i → & repeat same

* Above steps are for triplets which includes $i=0$

* Do same steps for $i=1, 2, \ldots n-2$

* Then our ans will consists of every possible triplet

T.C: $O(n\log n)$ → sorting

$O(n^2)$ to traverse (i, j, k) $\approx O(n^2)$

Container with Most Water

11. Container With Most Water

Solved

Medium | Topics | Companies | Hint

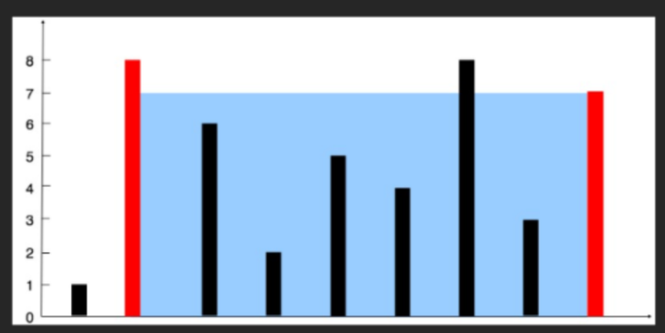
You are given an integer array `height` of length `n`. There are `n` vertical lines drawn such that the two endpoints of the ith line are `(i, 0)` and `(i, height[i])`.

Find two lines that together with the x-axis form a container, such that the container contains the most water.

Return *the maximum amount of water a container can store.*

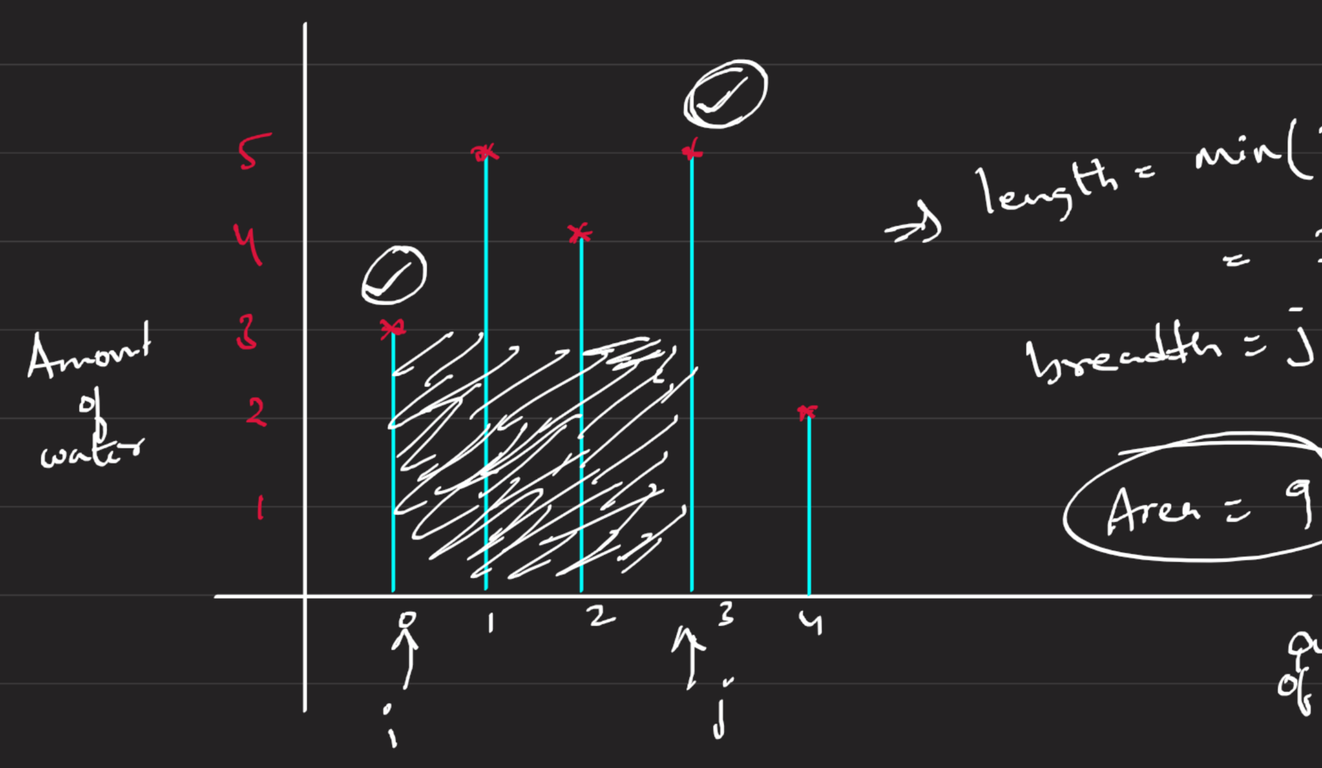
Notice that you may not slant the container.

