LeetCode

STRIVER'S SDE SHEET

RAISONI GROUP
— a vision beyond —

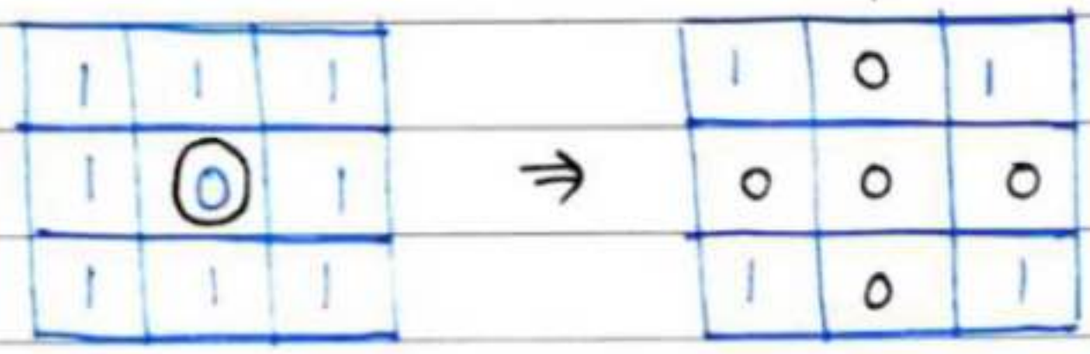
Day - 1

① Set Matrix Zeroes :- [Medium]

Given an 'mxn' integer matrix, if an element is '0' set the entire row and column to ⇒ 0.

You must do it in place :

Input :-



Brute-force approach :-

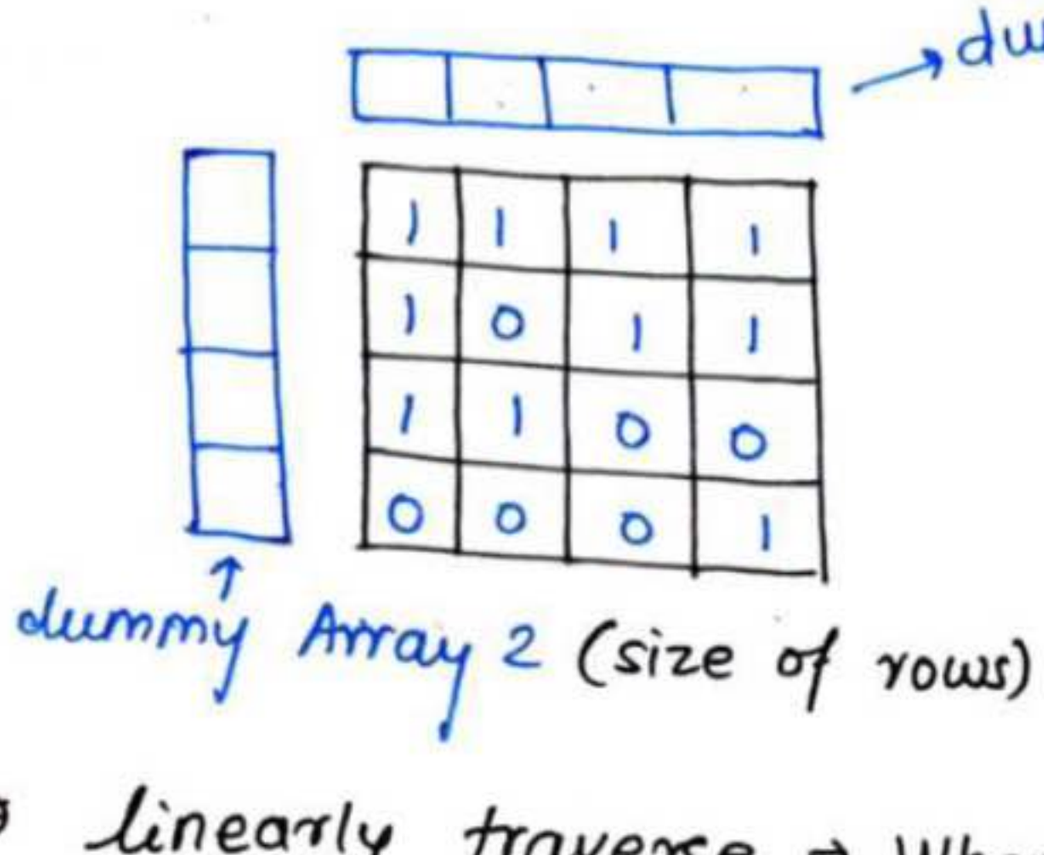
Whenever we find "0" → then go through its row and make all elements ⇒ -1. and same, go through column and make all elements ⇒ -1

After checking the whole 2D array, then change the -1 element ⇒ to "0" and here we have got our answer.

Complexity :- $(N \times m) \times (N + M)$

Space complexity ⇒ O(1)

Optimized approach



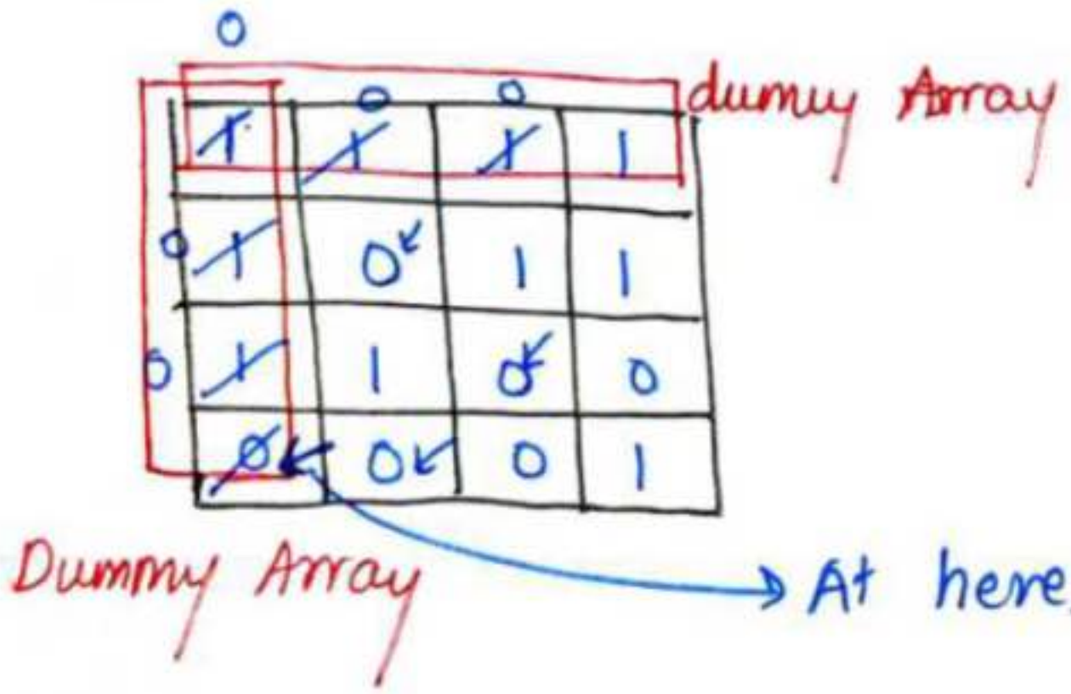
We'll take two dummy arrays

⇒ linearly traverse ⇒ Whenever you find ⇒ 0 then make
0 ⇒ in the columnth index &
0 ⇒ in the rowth index

Complexity ⇒ $O(N \times m + N \times m)$

Space Complexity ⇒ $O(N) + O(M)$

Most Optimized Approach :-



Take new variable
bool col = true

Now traverse from back & check whether for that particular element, zero is present in the dummy column array or dummy row array if yes then convert that element to Zero (0).

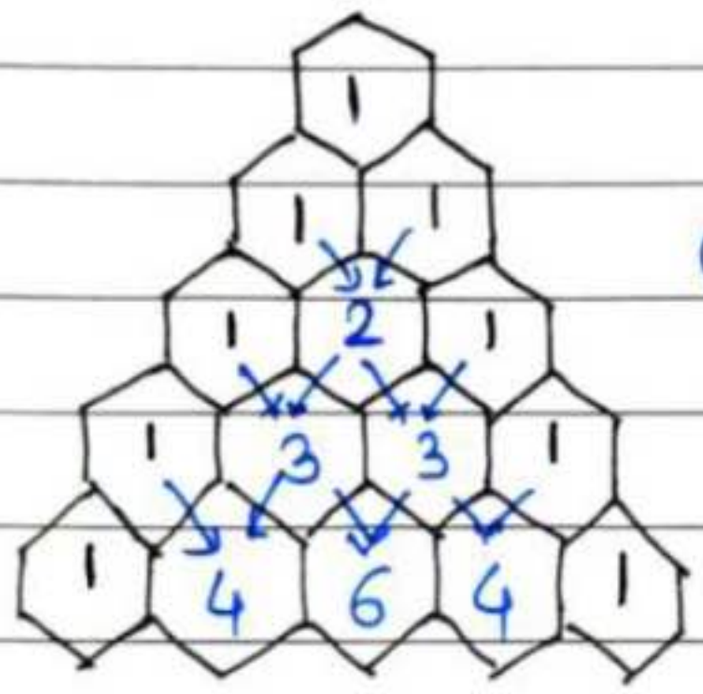
Time complexity = $2*(N \times m)$

Space Complexity ⇒ $O(1)$

RAISONI GROUP
— a vision beyond —

(2) Pascals Triangle :-

Given an integer "numRows", return the first numRows of pascal triangle.



numRows = 5

Output = [[1],
[1,1],
[1,2,1],
[1,3,3,1],
[1,4,6,4,1]]

Approach :-

Another Subproblem

print only one of the row from pascals triangle

let's suppose 5th row should be printed then

```
for(i=0; i<k ; ++i){
    res *= (n-i);
    res /= (n+i);
}
```

If in interview, they ask what is the value present at R^{th} row & C^{th} column i.e 5th row & 3rd column ⇒ (6)

then formula $(R-1)C_{(C-1)}$

Now for the original problem.

we can resize the vector to (i+1), then for(i=0 ; i<numRaw; ++i)

ans[i][0] = ans[i][i] = 1

↓ first column element

↓ last column element

then

for(j=1 ; j<i ; ++j) ⇐

ans[i][j] = ans[i-1][j-1] + ans[i-1][j];

return ans.

③ Next Permutation :-

Array = [1,2,3] then its permutations are

[1,2,3]

[1,3,2]

[3,1,2]

[2,3,1]

then next permutation for [1,2,3] ⇒ **[1,3,2]**

Brute-Apporach

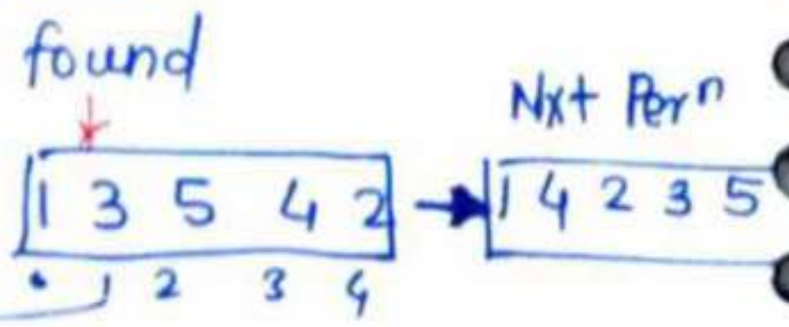
Generate All possible combos (permutations)
then find the nums array
return its next permutation

and
if last nums is found then the first permutation will be its next permutation

Optimal Approach:-

linearly traverse from backwards
and we have to traverse till we found

arr[i] < arr[i+1]



index 1 = 1

~~Again traverse~~

II^nd step linear traversal from back and find element which is actually greater than the value at index = 1
then we have got 4

index 2 = 3

~~Swap (index 1~~

III^rd step :- Swap(arr[index 1], arr[index 2])
↳ 1 4 5 3 2

IV^th step :- reverse(index 1+1, last);

Time complexity ⇒ O(n)

RAISONI GROUP
— a vision beyond —

(5) Maximum Subarray [Kadane's Algorithm] :-

Given an integer array nums, find the contiguous subarray which has the largest sum and returns its sum.