Example 1:



Input: height = [1,8,6,2,5,4,8,3,7]
Output: 49
Explanation: The above vertical lines are represented by array [1,8,6,2,5,4,8,3,7]. In this case, the max area of water (blue section) the container can contain is 49.

use a pointer technique



→ length = min(3,5)
= 3

breadth = j - i = 3 - 0
= 3

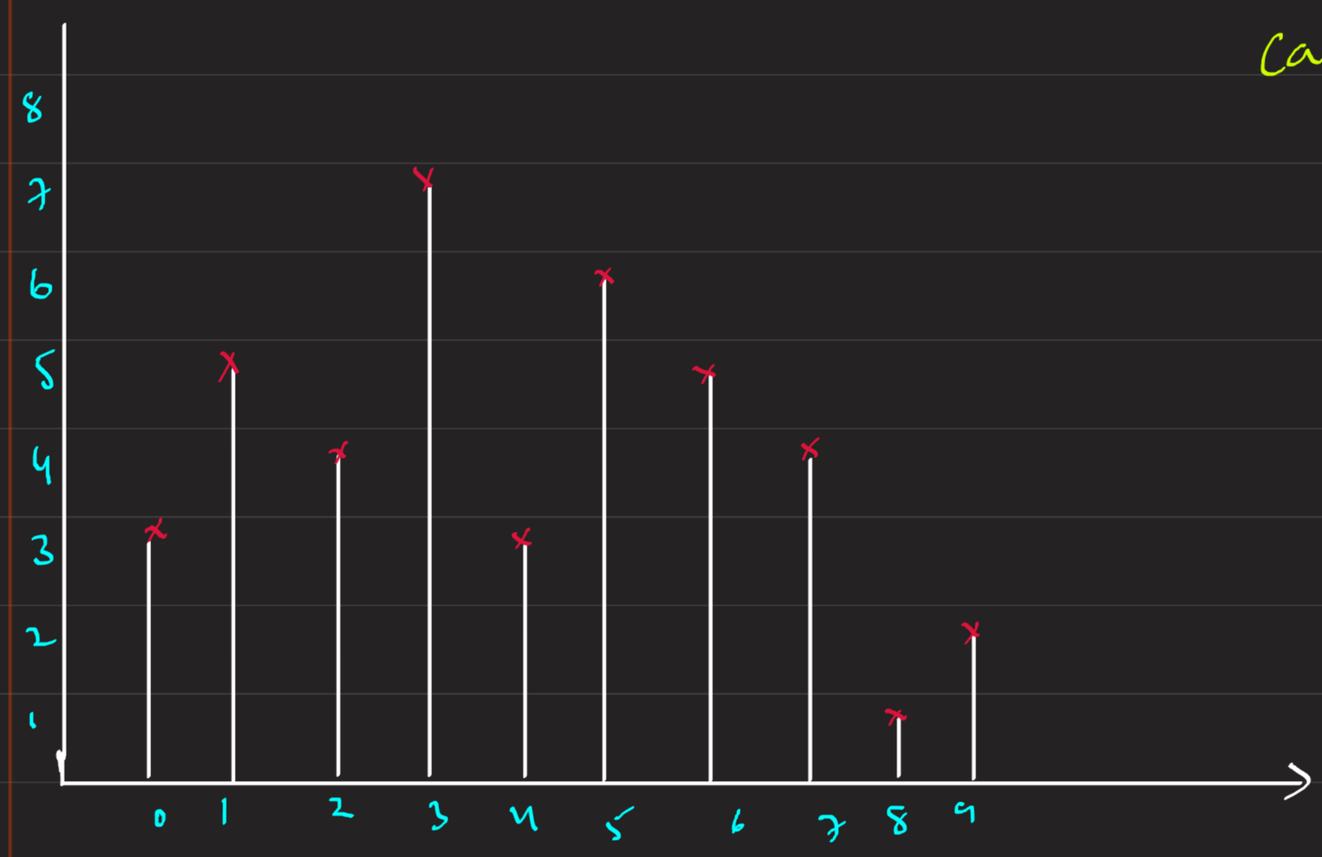
Area = 9

amount of water stored

Amount of water = length × Breadth → nothing but area

$$= (j - i) \times (\min(arr[i], arr[j]))$$

arr[] : (3, 5, 4, 7, 3, 6, 5, 4, 1, 2)
(0 1 2 3 4 5 6 7 8 9)