Example :-

Input :- nums = [-2, 1, -3, 4, -1, 2, 1, -5, 4]

Output = 6

max. sum = 6

Solution :-

```
int mx = INT_MIN ;
```

~~Take one~~ Traverse through array compute sum and take mx = max(sum, mx)
then whenever sum will be <0
then set sum = 0
return mx.

Time complexity = O(n)

6. Sort an array of 0's, 1's & 2's :-

Sort colors [leetcode]

Given an array of 'nums' with n colored objects as 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 & blue

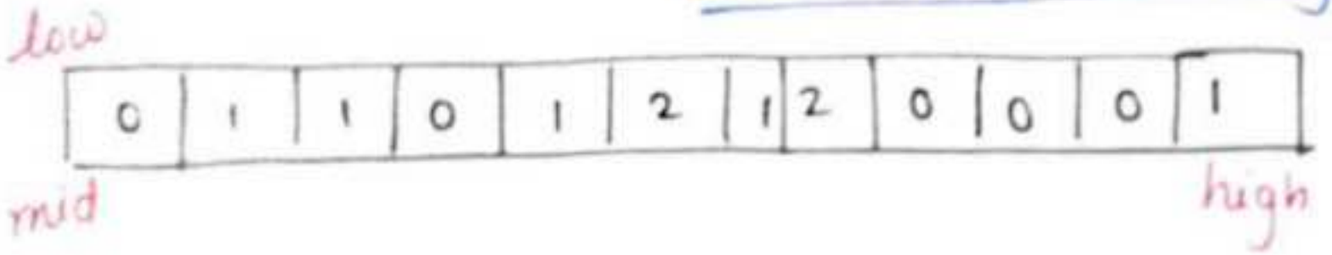
Ex :-

Input :- nums = [2,0,2,1,1,0]

output = [0,0,1,1,2,2]

Approach :- This problem can be solved by using

Dutch National Flag Algorithm



All the no from

[0 .. low-1] ⇒ are "0".

[high+1 .. n] ⇒ are "2".

Now shift mid pointer

if mid pointer points ⇒ 0 → then [swap → (arr[low], arr[mid])
low++;
mid++;

if mid pointer points ⇒ 1 → then mid++;

if mid pointer points ⇒ 2 → then { swap(arr[mid], arr[high])
high--;

RAISONI GROUP
— a vision beyond —

Day2

⑦ Rotate Image [Medium]

You are given an ~~n*n~~ nxn 2D matrix representing an Image rotate the image by 90 degree clockwise.

You have to rotate the image in-place, which means you have to modify the input 2D matrix directly.
* Do not allocate another 2D matrix and do the rotation

Example :-

Input ⇒

1	2	3
4	5	6
7	8	9

⇒

7	4	1
8	5	2
9	6	3

Approach :-

Take transpose of the matrix

A =

1	2	3
4	5	6
7	8	9

⇒ A^T =

1	4	7
2	5	8
3	6	9

then

reverse each row of the transpose matrix

Resultant matrix =

7	4	1
8	5	2
9	6	3

⇐ ANSWER

⑧ Merge Intervals [Medium]

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 :-

Intervals = [[1,3], [2,6], [8,10], [15,18]]

Output = [[1,6], [8,10], [15,18]]

Approach :-

① **Brute-force** sort all the given intervals.
then check if the interval is merging or not
if yes ⇒ then add to the DS. **T.C = O(nlogn)**

② **Optimized Approach**

	1,3	2,6	8,10	8,9	9,11	15,18	2,4	16,17
Sort	1,3	2,4	2,6	8,9	8,10	9,11	15,18	16,17

linearly iterate. and check [1,3 / 1,3] are merging or not if yes ~~then add~~ in ~~DS.~~

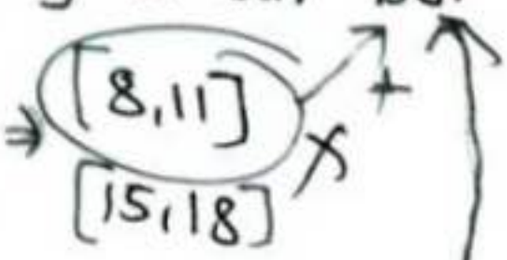
move pointer on next index, Now [1,3 / 2,4] then ⇒ [1,4]

Now [1,4 / 2,6] ⇒ [1,6]

then [1,6] & [8,9] they are not merging ⇒ At this point add [1,6] to our DS.

Now [8,9 / 8,10] ⇒ [8,10]

Again [8,10 / 9,11] ⇒ [8,11]



Now [15,18 / 16,17] ⇒ [15,17]

Add to DS

Answer = [1,6], [8,11], [15,17]

TC = O(NlogN) + O(N)
↑ sorting
→ for traversing

RAISONI GROUP
— a vision beyond —

(9) Merge- sorted Array in O(1) space :-

You are given two integer arrays 'num1' & 'num2' sorted in non-decreasing order, and two integers m & n. representing the number of elements in num1 & num2 respectively.

Merge num1 & num2 into a single array sorted in non-decreasing order.

Example :-

$a_1[] =$ | 1 | 4 | 7 | 8 | 10 | size = m

$a_2[] =$ | 2 | 3 | 9 | size = n

Approach :-

Take one more array a_3 of size (m+n) and then add a_1 elements & a_2 elements in the a_3 array.

⇓

Sort it

⇓

Take out ~~ete~~ m elements from a_3 put in a_1 then take n elements from a_3 put in a_1

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

$$S.C = O(n)$$

Approach :- 2

You can't take extra Space

arr1 = [1] 4 7 8 10

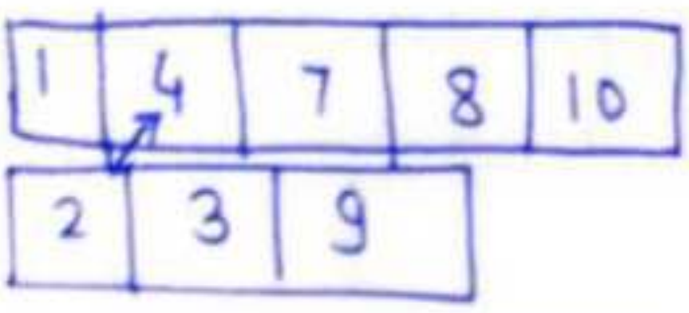
arr 2 = 2 3 9

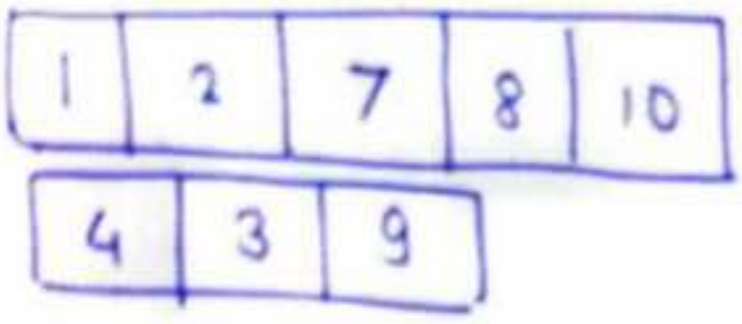
Traverse linearly in arr1 & check if arr1[i] > arr2[i]

⇓

if yes then

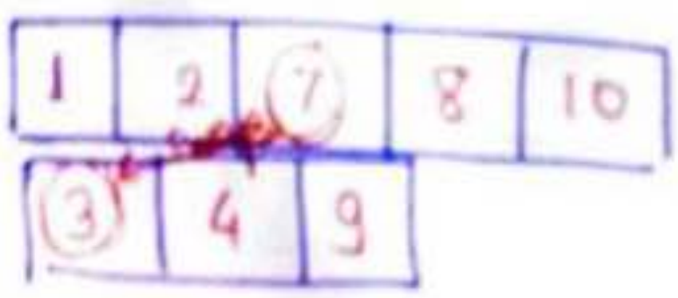
swap(arr1[i], arr2[i])



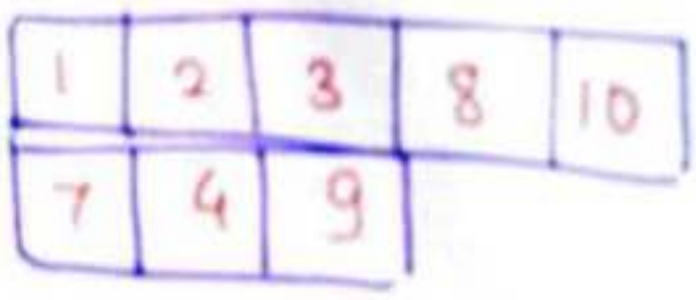


T.C = O(n1 x m1)

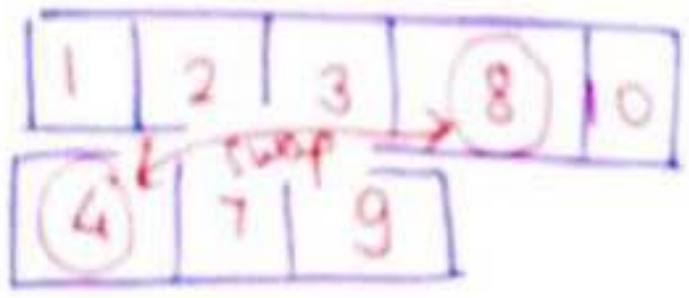
S.C = O(1)

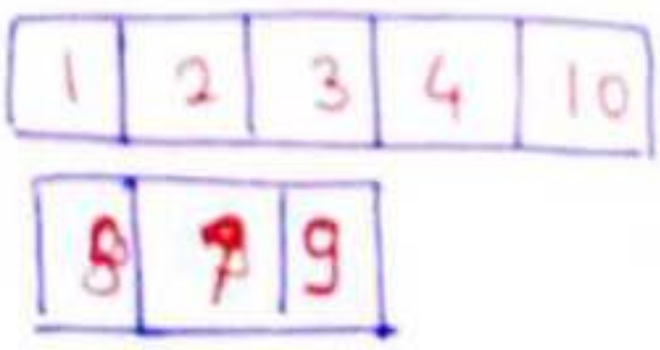


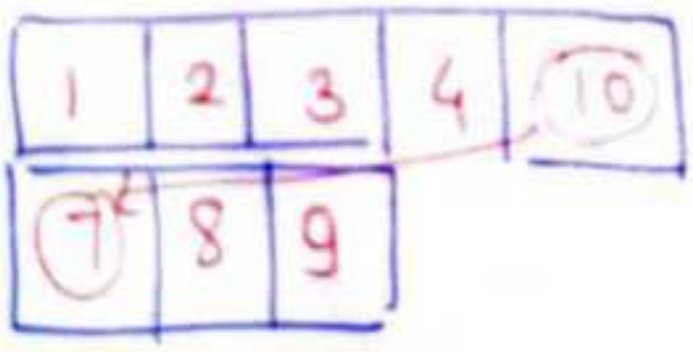
Sort



⇒

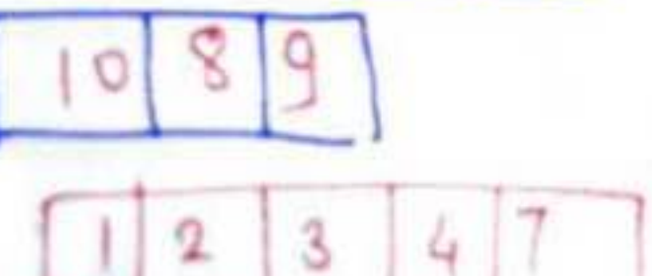






⇒

Sort ⇒



✓ Answer

(10) Find the duplicate Number :-

Given an array of integers 'nums' containing `n+1' integers where each integer is in range [1,n] inclusive.

There is only 'one repeated number' in nums, return its repeated number.

Example :- nums = [1,3,4,2,2]

output = 2

Approach :-

① Sort the nums array,
then we can iterate through nums array

```
for(i=1 to n)
  if(nums[i] == nums[i-1])
      return nums[i];
```

⇒ O(nlogn + n)
space = O(1)
Complexity

② Using the frequency array.

Count the frequency of each element in the array & check if the frequency of any element is > 1, return that element

T(n) = O(n)
S(n) = O(n)

```
vector<int> freq(n+1)
for(i=0 to n)
  if(freq[nums[i]] == 0) → freq[nums[i]]++;
  else → return nums[i];
```

most-optimized method :-

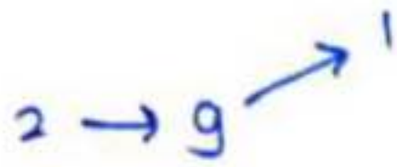
Linked-List Cycle method

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

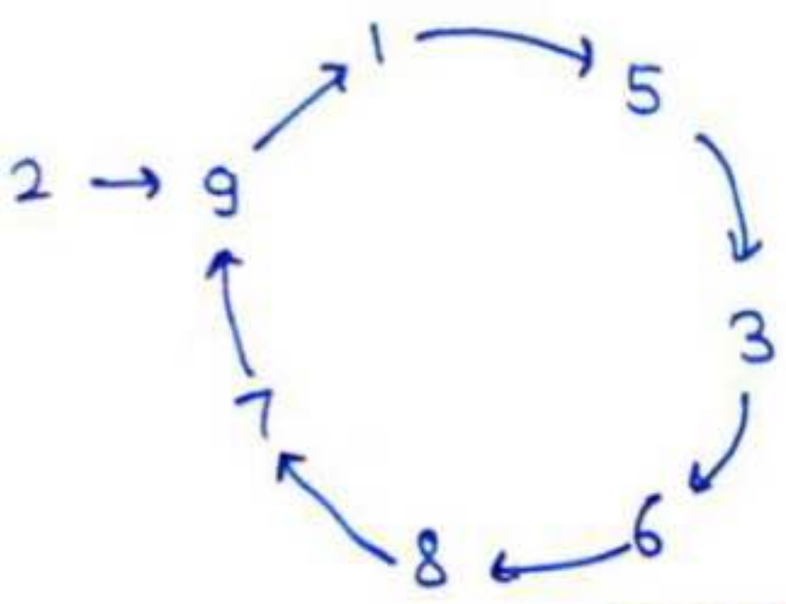
2 → then we take element present at index → 2
⇒ 9

2 → 9

then take element of 9th index

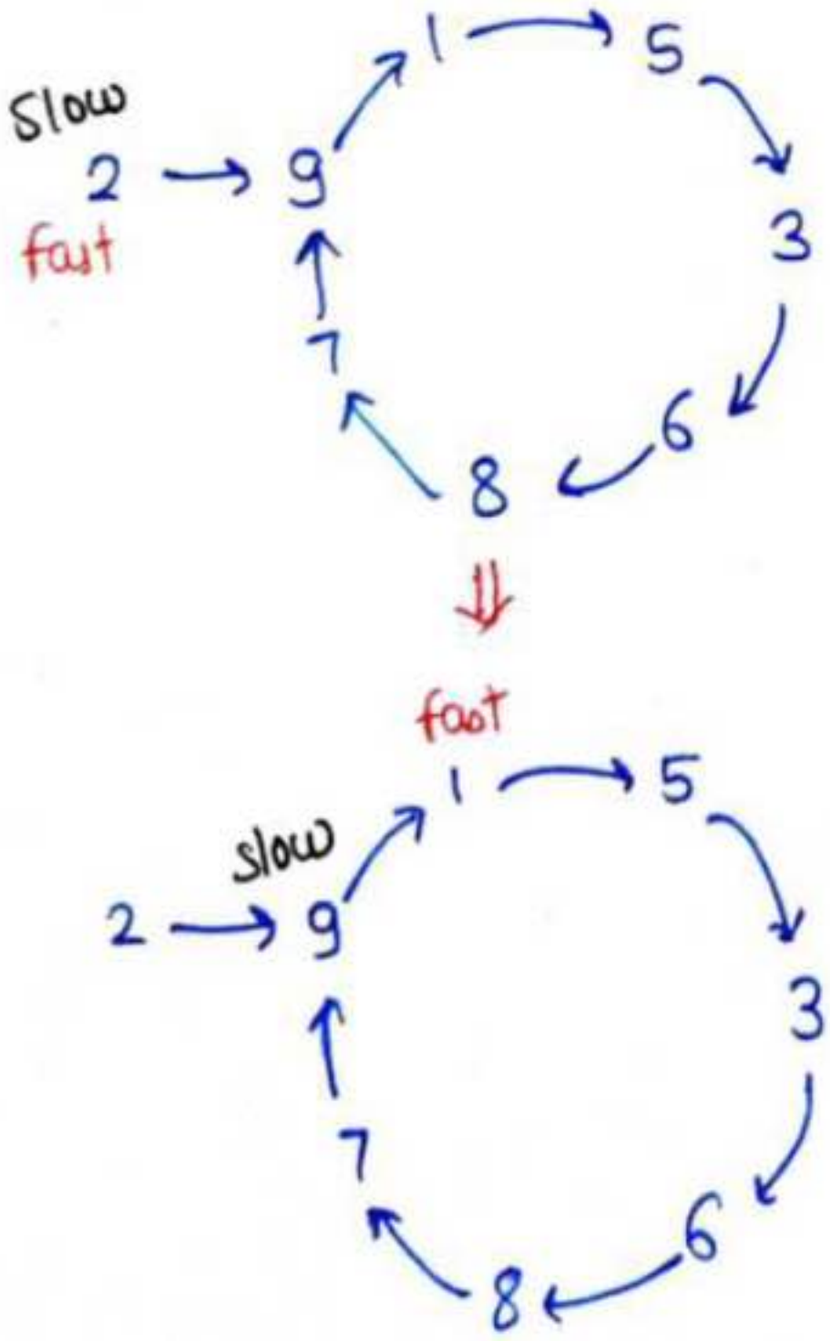


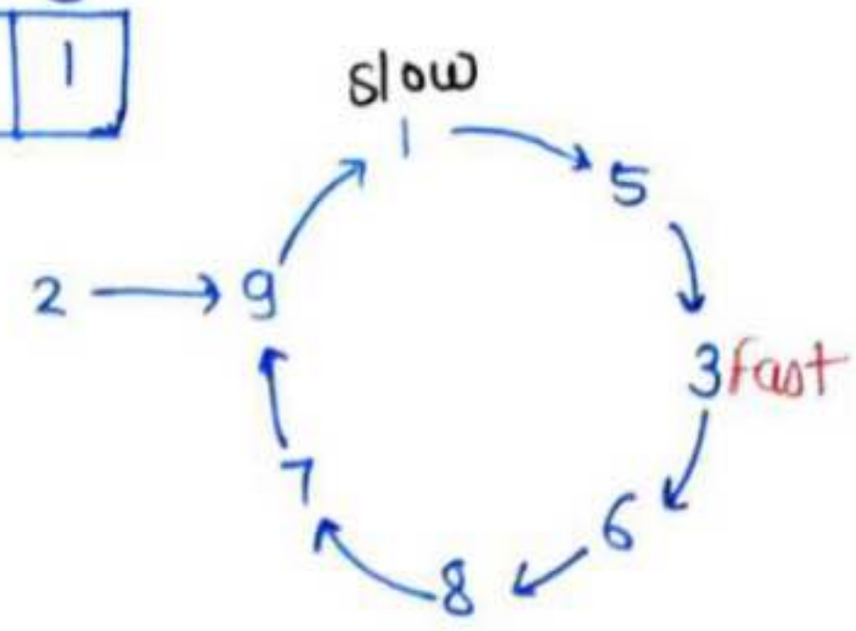
like wise



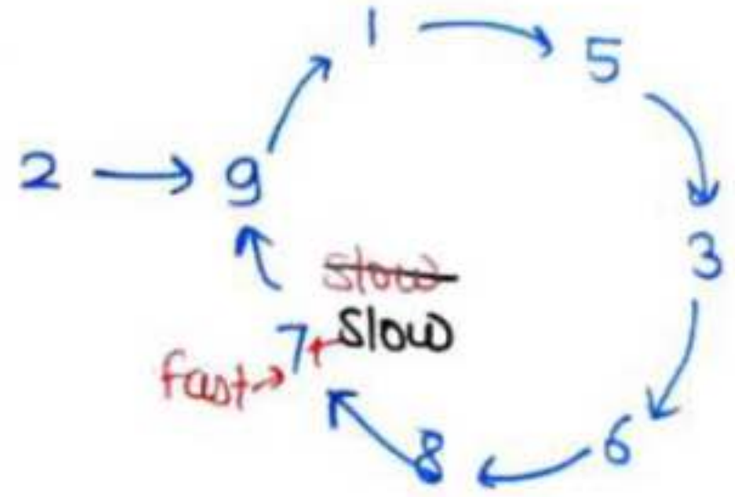
Now, we will apply **tortoise-method**

slow → it moves 1 step
pointer → it moves 2 step

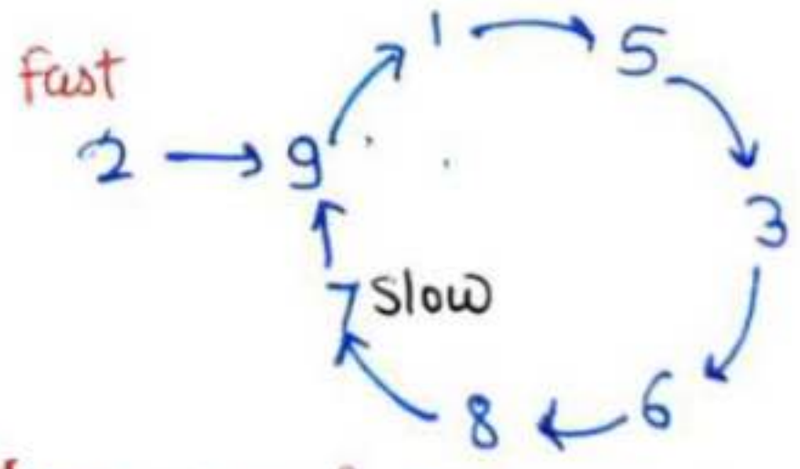




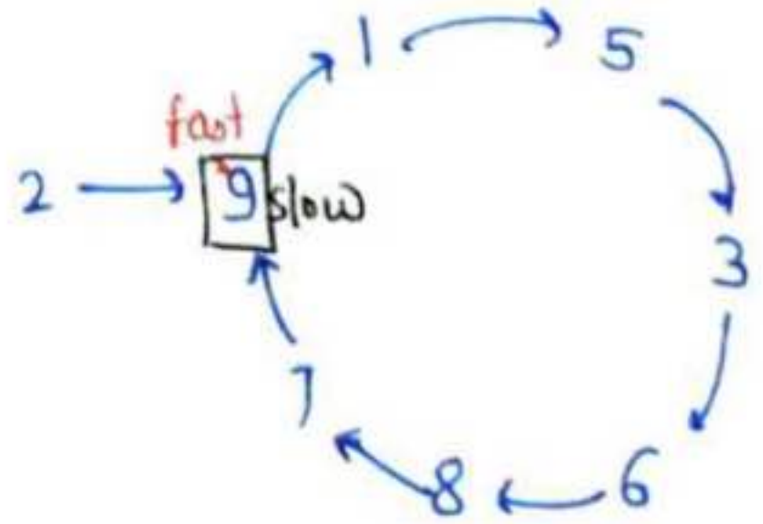
this process takes place 9 again
4 again



Now we will place the fast pointer at first



Move slow & fast pointer by +1



Return 9.