Case - I

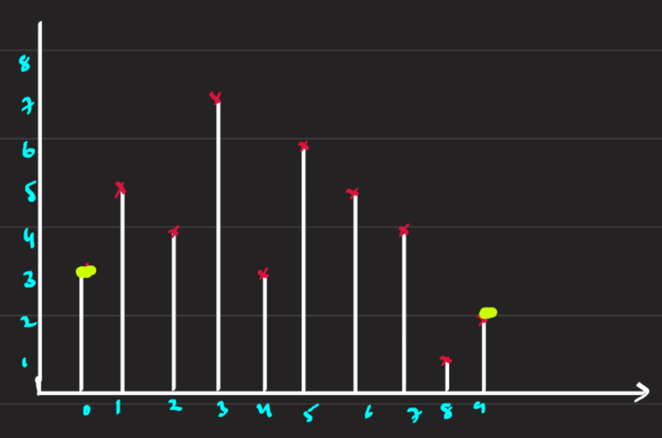
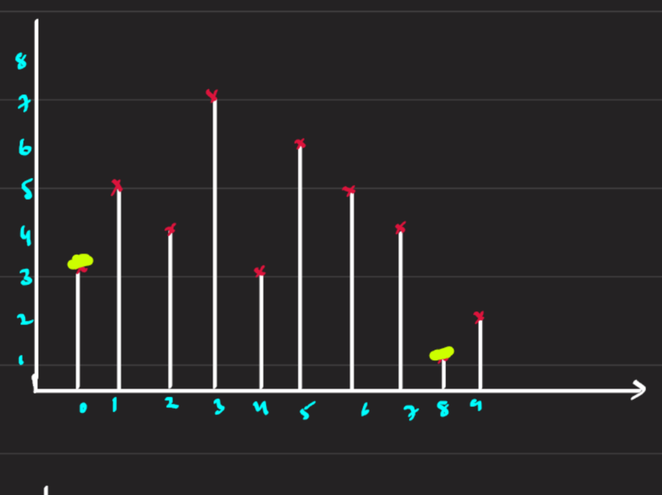
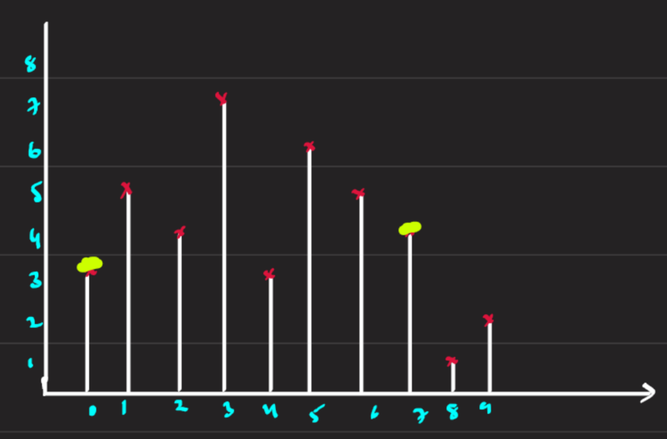
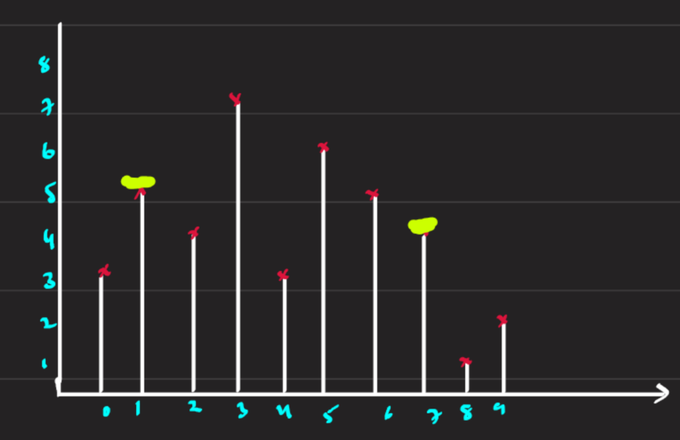
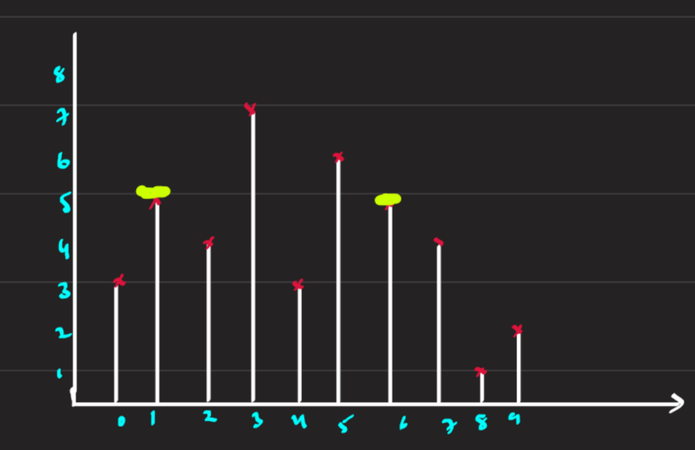
if arr[i] > arr[j]
↓
j is not useful in making max area

so → j--

Case - II

if arr[i] < arr[j]
↓
i is not useful in making max area

so → i++

	a(i)		a(j)	height	width	amount	max ans	operation
	3	>	2	2	9	18	~~0~~ 18	j--
	3	>	1	1	8	8	18	j--
	3	<	4	3	7	21	~~18~~ 21	i++
	5	>	4	4	6	24	~~21~~ 24	j-
	5		5	5	5	25	~~24~~ 25	j-

likewise update (i, j) and store max potential area of ans

* After (i, j) crosses which area is lastly updated, that will be our answer

most water contained

T.C : O(n) , S.C : O(1)