Tc = O(N)
Sc = O(1)

```
slow = nums[0];
fast = nums[0];

while (slow != fast) {
        slow = nums[slow];
        fast = nums[nums[fast]];
    }
Fast = nums[0];
while (slow != fast) {
        slow = nums[slow];
        fast = nums[fast];
    }
return slow;
```

(11) Find the repeating and missing numbers :-

You are given an array of N integers with the values in the range [1, n] both inclusive. Each integer appears exactly once except A which appears twice & B which is missing.

Find the repeating no. & the missing no.

Example 1-

nums = [3, 1, 2, 5, 3]

Result = {3, 4}

Brute force :- Take one substitute Array of size (N+1) and initialize it with 0 then add the values to the substitute array (frequency) and find if the frequency == 0 → missing No.
& frequency > 1 → Repeating NO.

```
for (i=0 to n)
 {  Substitue [Array[i]] ++;
 }
for ( 1 to n)
 {  if (substitute [i] == 0  ||  Substitute [i] > 1)
        {  ans.push_back (i);
        }
   }
```

Time Complexity = O(N) + O(N) ≈ O(N)

Space Complexity = O(N)

RAISONI GROUP
— a vision beyond —

Approach 2 :-

Using Maths

let the missing no. → x
& the repeating no. → y

S = Sum of Natural no. from 1 to n

$$S = \frac{n(n+1)}{2} \quad \text{——— (I)}$$

P = Sum of Squares of natural no.

$$P = \frac{n(n+1)(2n+1)}{6} \quad \text{——— (II)}$$

Now for Array, also calculate Sum → S_1 ——— (III)
& Sum of Squares → P_1 ——— (IV)

then Subtract the

Sum of elements of Array	−	Sum of natural No. from 1 to N

⇓ gives

$$(x-y) = S - S_1 = S'$$

&

$$x^2 - y^2 = P - P_1 = P'$$
$$(x+y)(x-y) = P'$$
$$(x+y) * S' = P'$$
$$x+y = \frac{P'}{S'}$$

Now
$x - y = S'$
$x + y = P'/S'$

$$2x = S'\left(1 + \frac{1}{P'}\right)$$

$$x = \frac{S'}{2}\left(1 + \frac{1}{P'}\right)$$

and find y also

T.C = O(N)
S.C = O(1)

Solution 23 :-

(12) Inversion of Array :-

for a given integer array of size 'n' containing all distinct values, find the total no. of inversions that may exist

A pair (arr[i], arr[j]) ⇒ said to be an inversion when

a) arr[i] > arr[j]

and i < j

Ex :- nums = {5, 3, 2, 1, 4}

There are 7 pairs

(5,1), (5,3), (5,2), (5,4), (3,2), (3,1), (2,1)

* For the Array which is sorted in decreasing order, the maximum inversions = $\frac{n*(n-1)}{2}$

Approach

Brute force :-

Traverse through array with two loops

and check if(arr[i] > arr[j] && i<j)

count++;

TC = O(n²)

Sc. = O(1)

Day - 03

(13) Search in a Sorted 2D matrix :-

Given a 'm*n' 2D matrix and an integer write a program to find the particular integer exists

Input :-

$$matrix = \begin{bmatrix} [1,3,5,7], \\ [10,11,16,20], \\ [23,30,34,60] \end{bmatrix} \qquad target = 3$$

Output = True

Approach

① Brute force :- We can traverse through the 2D matrix & if we found the target element, return true.

TC = O(m*n)
SC = O(1)

```
for (i=0 ; i<mat.size() ; i++)
    for (j=0 ; j<mat[0].size(); j++)
        if (mat[i][j] == target)
            return true;
```

② Binary Search :-

	0	1	2	3
0	1 (0)	3 (1)	5 (2)	7 (3)
1	10 (4)	11 (5)	16 (6)	20 (7)
2	23 (8)	30 (9)	34 (10)	50 (11)
3	56 (12)	64 (13)	76 (14)	86 (15)

target = 30

low = 0 high = 15

$mid = \frac{0+15}{2} = 7$ ↓ so

$7/column = 7/4 = 1 \Rightarrow (1,3)$

$7\%4 = 3$

so it will present in the right part (20 < target)

low = 8, high = 15

$mid = \frac{8+15}{2} = 12$ → 56 element

$12/4 = 3$ → 56 present at (3,0)

$12\%4 = 0$

But our target element present at left part

low = 8, high = 11

$mid = \frac{8+11}{2} = \frac{19}{2} = 9$ → element = 30

Matches with the target element return true.

```
int n = matrix.size();
int m = matrix[0].size();

int low = 0 , high = (n*m) - 1;
while (low <= high) {
    int mid = (low + (high - low) / 2);
    if (matrix[mid/m][mid%m] == target) → return true
    else if (matrix[mid/m][mid%m] < target)
        low = mid + 1;
    else → high = mid - 1;

return false;
```

T.C = O(log(m*n))
S.C = O(1)

(14) Pow(x,n)

Given a double 'x' & integer 'n', calculate x raised to power 'n' basically implement pow(x,n);

Input :-

x = 2.00000
n = 10
output = 1024.00000

Approach

① Brute force :-

```
double ans = 1.0
for (int i=0 ; i<n ; i++)
        ans = ans * x;
}
return ans;
```

Time Complexity = O(n)
S.C = O(1)

② Optimized Approach :-

* Using Binary Exponentiation

n = +ve / -ve

$2^{10} = (2 \times 2)^5 = 4^5$

$4^5 = 4 \times 4^4$

$4^4 = (4 \times 4)^2 = 16^2$

$16^2 = (16 \times 16)^1 = 256^1$

$256^1 = 256 \times \frac{(256)^0}{1}$

$= 256$

$= 4 \times 256 = 1024$

```
if(n%2==0)
    x=x*x
    n=n/2
(n%2==1)
    ans = ans * x
    n= n-1;
```

```
double ans = 1.0
long long nn = n

if (nn < 0)
    nn = -1 * nn;
while (nn > 0)
    if (nn % 2 == 1) {
        ans = ans * x;
        nn = nn - 1;
    }
    else {
        x = x * x;
        nn = nn/2;
    }
}

if (n < 0)
    ans = double(1.0) / (double)(ans);

return ans;
```

T.C = O(logn)

S.C = O(1)

(15) Find the majority element that occurs more than $(N/2)$ times

Given an array of N integers, write a program to return an element that occurs more than $(N/2)$ times.

Input N = 3 nums = [3, 2, 3]

3 occurs more than $3/2 = 1$ time so

result = 3

Solution

① Brute force :-

Check the count of occurences of all elements of the array one by one.

Start from the first element of the array and if count > floor$(N/2)$ then return that element as the answer.

If not, proceed with the next element inthe array & repeat the process.

Time complexity = $O(N^2)$
Space Complexity = $O(1)$

② Using Hashmap :-

Hashmap (key, value)

⇓

(element, No. of occurrence of element)

```
map <int, int> mp
for (int i=0 ; i<arr.size() ; i++)
    mp[arr[i]]++;

for (auto i : ~~arr~~mp)
    if( i.second > (n/2))
        return i.first ;
```

T.C = $O(n \log n)$
S.C = $O(n)$

Why this intution worked!

there are 4 partitions

7 7 5 7 5 1 → Majority Ele = minority Elewet

7 = 3 times

5 = 2 times
1 = 1 time
3 2 1
= 3

57 → —11—

5577 → —11—

5555 → majority Ele = 5 (4 times appeared)

This is the ans.

T.C = O(N)

S.C = O(1)

③ Most Optimal Solution

Moore's Voting Algorithm

7	7	5	7	5	1	5	7	5	5	7	7	5	5	5	5
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

n = 16

cnt = ~~0~~ 1

el = ~~0~~ 7

Here cout will be zero

```
if (cnt == 0)
    el = a[i];
if (el == a[i])
    cnt++;
else
    cnt--;
```

Now i++; i=1 ⇒ cnt = ~~1~~ 2

a[1] = 7 el = 7

Now i++ ; i=2 ⇒ cnt = ~~2~~ 1

a[2] = 5 el = 7

Now i++; i=3 cnt = ~~1~~ 2

a[3] = 7 el = 7

i++, i=4 cnt = ~~2~~ 1

a[4] = 5 el = 7

i=5, a[5] = 1 cut = ~~1~~ 0

el = 7

count = 0

i++, i=6 cnt = ~~0~~ 1

a[6] = 5 el = 5

i=7 cut = ~~1~~ 0

a[7] = 7 el = 5

count = 0

i=8 cut = ~~0~~ 1

a[8] = 5 el = 5

i=9, a[9] = 5 el = 5

cut = ~~1~~ 2

i=10, a[10] = 7 cut = ~~2~~ 1

el = 5

i=11, a[11] = 7 cnt = ~~1~~ 0

el = 5

count = 0

i=12, a[12] = 5 cut = ~~0~~ 1

el = 5

i=13, a[13] = 5 cut = ~~1~~ 2

el = 5

i=14, a[14] = 5 cut = ~~2~~ 3

el = 5

i=15, a[15] = 5 cut = ~~3~~ 4

el = 5

el = 5 ⇒ This is the answer.

(16) Majority Elements (>N/3 times)

Given an array of N integers. Find the elements that appears more than (N/3) times in the array. IF no such element exists, return an empty vector.