Sort an array of 0's, 1's, 2's

↳ Dutch National Flag Algorithm

75. Sort Colors

Solved

Medium | Topics | Companies | Hint

Given an array `nums` with `n` objects colored red, white, or blue, sort them **in-place** so that objects of the same color are adjacent, with the colors in the order red, white, and blue.

We will use the integers `0`, `1`, and `2` to represent the color red, white, and blue, respectively.

You must solve this problem without using the library's sort function.

Example 1:

```
Input: nums = [2,0,2,1,1,0]
Output: [0,0,1,1,2,2]
```

Example 2:

```
Input: nums = [2,0,1]
Output: [0,1,2]
```

Constraints:

- `n == nums.length`
- `1 <= n <= 300`
- `nums[i]` is either `0`, `1`, or `2`.

Follow up: Could you come up with a one-pass algorithm using only constant extra space?

Algorithm

① This algorithm contains 3 pointers, i.e... (low, mid, high)

② And 3 main rules,

* arr[0 ... low − 1] → Contains 0
* arr[low .... mid − 1] → Contains 1
* arr[high+1 .... n−1] → Contains 2

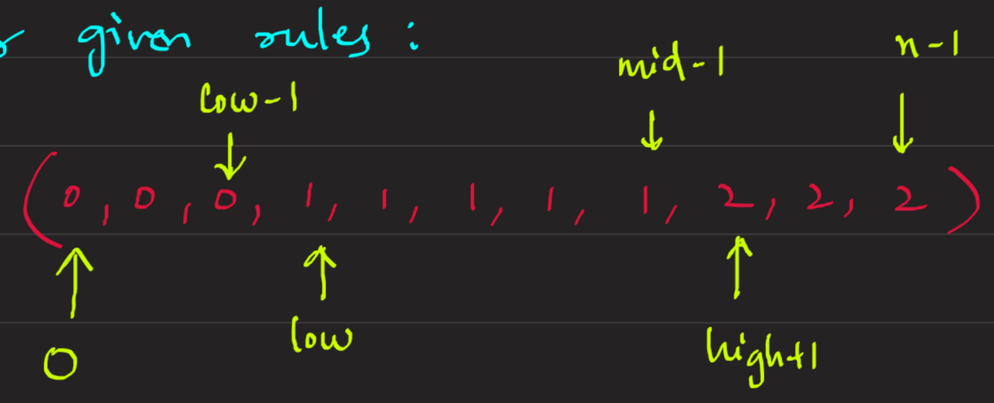
* Whatever array you are given, make them into above structure

input array : (1, 0, 2, 0, 0, 1, 2, 2, 1, 1, 1)

↓

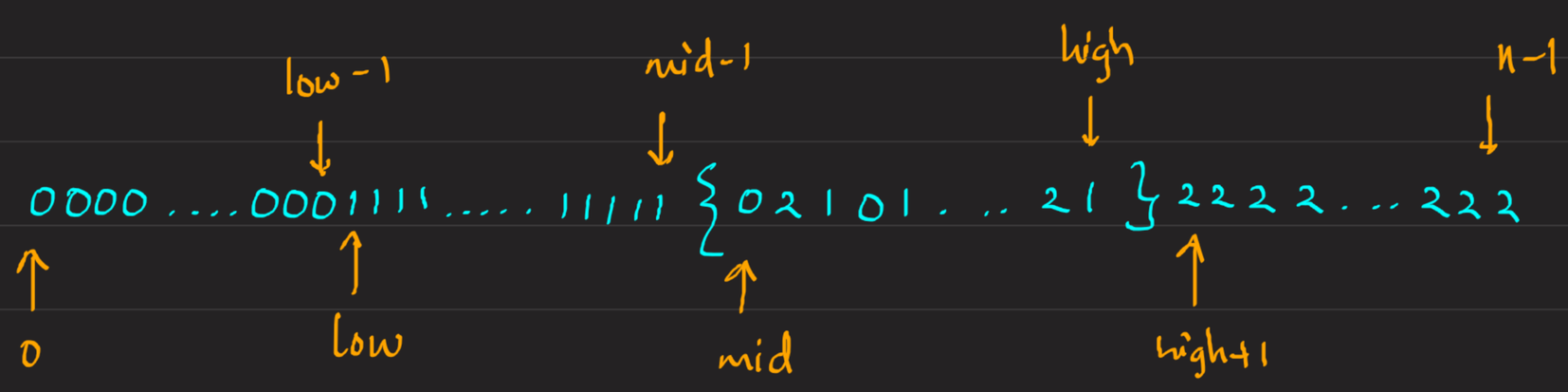
final array : (0, 0, 0, 1, 1, 1, 1, 1, 2, 2, 2)

place pointers as per given rules :



* If you observe carefully, we don't find (mid to high) portion, that is because, this is already a sorted array

Generalizing



We have to clear that (mid to high portion) → then our array will be sorted

// The unsorted portion lies b/w (mid, high)

* We have to arange these mid to high values, so, that our top 3 rules won't break

* In our case, we assume whole array is unsorted, so we place pointers accordingly

mid

arr : (2 0 2 1 1 0)

low high

After all the values move to the original places,

(high, mid) will cross

↳ so erases

Observations

① Case - I

if (arr[mid] == 0)

↳ ① we will swap(arr[low], arr[mid]

② increment both mid & low

② Case - II

if (arr[mid] == 1)

↳ we will just increment mid

③ Case - III

if (arr[mid] == 2)

↳ ① swap(arr[mid], arr[high])

② decrement high

Dry Run

arr[]: (0 1 1 0 1 2 1 2 0 0 0)
mid → 0, low → 0, high → 0

① arr[mid] == 0 (case - I)

arr[]: (0 1 1 0 1 2 1 2 0 0 0)
mid → 1, low → 1, high → 0

② arr[mid] == 1 (case - II)

arr[]: (0 1 1 0 1 2 1 2 0 0 0)
mid → 1, low → 1, high → 0

③ arr[mid] == 1 (case - II)

arr[]: (0 1 1 0 1 2 1 2 0 0 0)
mid → 0, low → 1, high → 0

④ arr[mid] == 0 (case - I)

arr[]: (0 ~~1~~ 1 ~~0~~ 1 2 1 2 0 0 0)
0, 1, mid, low, high

arr[]: (0 ~~1~~ 1 ~~0~~ 1 2 1 2 0 0 0)
0, 1, mid, low, high

⑤ arr[mid] == 1 (case - II)

arr[]: (0 0 1 1 1 2 1 2 0 0 0)
mid → 2, low → 1, high → 0

⑥ arr[mid] == 2 → (case - III)

arr[]: (0 0 1 1 1 ~~2~~ 1 2 0 0 ~~0~~)
0, mid, 2, low, high

arr[]: (0 0 1 1 1 ~~2~~ 1 2 0 0 ~~0~~)
0, mid, 2, low, high

⑦ arr[mid] == 0 → case (I)

arr[]: (0 0 ~~1~~ 1 1 ~~0~~ 1 2 0 0 2)
0, mid, 1, low, high

arr[]: (0 0 ~~1~~ 1 1 ~~0~~ 1 2 0 0 2)
0, mid, 1, low, high

likewise --> it goes

when mid & high crosses our array goes sorted

T.C : O(n)
S.C : O(1)

Merge Overlapping Intervals

56. Merge Intervals

Solved

Medium | Topics | Companies

Given an array of `intervals` where `intervals[i] = [start_i, end_i]`, merge all overlapping intervals, and return *an array of the non-overlapping intervals that cover all the intervals in the input.*

Example 1:

```
Input: intervals = [[1,3],[2,6],[8,10],[15,18]]
Output: [[1,6],[8,10],[15,18]]
Explanation: Since intervals [1,3] and [2,6] overlap, merge them into [1,6].
```

Example 2:

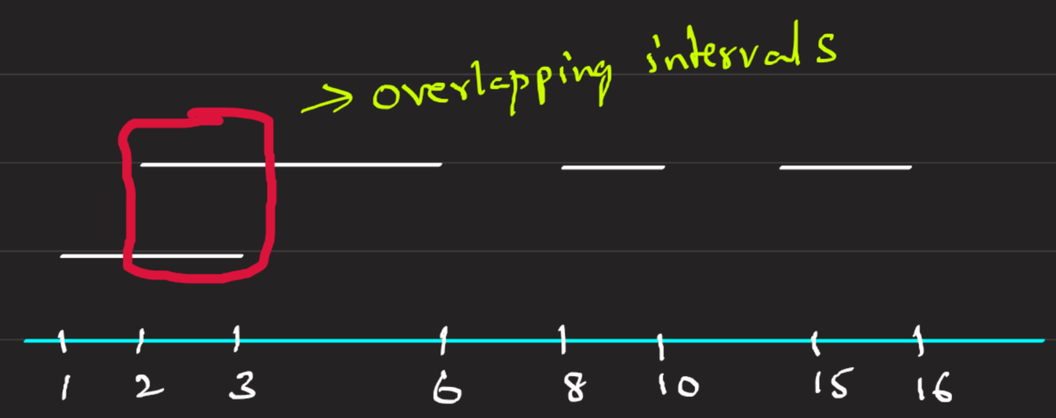
```
Input: intervals = [[1,4],[4,5]]
Output: [[1,5]]
Explanation: Intervals [1,4] and [4,5] are considered overlapping.
```