Approach :-

① **Brute force :-** Simply count the no. of appearance for each element using nested loops and whenever you find the count of an element greater than N/3 times, that element will be your ans.

```
for (int i=0 ; i<n ; i++)
    cnt=1
    for (int j= i+1 ; j<n ; j++)
        if (arr[i] == arr[j])
            cnt++

    if ( cnt > (n/3))
        ans. push_back (arr[i]);
```

Time complexity = $O(N^2)$
Space complexity = $O(N)$

② **Using Hashmap :-**

```
unordered_map <int, int> mp
for (int i=0 ; i<n ; i++)
    mp[arr[i]]++;

for (auto i :mp)
    if (i.second > (n/3))
        ans.push_back (i.first);
```

T.C = $O(n)$
S.C = $O(n)$

RAISONI GROUP
— a vision beyond —

③ Extended Boyer Moore's Voting Algorithm

num1 = -1

1	1	1	3	3	2	2	2

num2 = -1

c1 = 0

c2 = 0

```
vector<int> ans
   count1 = count 2 = 0;

if (i=0; i<sz; i++){
    if (nums[i] == num1)
          count1++;
    else if (nums[i] == num2)
          count2++;
}
if (count 1 > (sz/3))
     ans.push_back (num1);
if (count 2 > (sz/3))
     ans.push_back (num2);

return ans;
```

```
for(int el = nums)
{
  if (el == num1) c1++;
else if (el == num2) c2++;
else if (c1 == 0)
        num1 = el;
        c1 = 1;
else if (c2 == 0)
        num2 = el;
        c2 = 1;
else
        c1--;
        c2--;
```

T.C = O(n)

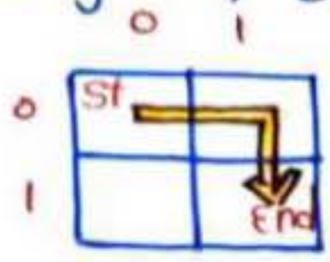
S.C = O(1)

(11) Unique Grid paths :-

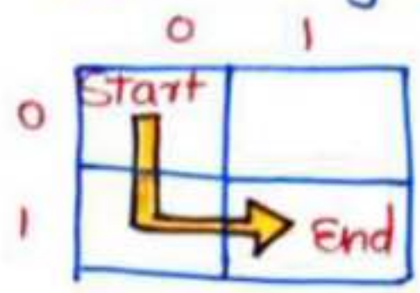
Given a matrix 'mxn', count paths from left-top to the right bottom of a matrix with the constraints that from each cell you can either only move to the rightward direction or to the downward direction.

Example Input = m: 2 n: 2
output = 2

Explanation :- ① Right ⇒ Down



② Down ⇒ right

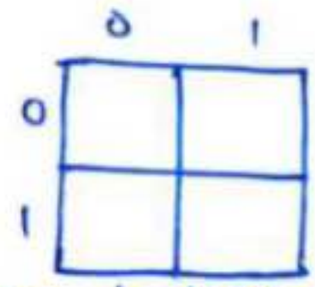


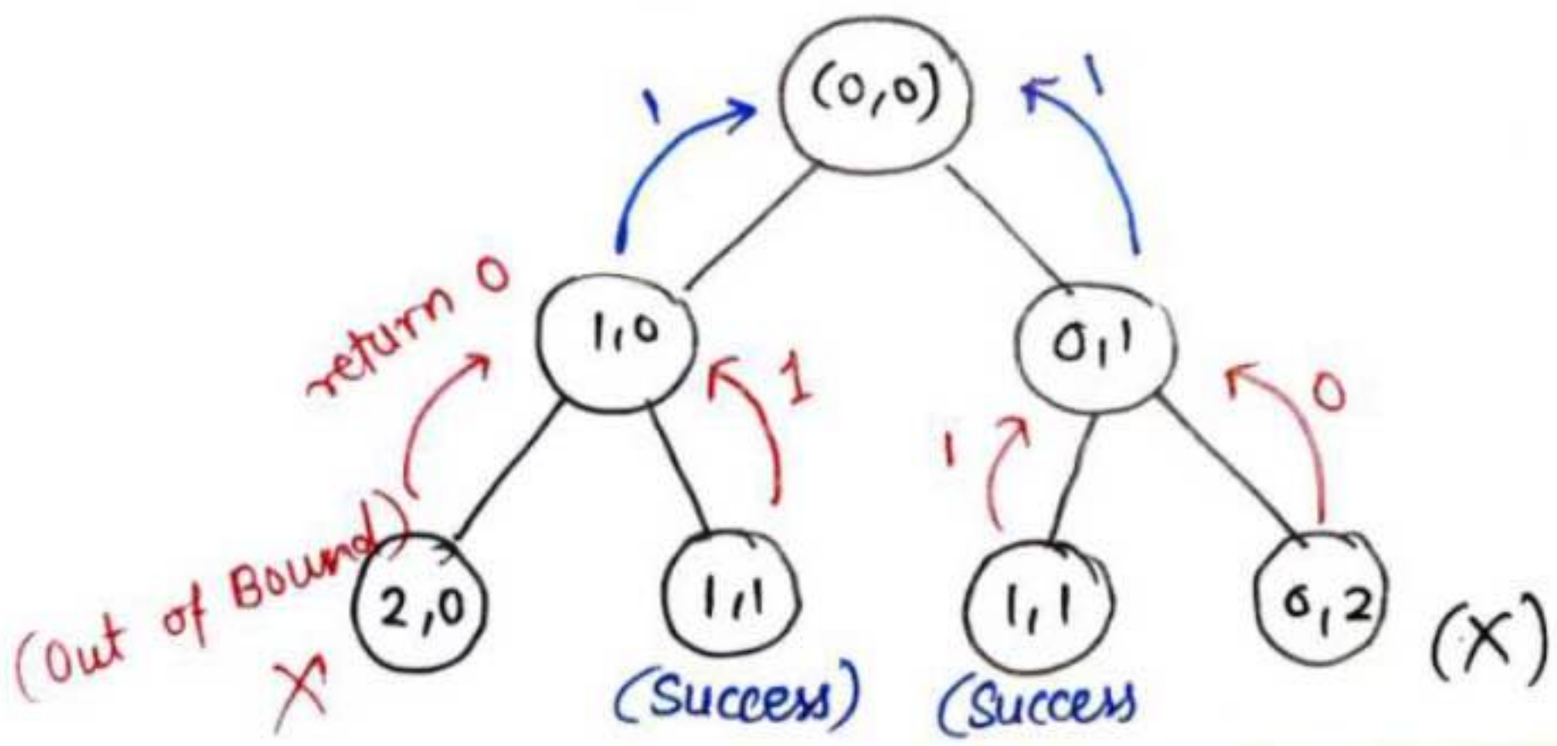
Total ② ways

Approach

① ~~Brute~~ Recursive Approach :-

Initially we are at (0,0)
↓
from here we can go to the right as well as bottom
and we will move until the base case hits





Ans = 2

Time & Space Complexity will be exponential

```
countPaths(int i, int j, int n, int m)
{
  if(i==(n-1) && j==(m-1))
      return 1;
  if (i>=n || j>=m) return 0
  else
  return countPaths(i+1, j, n, m)
         +
         countPaths(i, j+1, n, m);
}
```

RAISONI GROUP
— a vision beyond —

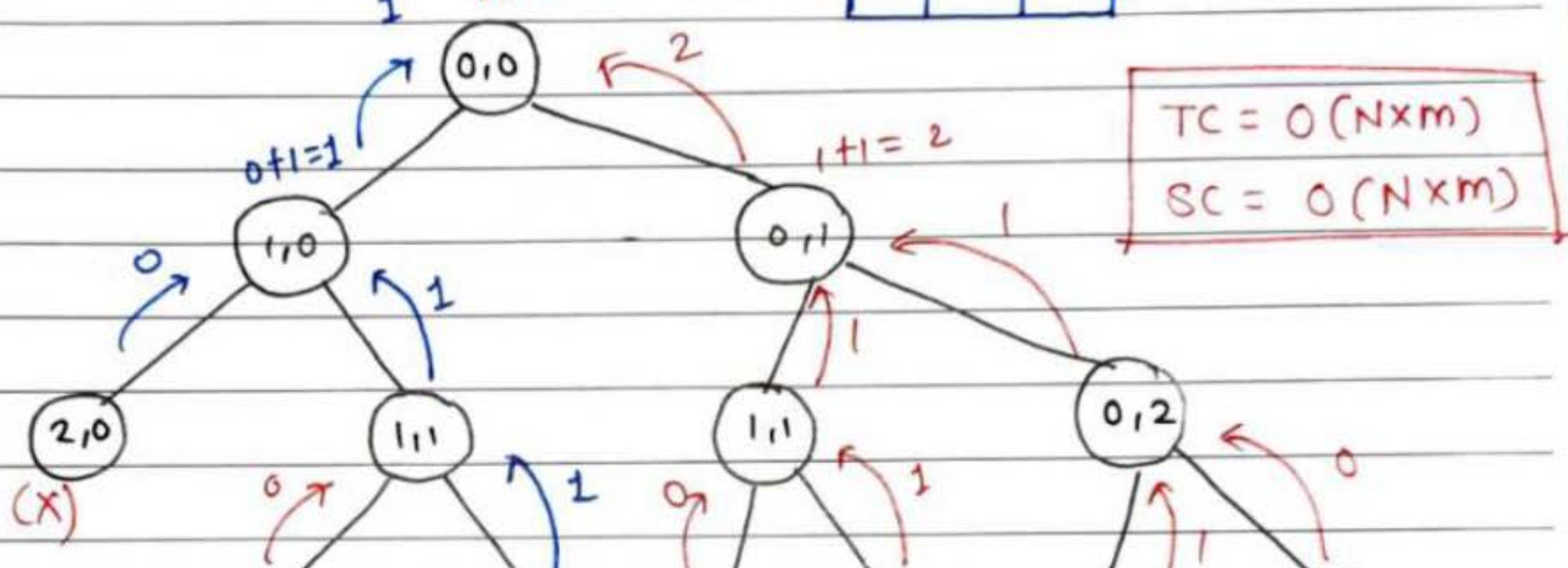
② Dynamic Programming Solution

① Take a dummy matrix A[m][n] and initialize it with "-1".

A[][]

	0	1	2
0	-1	-1	-1
1	-1	-1	-1

Ans = 1+2 = (3)

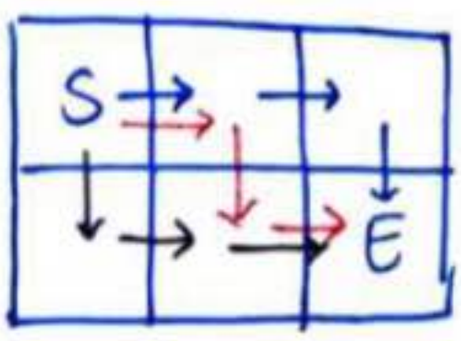


TC = O(N×m)
SC = O(N×m)

```
int countPaths(int i, int j, int n, int m, vector<vector<int>> &dp)
  {
     if(i==(n-1) && j==(m-1)) return 1;
     if(i>=n || j>=m) return 0;
     if(dp[i][j] != -1) return dp[i][j];
     else
      return dp[i][j] = countPaths(i+1, j, n, m, dp)
                        +
                        countPaths(i, j+1, n, m, dp);
  }
int uniquePaths(int m, int n) {
    vector<vector<int>> dp(m+1, vector<int>(n+1, -1));
    int num = countPaths(0,0, m,n, dp);
    if(m==1 && n==1) return num;
    else return dp[0][0];
```

③ Most Optimal Approach :-

Combinatorics method :-



① RRD
② RDR
③ DRR
} ③ ways

① To reach the destination we must have to take certain number of steps to the right & certain number of steps to bottom.

So

possible moves to the right = m-1
possible moves to the down = n-1

Total moves = (m-1 + n-1)
= $\boxed{m+n-2}$ moves

for given Example $m=2$, $n=3$

So $m+n-2 = 2+3-2 = (3)$ places

so

— — —

if I am able to fill these places

$^{m+n-2}C_{m-1}$ { Right paths

or

$^{m+n-2}C_{n-1}$ { Bottom paths

$^{n}C_{r}$

9. $^{10}C_3 = \frac{8 \times 9 \times 10}{3 \times 2 \times 1}$

T.C = O(m-1) or O(n-1)
S.C = O(1)

```
int Total = m+n-2;
int right = m-1;
double ans = 1;

for (int i=1 ; i<=right ; i++)
    ans = ans * (total-right+i) / i ;

return (int)ans ;
```

(18) Count Reverse ~~Paths~~ Pairs :- (HARD)

Given an array of numbers, you need to return the count of reverse pairs.