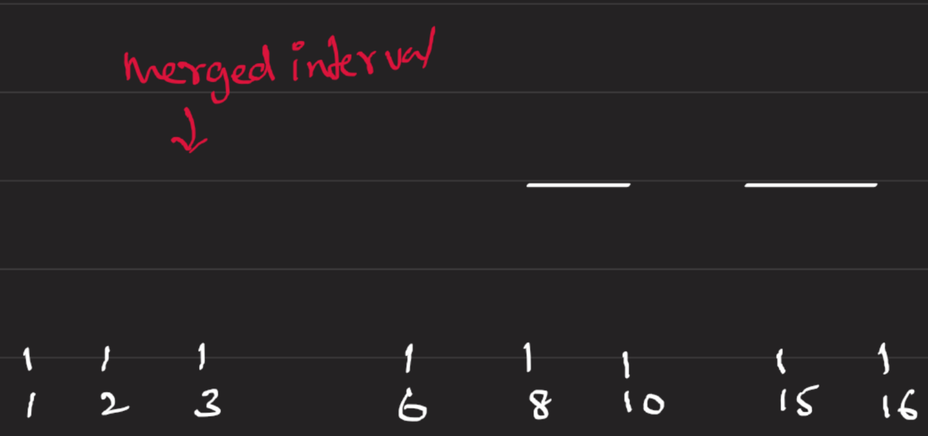
Constraints:

- `1 <= intervals.length <= 10^4`
- `intervals[i].length == 2`
- `0 <= start_i <= end_i <= 10^4`

First sort the array intervals

Ex:- [[1,3], [2,6], [8,10], [15,18]]





∴ Ans ⇒ [[1,6], [8,10], [15,16]]

* We have to make sure that, in final ans, there won't be any overlapping intervals

Approach

* Iterate from 0 → n
  ↳ as initially, our ans list is empty, we add first element into it

Observations

if (currentInterval[i][0] < lastInsertedArray[1])

lastInsertedArray[1] = max(currentInterval[i][1], lastInsertedArray[1])

Case-I

If the current interval can be merged with the last inserted interval of the answer list

comparator 1

ans list

Ex: ([0, 2], [1, 4], [8, 10])

[0, 2] → lastInsertedInterval Array

1 is in range of the list,

So update interval end as max(4, 2)

Updation

maximum (currentIntervalsEnd, lastInsertedIntervalEnd)

Case-II

If the current interval cannot be merged with last inserted interval of answer list

↓

we insert the current interval in the answer list and update last inserted interval as current interval

if (currentInterval[i][0] > lastInsertedInterval[1]) → ignore

For your Benefit

To Sort 2D arrays

Any 2 random variables

this says based on 0th index sorting

① Arrays.sort(arr, (a, b) → Integer.compare(a[0], b[0]))

array name

if you put a first → increasing order

Now, if i want a 2D array to sort descending order

Arrays.sort(arr, (a, b) → Integer.compare(b[0], a[0]))

Dry Run

i	currentIntervalStart	lastIntervalEnd	Check Merge	Current Interval End	update lastIntervalEnd
1	[2,6] → 2	[1,3] → 3	2 < 3 ✓	6	[1,6]
2	[8,9] → 8	[1,6] → 6	8 > 6 ✗	✗	[1,9]
3	[10,13] → 10	[8,9] → 9	10 > 9 ✗	✗	[10,13]
4	[11,12] → 11	[10,13] → 13	11 < 13 ✓	12	[10,13]
5	[14,15] → 14	[10,12] → 13	14 > 13 ✗	✗	[14,15]

Set Matrix to zeroes

73. Set Matrix Zeroes

Solved

Medium | Topics | Companies | Hint

Given an `m x n` integer matrix `matrix`, if an element is `0`, set its entire row and column to `0`'s.

You must do it in place.

Example 1:

1	1	1	⇨	1	0	1
1	0	1		0	0	0
1	1	1		1	0	1

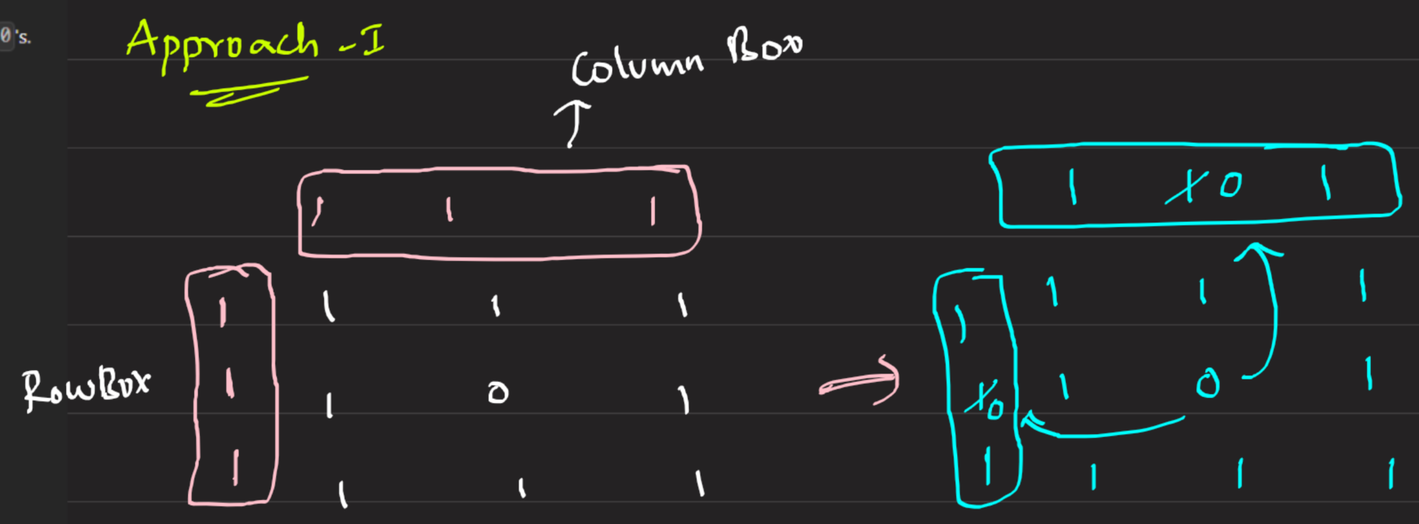
```
Input: matrix = [[1,1,1],[1,0,1],[1,1,1]]
Output: [[1,0,1],[0,0,0],[1,0,1]]
```

Example 2:

0	1	2	0	⇨	0	0	0	0
3	4	5	2		0	4	5	0
1	3	1	5		0	3	1	0

```
Input: matrix = [[0,1,2,0],[3,4,5,2],[1,3,1,5]]
Output: [[0,0,0,0],[0,4,5,0],[0,3,1,0]]
```

Approach - I



* Create a rowBox & colBox, set to '1' initially

Step 1: Traverse through the array and make rowBox value & colBox value of particular index to zero, if you found matrix value as 0

Step 2: Then re-iterate on matrix & check rowBox & colBox, if any of them is zero, make the matrix value to 0

```
int rowBox[] = new int[ m ]
int colBox[] = new int[ n ]

for(int i -> 0 to m)
    for(int j -> 0 to n)
        if (matrix[i][j] == 0) {
            rowBox[i] = 0
            colBox[j] = 0
```

then

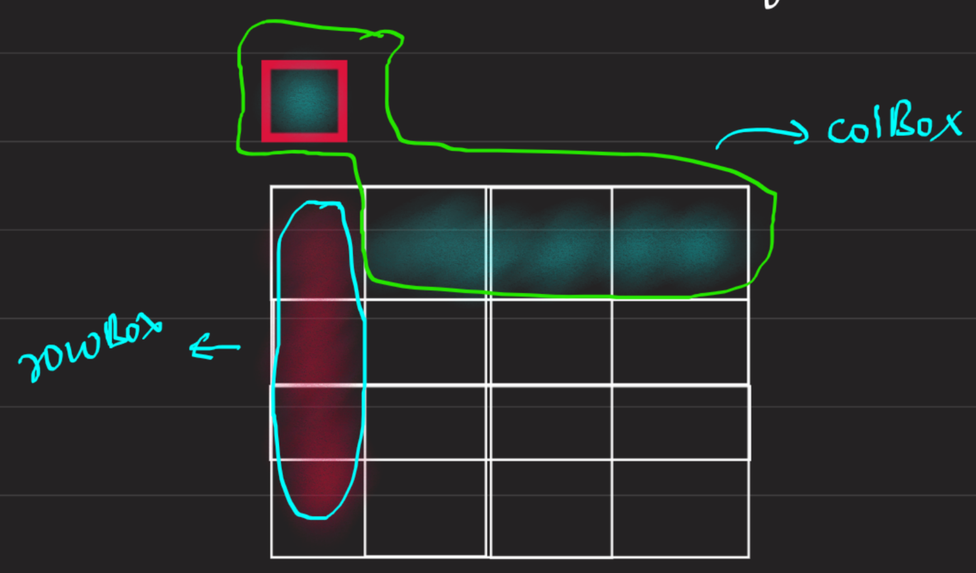
```
for (int i = 0 -> m){
    for(int j = 0 -> n)
        if (rowBox[i] == 0 || colBox[j] == 0)
            matrix[i][j] = 0
```

T.C : $O(N^2)$, SC : $O(N) + O(M)$

😮‍💨 we have to reduce Space Complexity ....