Reverse pairs :- $arr[i] > 2 * arr[j]$ & $i<j$

Approach :-

① Brute force :-

Two nested loops

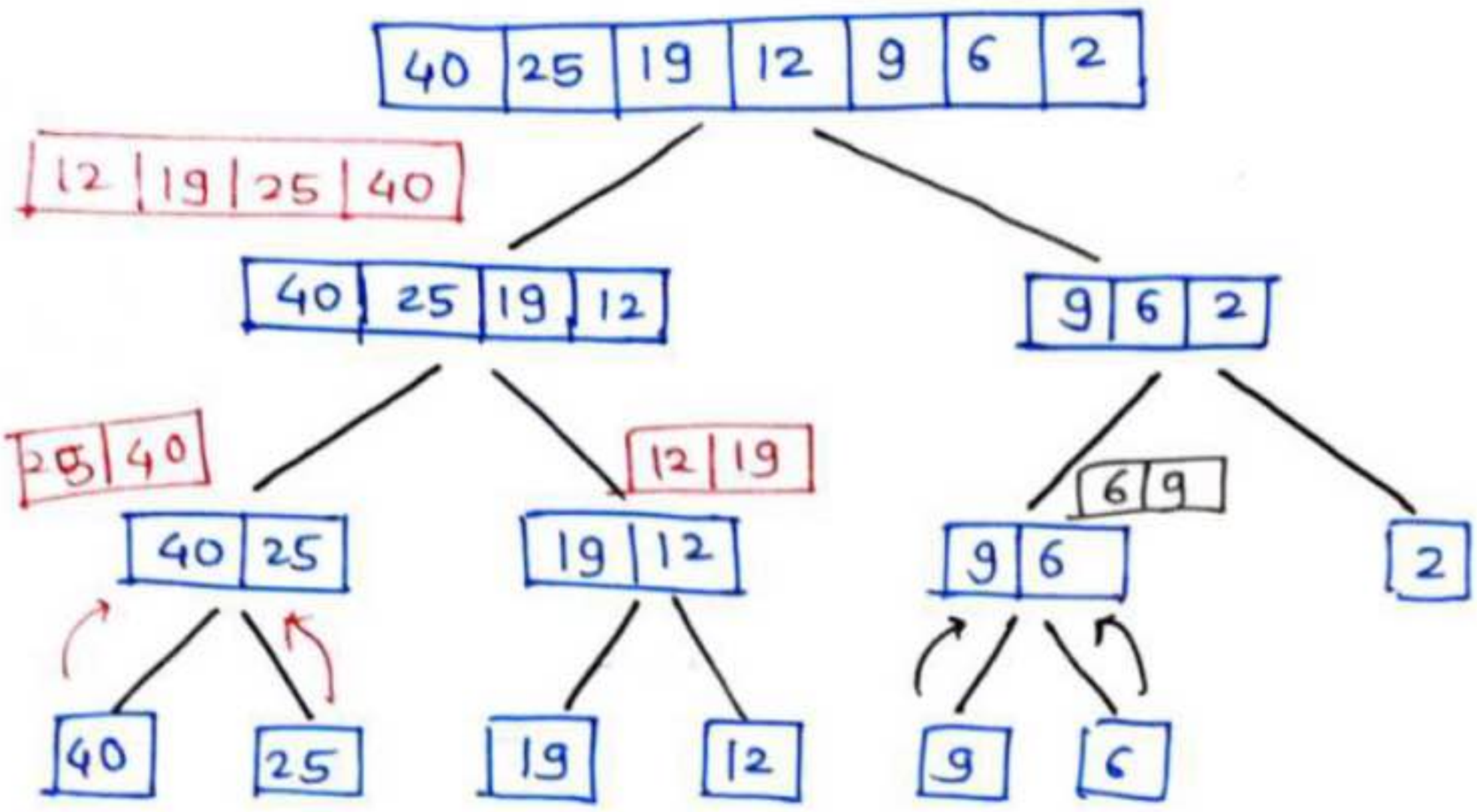
```
for(i=0 to n)
for(j=i+1 to n)
  if(arr[i] > 2 * arr[j])
      count++;

return count;
```

Time Complexity = $O(N^2)$
Space Complexity = $O(1)$

② Optimal Approach

We will use merge Sort.



(i) ↓(j)

→ 40 25

Now, we will place the 'j' to such a index such that $a[i] <= 2 * a[j]$ (keeping 'i' constant)

⇒ j stays over there, so 25 will never contribute to our ans.

Now i move forward ⇒ (i) 19 (j) 12 ⇒ This is also not contributes to our ans

perform merge

Now (i) 25 ⇒(i) 40 (j) 12 ⇒(j) 19

25 <= 2*12 (false)

move j ⇒ j++ ⇒ 19

25 <= 2 * 19 } j moves by one step So ans = 1

40 <= 2 * 19 (false)

Stop. (Right array exhausted)

left array exhausted

perform merge operation

ans ⇒ +(2)

RAISONI GROUP
— a vision beyond —

(i) (j)
9 6 (No contribution)

perform merge step

(i) (j) → j++ (Right array exhausted)
6 9 2

6 <= 2*2 (false) (No contribution) (+1) ans

This is how we have to check.

6 (i)9 (j)2 (Right array) Exhausted)

and then add total no. of elements on the left so add 1 more to the answer (+1) ans

merge step

(i) (j) (j)
12 19 25 40 2 6 9

12 <= 2*2

move j

12 <= 2*6 (✓) (add no. of elements on left of i) ⇒ (+1)

move i ⇒ to 19

19 <= 2*6 (x)

move j

19 <= 2*9 (x) move j ⇒ Right array exhausted
Add # elements on left of j pointer
for (19) ⇒ (+3) ans

move i to 25

25 <= 2*2 (x)
25 <= 2*6 (x)
25 <= 2*9 (x)

} Right array exhausted
Add (+3) ans

for 40 ⇒ (+3)

So ans = 1+2+1+1+1+3+3+3

ans = 15 ⇒ This is no. of pairs

Now merge & Stp

2	6	9	12	19	35	40

Time Complexity = O(nlogn) + O(n) + O(n)
Merge Sort — merge operation — Count operation

Space Complexity = O(n)
Temp. array in merge sort

```
int reversePairs ( vector <int>& nums)
    return mergeSort (nums, 0, nums.size() -1) ;
int mergeSort (vector<int>& nums, int low, int high);
    if (low == high) return 0;
    int mid =(low + high)/2 ;
    int inv = mergeSort (nums, low, mid); // left recursion
      inv += mergeSort (nums, mid+1, high);// Right recursion
   inv += merge (nums, low, mid, high) ⇒(merge function)
   return inv ;
}
```

```
int merge(vector<int>& nums, int low, int mid, int high){
    int cnt=0; // store the total no. of pairs
    int j=mid+1; // put the j index to the first index of
                                        right half
    for(int i=low; i<=mid; i++){
        while(j<=high && nums[i] > 2LL * nums[j]){
            j++;
        }
        cnt += (j-(mid+1)); // count the # elements placed
                                to the left side of 'j'
    }
// merge function
    vector<int> temp;
    int left=low, right=mid+1;
    while(left<=mid && right<=high){
        if(nums[left] <= nums[right]){
            temp.push_back(nums[left++]);
        }
        else{
            temp.push_back(nums[right++]);
        }
    }

// if any array exhausted, then:
    while(left<=mid){                          } If left array
        temp.push_back(nums[left++]);          } is left out.
    }

    while(right<=high){
        temp.push_back(nums[right++]);
    }
```

```
// Copy back the temp array to nums array.
    for(int i = low ; i<= high ; i++) {
        nums[i]= temp[ i- low];
    }

    return cnt;
```

Day - IV

(19) **Two - Sum - Problem**

Given an array of intergers "nums" and an integer "target" return indices of the two numbers such that they add upto "target".

Example :- Input : nums = [2,7,11,15], target = 9
output : [0,1]

Solution :-

① Brute force :-

Simply traverse through the nums array

```
for (int i=0 ; i<n ; i++)
  for (int j=i+1 ; j<n ; j++)
      if (num[i] + nums [j] == target) {
          ans.push_back (i);
          ans.push_back (j);
          break;
      }
  if (ans.size() ==2) break;
  return ans.
```

Time complexity = $O(n^2)$
Space Complexity = $O(1)$

(2) Two-pointer Approach

sort the array.

use two variables, each will start from one end of the array and traverse in both direction to find the required sum.

for each element 'i', we try to find the second element

'target - i'

```
vector<int> ans, temp;
 temp = nums ; ⟶ store the nums array in temp array
 sort(temp);
 int left=0 , right = n-1;
  int x,y;
  while (left <right) {
     if (temp[left] + temp[right] == target) {
                x = temp[left];
                y = temp [right];
               break;
     }
    else if ( temp[left] + temp [right] > target)
                right--;

    else
              left++;
  }
// Now Run one more loop to find the indices of x & y in nums array.

for (int i=0 ; i<nums.size() ; i++) {
     if (nums[i] == x )
          ans.push_back (i);
    else if (nums [i] == y)
          ans.push_back (i);
  }
return ans;
```

T.C = O(nlogn)
S.C = O(n)

(3) Hashing (Most Efficient)

We'll use the hashmap to see if there's a value (target - i) that can be added to the current array value (i) to get the sum equals to target.

If (target - i) found in the map we return the current index & index stored at (target - nums[index]) location in the map.

```
unordered_map <int, int> mp;

for (i=0; i<n; i++) {
    if (mp.find (target - nums[i]) != mp.end()) {
        ans.push_back (i)
        ans.push_back (mp[target - nums[i]]);
        return ans;
    }
  mp[nums[i]] = i;
}

return ans.
```

Time complexity = O(n)
Space Complexity = O(n)

(20) # 4 - Sum problem

Given an array of nums
return an array of unique quadruplets, such that

nums[a] + nums[b] + nums[c] + nums[d] == target

Example :- nums = [1, 0, -1, 0, -2, 2] , target = 0

output :- [[-2, -1, 1, 2],
[-2, 0, 0, 2],
[-1, 0, 0, 1]]

- Solutions ⇒

① Using 3 pointers & Binary Search :-

Sort the array

Target = 9

Ex :- Given array :- | 4 | 3 | 3 | 4 | 4 | 2 | 1 | 2 | 1 | 1 |

sort it ⇒ | 1 | 1 | 1 | 2 | 2 | 3 | 3 | 4 | 4 | 4 |

i j k_x

so nums[i] + nums[j] + nums[k] = 1+1+1 = 3

so we have to find = 9 - 3 = (6) in the right half (using binary Search)

'6' is not there in the array. So move k

again (i+j+k) = 1+1+2 = 4 ⇒ 9-4 = (5) find element X

Now when

| 1 | 1 | 1 | 2 | 2 | 3 | 3 | 4 | 4 | 4 |

i j k

Do right-half (Binary Search)

nums[i] + nums[j] + nums[k] = 1+1+3 = 5

we have to find = 9 - 5 = (4)

so we have got | 1 | 1 | 3 | 4 | as one quad

push

| 1 | 1 | 3 | 4 |

starts
i = 0
j = i+1
k = j+1

T.C = O(NlogN) + O(nlogn)
S.C = O(1)

```
set <vector<int>> st;
for ( i=0 to n)
  for (j=i+1 to n)
    for ( k=j+1 to n){
          int x = target - (nums[i]+nums[j]+nums[k])
          if (binary_Search (nums.begin()+k+1 , nums.end(), x))
           {
                  vector < int > quad ;
                  quad . push_back (nums [i]);
                  quad . push_back (nums[j]);
                  quad . push_back (nums [k]);
                  quad . push_back (x);
                  quad sort(quad.begin() , quad.end());
                  st. insert (quad);
           }
    }
vector<vector <int>> ans {st.begin(), st.end());
return ans;
```

② Optimal Approach (Two pointers)

Target = 9

Given array = | 4 | 3 | 3 | 4 | 4 | 2 | 1 | 2 | 1 | 1 |

sort this

left → right

1	1	1	2	2	3	3	4	4	4
i	j								

nums[i] + nums[j] = 1+1 = 2

To find :- (7) ⇒ x

and now (nums[left] + nums[right])

(1+4 < 7) ⇒ so no soln

so left++; (2+4 < 7) ⇒ X

Now

check for this ↓

1	1	1	2	2	3	3	4	4	4

↑ we have checked for this

There is no need to check for this '2'

(nums[left] + nums[right] == x)

(3+4 == 7) ✓

quad ⇒ | 1 | 1 | 3 | 4 |

⇒

1	1	3	4

Now Eliminating duplicates and shifting left & right pointers

right, skip ←, skip ←

1	1	1	2	2	3	3	4	4	4

skip → skip → left

left pointer crosses the right pointer

Now 'j' will jump

1	1	1	2	2	3	3	4	4	4
i				left					right

Remaing value we are = 9 - (1+2) = (6) ⇒ x

looking for

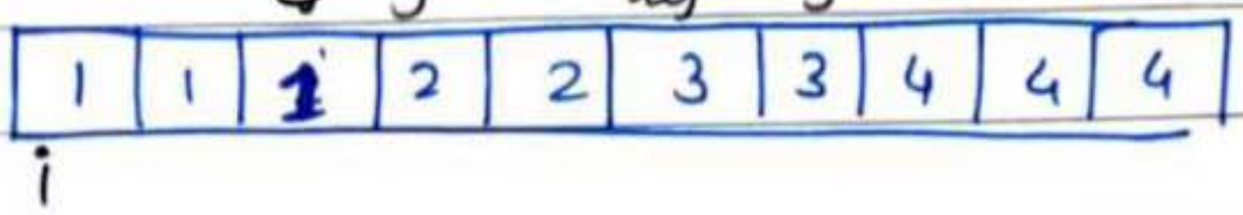
(nums[left] + nums[right] == x)

(2+4 == 6) ✓

quad ⇒ | 1 | 2 | 2 | 4 |

1	2	2	4
1	1	3	4

Now skip the duplicates



(3+3 == 6) ✓

quad | 1 | 2 | 3 | 3 | ⇒

1	2	3	3
1	2	2	4
1	1	3	4

likewise we can find

Time Complexity = $O(n^3)$
Space Complexity = $O(1)$

```
for (i = 0 to n)
    target_3 = target - nums[i]
    for (j = i+1 to n)
        target_2 = ~~target~~ (target_3) - nums[j]

        ~~front = j+1~~      left = j+1 ;
        ~~back =~~           right = n-1 ;
        While (left < right)
            if x = nums[left] + nums[right]
            if (x < target_2) ⇒ left++;
            else if (x > target_2) ⇒ right--;
            else {
                vector<int> quad(4,0)
                push all elements (i, j, left, right)
Avoiding duplicates
                while( left<right && nums[left] == quad[2]) ⇒ left++;
                while( left<right && nums[right] == quad[3]) ⇒ right--;
            }
        }
        while (j+1 < n && nums[j+1] == nums[j])
            j++;
        }
        while (i+1 < n && nums[i+1] == nums[i]) ⇒ i++;
```

(21) Longest Consecutive Sequence in an Array

You are given an array of 'N' integers.
You need to find the length of the longest sequence which contains the consecutive elements.

Ex:- nums = [100, 200, 1, 2, 3, 4]

output = 4

* Solutions

① Brute force :-

```
sort the Array.
int prev = nums[0];
int ans = cur = 1;
for (i=1 to n)
    if (nums[i] == prev + 1) {
            cur++; (Check if the next element is just +1 to the previous)
    }
    else if (nums[i] != prev){
            cur = 1;  // set default
    }
    prev = nums[i]
    ans = max(ans, cur);
}
return ans;
```

Time Complexcity = O(nlogn) + O(n)
≈ O(nlogn)

Space complexity = O(1)

② Optimal Approach :- (using Hashset)

Insert all elements into HashSet;
Now traverse the ~~hashset~~ nums, and will check the (num-1) element is present or not
then we'll do currentNum = num;
currentStreak = 1

then will increase the currentStreak until the (currentNum+1) element is present in the HashSet.

longestStreak = max(longestStreat, CurrentStreak);

```
set<int> hashSet;
for(int num : nums)
    hashSet.insert(num);

longestStreak = 0;
for(int num : nums) {
    if(!hashSet.count(num-1)) {
        currentNum = num;
        currentStreak = 1;
        while(hashSet.count(currentNum+1)) {
            currentNum += 1;
            currentStreak += 1;
        }
    longestStreak = max(longestStreak, currentStreak);
    }
}
return longestStreak;
```

T.C = O(N)
S.C = O(N)

(22) Length of the longest subarray with zero sum

Given an array containing both positive and negative integers, we have to find the length of the longest subarray with the sum of all elements equal to zero.

Ex:- nums = [9, -3, 3, -1, 6, -5]

output = 5

subarrays with sum = 0

[-3, 3] ⇒ len = 2

[-1, 6, -5] ⇒ len = 3

[-3, 3, -1, 6, -5] ⇒ len = 5 (O/P)

- Solutions :-

① Naive Approach :-

① Initialize a variable maxLen = 0 (stores the maximum length of Subarray with sum 0)

② Traverse the array from start and initialise a variable sum = 0 which stores sum of subarray starting with currentIndex

③ Traverse from next element of currentIndex up to n

```
sum += nums[i]
if (sum == 0)
    maxlen = max(maxlen, j - i + 1);

return maxlen.
```

T.C = O(N²)
S.C = O(1)

```
iut maxlen = 0
    for (iut i=0 ; i<n ; ++i)
        sum = 0
        for (iut j= i ; j<n ; j++)
            sum += nums[i];
            if (sum==0)
                maxlen = max(maxlen , j-i+1);

return maxlen ;
```

RAISONI GROUP
— a vision beyond —

② Optimized Approach :-

Maintain (maxi) to store the maxlen of subarray.

	0	1	2	3	4	5	6	7	8	9
nums =	1	-1	3	2	-2	-8	1	7	10	23

Now lineally traverse through the array, and we'll store the prefix sum

HashMap

(key, Value)
(1,0)

At i=0

sum = 1 → Add to hashmap with its index i.e. (1,0)

At i = 1

sum = 0

1	-1

⇒ This subarray with sum = 0

maxi = 2

At i = 2

sum = 3 → Add to hashmap (3,2)
(but before check if "3" exists in the hashmap or not)
(if it does not exist then add.

(key, Value)
(5,3)
(3,2)
(1,0)

At i = 3

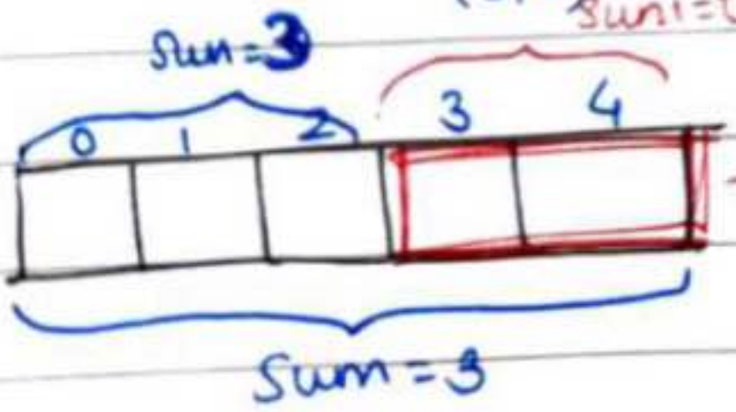
sum = 5 → Add to hashmap (5,3)

At i=4

sum = 5-2 = 3 (It is exists in the hashmap)

So this means that till index ⇒ 2 we had sum = 3

(3,2) sum=0 and till 4 we have sum = 3



⇒ So this is one of the subarray.

its length = the index at which we have got S=3 − the index at which we have got S=3 (previously)

= 4 - 2

= (2)

Now At i=5

sum = 3 - 8
= -5

[check if -5 is present in the hashmap or not
if not add it along with its index]

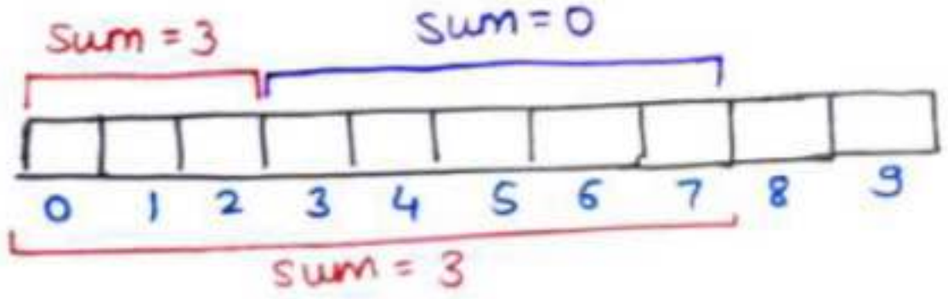
(-4,6)
(-5,5)
(5,3)
(3,2)
(1,0)

At i=6

sum = -5 + 1 Add (-4,6)
= -4

At i=7

sum = -4 + 7
= 3 ⟹ is present



len = 7 - 2
= 5

maxi =
~~len~~ = max(len, maxi)
= max(5, 2)

maxi = 5

At i=8

Sum = 3 + 10
= 13 → Add (13,8) in hashmap

(36, 9)
(13, 8)
(-4,6)
(-5,5)
(5,3)
(3,2)
(1,0)

hashMap

At i=9 sum = 13 + 23 → Add (36,9)
= 36

So the maximum length = 5

```
unordered_map<int, int> mp;
int maxi = 0
int sum = 0
for(int i=0; i<n; i++){
    sum += nums[i];
    if (sum == 0) {
        maxi = i+1;
    }
    else {
        if (mp.find(sum) != mp.end()){
            maxi = max(maxi, i - mp[sum]);
        }
        else {
            mp[sum] = i;
        }
    }
}
return maxi
```

Time Complexity = ~~O(nlogn)~~ O(n) + O(nlogn) ≈ O(nlogn)
Space Complexity = O(n)

(23) **Count the number of Subarrays with given XOR k**

Given an array of integers A and B an integer B, find the total number of subarrays having bitwise **XOR** of all elements equal to B.