Approach

Instead of using seperated arrays for rowBox & colBox

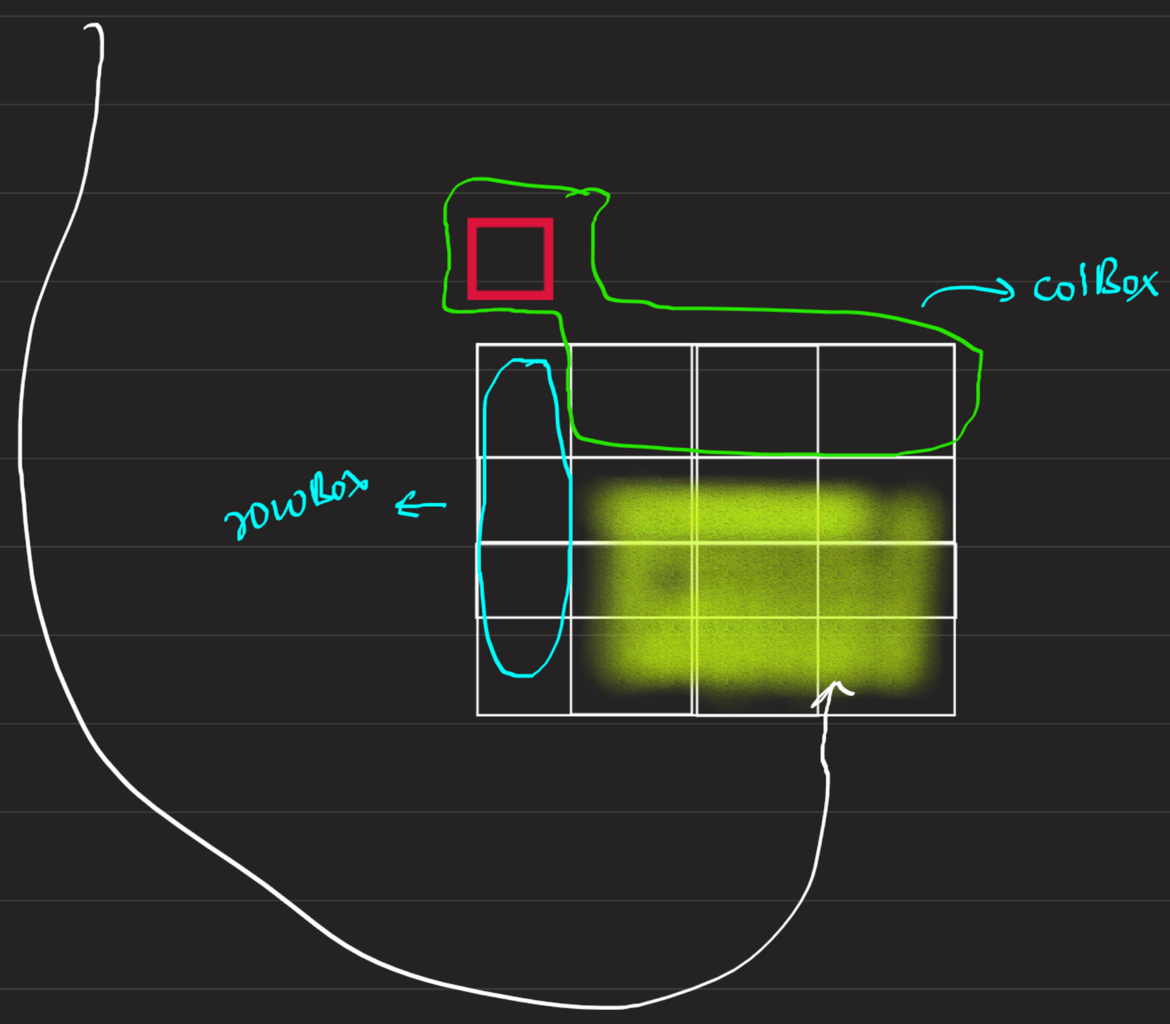


rowBox ⇒ col[0]

colBox ⇒ col[0], row[0] → from 1st index

↑ variable

* Traverse for (1,1) to (m x n) without include first row, first Column
* Make rowBox, colBox as zeroes
* Iterate from (1,1) to (m x n) again and make them zeroes



* After that, modify columnBox ⟶ mean's 0th row

Because, 0th row is dependent on 0th col if you modify 0th col first, then 0th row ans will change.

* Then modify ⇒ 0th column

T.C : $O(n^2)$

S.C : $O(1)$