Ex:- A = [4, 2, 2, 6, 4] B = 6

o/p = 4

The Subarrays having XOR == B

[4,2]
[4,2,2,6,4]
[2,2,6]
[6]

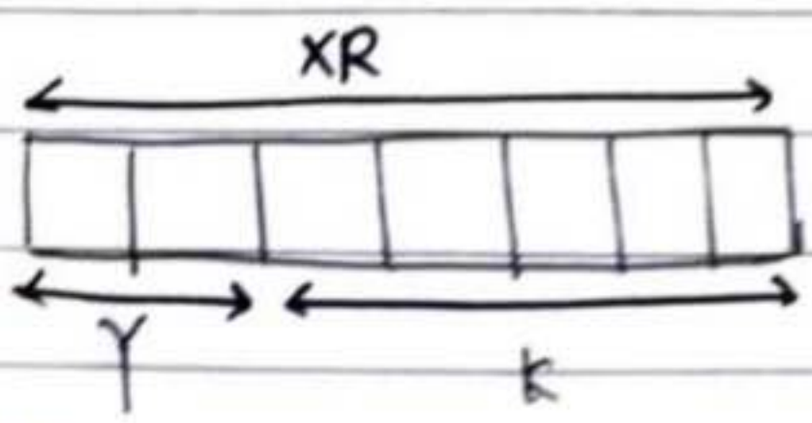
- Solutions ⇒

① Brute force :-

```
long long int count = 0
for(int i=0 ; i<n ; i++){
    int curr_xor = 0;
  for(int j= i  ; j<n ; j++) {
        curr_xor = curr_xor ^ A[i];
        if(curr_xor == B){
            count++;
        }
    }
}
return count ;
```

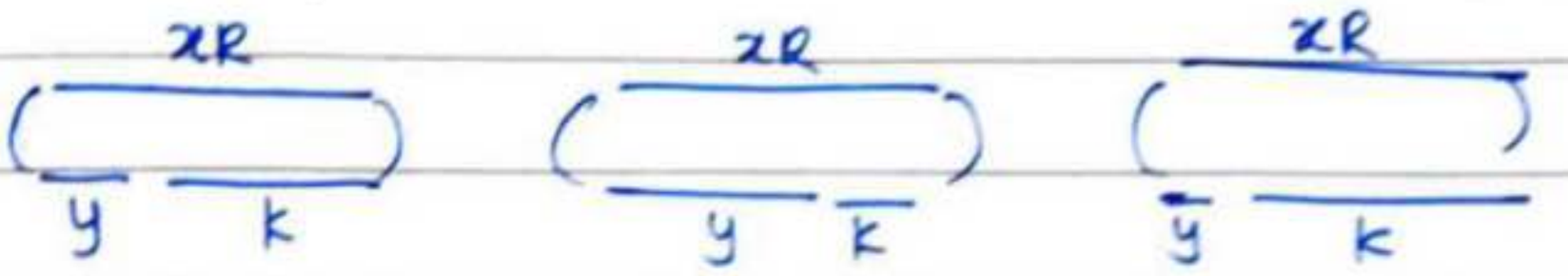
Time Complexity = $O(n^2)$
Space Complexity = $O(1)$

② Optimized Approach :-



~~then~~ $Y \wedge k = XR$

$\Rightarrow$ ~~ie~~ $\boxed{Y = XR \wedge k}$



there are multiple 'y's so

the $\boxed{\text{no. of } y\text{'s} = \text{no. of Subarrays}}$

dry Run :-

	0	1	2	3	4
nums =	4	2	2	6	4

K = 6

Assign XOR = 0, count = 0;

At i=0

XOR = 0 ∧ 4 = 4 → Does this giving k=6 (x)

Add to hashmap with its count

(4,1) → (6,1) (4,1)

Hashmap (prefix_xor, count)

At i=1

XOR = 4 ∧ 2

= (6) ⇒ Yes it is giving

4, 2, 2, 6, 4

increment count ⇒ count++;

$y = XOR \wedge k$

$= 6 \wedge k$

$\boxed{y = 0}$ → Check if '0' prefix_xor is present in the Hashmap or not.

if it is present then we will increment the count

else Add (6,1) to hashmp.

At i=2

XOR = 6 ^ 2

= (4) → y = XR ^ k

= 4 ^ 6

y = 2 → check in the hashmap

~~Add(4,1)~~ Add(4, cnt)

But '4' is present already

So we'll just increment the count

(6,1)
(4,2)

At i=3

XOR = 4 ^ 6

= 2 → y = XOR ^ k

= 2 ^ 6

y = 4 (present!)

XOR = 2

4 | 2 2 | 6 → one subarray → cnt++;

XOR=4 XOR=6

= 2.

XOR = 2

4 2 2 | 6 → another Subarray

XOR = 4 XOR=6

= 2 → cnt++;

(2,1)
(6,1)
(4,2)

Count = 3

At i=4

XOR = 2 ^ 4

= (6) → y = XOR ^ 6

= 6 ^ 6

y = 0

cnt++;

cnt=4

Count = 4

T.C = O(nlogn)

S.C = O(n)

```
cnt=0
XOR=0
for (auto i: A) {
    XOR = XOR ^ i;
    if(XOR == B) {
        cnt++;
    }
    if (freq.find (XOR ^ B) != freq.end()) {
        cnt += freq[XOR ^ B];
    }
    freq[XOR] += 1;
}

return cnt;
```

(24) Length of Longest Substring without any Repeating character

Given a string, find the length of the longest substring without any repeating character.

Ex :- s = "abcabcbb"

o/p = 3

- Solutions

① Brute force :-

Take two loops ⇒

① One for traversing the string

② Another nested loop for finding different substrings

we will check for all substrings one by one & check for each and every element.

if the element is not found then we will store the element in hashset otherwise break from the loop & count it.

~~int ans = 0;~~

```
int count = INT_MIN;
for(int i=0 ; i<n ; i++){
    unordered_set <int> st;
    for(int j=i ; j<n ; j++) {
        if(st.find(s[j]) != st.end()) {
            count = max(count , j-i);
            break;
        }
        st.insert(s[j]);
    }
}
return ~~ans~~ count;
```

Time Complexity = $O(n^2)$
Space Complexity = $O(n)$

② Optimized Approach :-

len = 0

S = " a b c a a b c d b a "
0 1 2 3 4 5 6 7 8 9

Left → a
Right → a

c
b
a

set

At i=0 :-

Check if S[i] is present in the set (Not present)

so Range (L - R) ⇒ This substring has no repeating characters

len = (r-l+1) = 0-0+1 = 1

update len = 1

insert the character into set

move the right pointer

At i=1 :-

'b' is not present

(L-R) ⇒ (r-l+1)

= 1-0+1 = 2

update ⇒ len = 2 (Insert b)

At i=2

'c' is not present in set

(L-R) ⇒ (r-l+1)

= (2-0+1)

= 3

update ⇒ len = 3 (Insert c)

At i=3

'a' is present so we can say that (L-R) range has some repeating characters, so we've to remove that

remove 'a' from set & do `left++;`

(L-R) ✓ (r-l+1)

= [(3-1)+1] = 2+1 = 3 (len = 3) push 'a' to set

At i=4

a b c a a b c d b a

0 1 2 3 4

↳ This has repeating characters

remove ⇒ S[left] ⇒ remove b

then left++;

Now c a a → It still has repeating characters

remove c ⇒ left++;

left

a a

remove 'a'

L

a (It does not have repeating characters)

len = 1

R

But we'll not update the length

push this "a" to set

move the "R++;"

d
c
b
a

At i=5

'b' is not present

len = 2 (No updation)

push b into set

At i=6

'c' is not present

len = 3 (No updation)

push 'c' into set

At i=7

'd' is not present

~~len = 4~~ (L - R) ⇒ (7 - 4) + 1

= 4

update len = 4

push 'd' into set

Now At i=8

a b c d b

~~left++;~~

remove [left] ⇒ remove 'a'

left++;

b c d b

we still have 'b'

remove 'b'

left++;

c d b

len = 3 (No updation)

d
c
b

d
c

b
d
c

Now At i=9

'a' is not present

len = 4 ⇒ (L - R)

= (9 - 6) + 1

= 3 + 1 = 4

push 'a'

a
b
d
c

Time Complexity = O(2*n)

Space Complexity = O(n)

```
int  maxlen = INT_MIN ;
unordered_set <int> st ;
int l=0
for( int r=0 ; r<n ; r++)
  {
     if (st.find(s[r])  != st.end())
      {
        while ( l <r && st.find (s[r])  != st.end())
            {
               st.erase (st [l]);
               l++;
            }
      }
     st. insert (s[r]);
     maxlen = max (maxlen , r-l+1 ) ;
  }

return  maxlen ;
```

③ Most Optimized Approach

use map

L
" a b c a a b c d b a "
0 1 2 3 4 5 6 7 8 9
R

(c,2)
(b,1)
(a,0)

① push 'a' ⇒ (a,0)
len = 1 R++;

② push 'b' ⇒ (b,1)
len = 2 R++;

③ push 'c' ⇒ (c,2)
len = 3 R++;

④ Now 'a' is present in the map ∴ I know 'a' is present at '0' index
so we will shift "L" to (0+1)
~~(L - R)~~
⇓
1st index

L
b c a (Now update the index of a)
R
1 2 3 ⇓ 3

(c,2)
(b,1)
(a,3)

⑤
L
b c a a
R
1 2 3 4
So we will
l = 3+1 = 4
a
↑r

(c,2)
(b,1)
(a,4)

⑥
l
a b
r
range [4-5]

(c,2)
(b,5)
(a,4)

a, b, c

(c,6)
(b,5)
(a,4)

Now at i=7

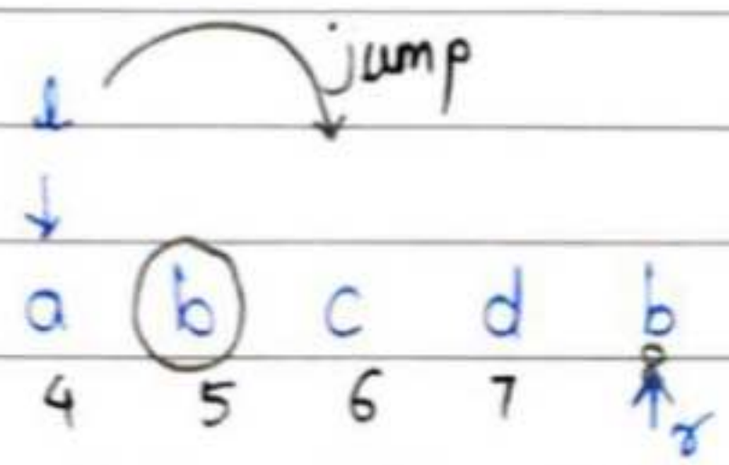
'd' is there (4-7)

length = 7-4+1

= 4

(d,7)
(c,6)
(b,5)
(a,4)

Now at i=8

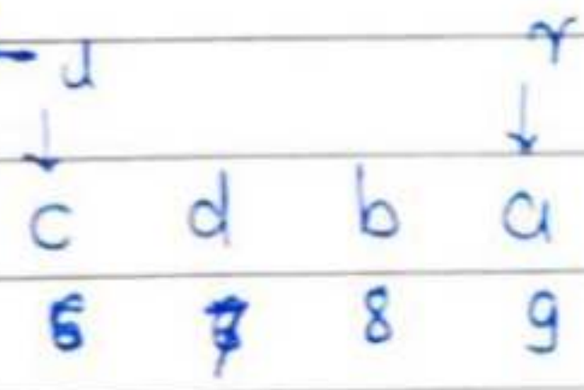


l = 5+1 = 6

c d b

(d,7)
(c,6)
(b,8)
(a,4)

Now at i=9



(6-9) ⇒ length = 4

(d,7)
(c,6)
(b,8)
(a,9)

Time Complexity = O(n)

Space Complexity = O(1) → Avg. case (S.C)

a string can have `256' char

```
vector<int> mp(256, -1);

int left=0, right=0
int len=0

while (right<n){

  if(mp[s[right]] != -1){
     left= max(mp[s[right]]+1, left);
  }
    mp[s[right]] = right;
    len = max(len, right - left +1);
    right++;

}
return len;
```