The logo is a stylized 'L' shape. The left vertical bar is black, and the right vertical bar is red. A horizontal bar connects them in the middle, colored black on the left and red on the right.

LeetCode

STRIVER'S

SDE

SHEET



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 :

<u>Input :-</u>		<u>output</u>																		
<table><tr><td>1</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	1	1	1	1	0	1	1	1	1	⇒	<table><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>1</td><td>0</td><td>1</td></tr></table>	1	0	1	0	0	0	1	0	1
1	1	1																		
1	0	1																		
1	1	1																		
1	0	1																		
0	0	0																		
1	0	1																		

Brute-force approach :-

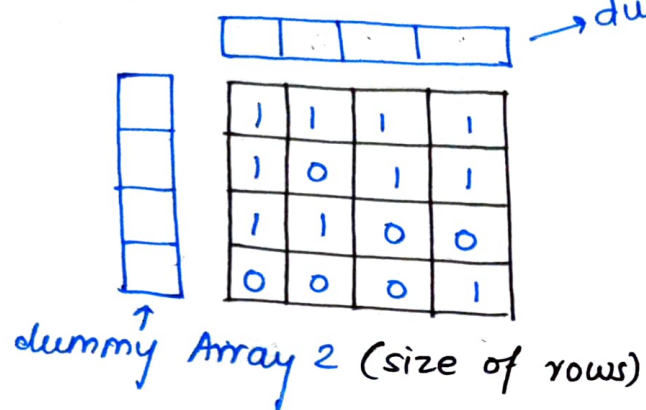
Whenever we find "0" → then go through its row and make all elements ⇒ -1 and same, go through column and make all elements ⇒ -1

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



→ dummy Array 1 (size of columns)

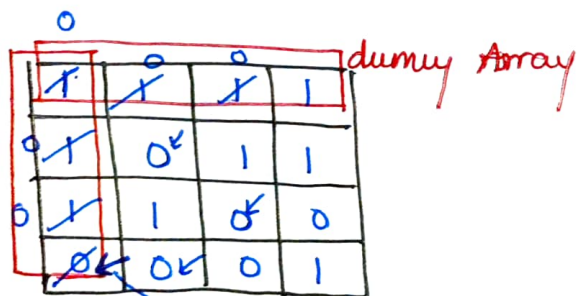
we'll take two dummy arrays

⇒ 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`

Dummy Array → At here, we change ⇒ `col = false;`

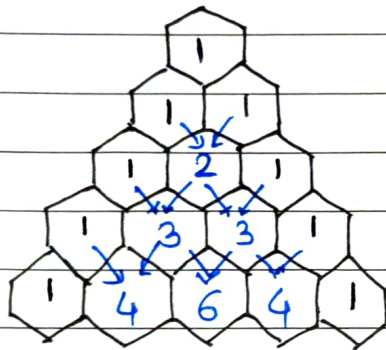
Now traverse from back & check whether for that particular element, zero is present in the dummy column array or dummy row array if yes then convert that element to Zero (0).

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

Space Complexity ⇒ $O(1)$

② Pascals- Triangle :-

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



numRows = 5
Output = $\begin{bmatrix} [1], \\ [1, 1], \\ [1, 2, 1], \\ [1, 3, 3, 1], \\ [1, 4, 6, 4, 1] \end{bmatrix}$

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 Rth row & Cth column i.e 5th row & 3rd column $\Rightarrow$ 6
then formula $\boxed{{}^{(R-1)}C_{(C-1)}}$

Now for the original problem.

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

$\boxed{ans[i][0] = ans[i][i] = 1}$

then

for (j=1; j < i; ++j)



first column element

last column element

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

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 $[1, 2, 3] \Rightarrow$

$[1, 3, 2]$

Brute - Approach

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

$$\text{arr}[i] < \text{arr}[i+1]$$

index 1 = 1

Nxt Perⁿ
 $[1, 3, 5, 4, 2] \rightarrow [1, 4, 2, 3, 5]$

IInd step Again traverse

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~~

IIIrd step :-

Swap($\text{arr}[\text{index 1}]$, $\text{arr}[\text{index 2}]$)

$\hookrightarrow 1, 4, 5, 3, 2$

IVth step :-

reverse($\text{index 1} + 1$, last);

Time complexity $\Rightarrow O(n)$

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 $mx = \max(\text{sum}, mx)$
then whenever sum will be < 0
then set $\text{sum} = 0$
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

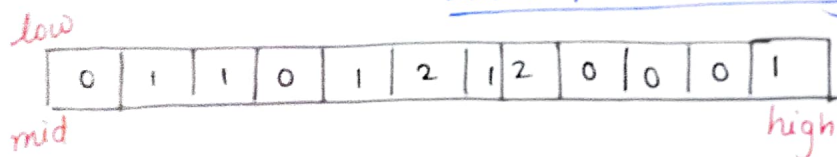
ex:-

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

$[0 \dots \text{low}-1] \Rightarrow$ are "0".

$[\text{high}+1 \dots n] \Rightarrow$ are "2".

Now shift mid pointer

if mid pointer points $\Rightarrow 0 \rightarrow$ then $\left[\text{swap}(\text{arr}[\text{low}], \text{arr}[\text{mid}]) \right]$
low++;
mid++;

if mid pointer points $\Rightarrow 1 \rightarrow$ then mid++;

if mid pointer points $\Rightarrow 2 \rightarrow$ then $\left\{ \text{swap}(\text{arr}[\text{mid}], \text{arr}[\text{high}]) \right\}$
high--;

Day 2

⑦ Rotate Image [Medium]

You are given an ~~2x2~~ $n \times n$ 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 $\Rightarrow$

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

 $\Rightarrow A^T =$

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

$\Leftarrow$ **ANSWER**

⑧ Merge Intervals [Medium]

Given an array of intervals where $\text{intervals}[i] = [\text{start}_i, \text{end}_i]$, 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 $\Rightarrow$ then add to the DS.

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

② Optimized Approach

sort

1,3	2,6	8,10	8,9	9,11	15,18	2,4	16,17
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 $\Rightarrow [1,4]$

Now

1,4
2,6

 $\Rightarrow [1,6]$

then $[1,6]$ & $[8,9]$ they are not merging $\Rightarrow$ At this point

add $[1,6]$ to our DS.
Now

8,9
8,10

 $\Rightarrow [8,10]$ | Again

8,10
9,11

 $\Rightarrow [8,11]$ |

8,11
15,18

 $\Rightarrow$ Add to DS

Now $[15,18] \Rightarrow [15,17]$ |

16,17

 $\Rightarrow$ Add to DS

TC = $O(N \log N) + O(N)$ for traversing

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



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 ~~the~~ m elements from a_3 put in a_1 then take n elements from a_3 put in a_2

$$\begin{aligned} \text{T.C.} &= O(n \log n) + O(n) + O(n) \\ &\approx O(n) \end{aligned}$$

$$\text{S.C} = O(n)$$

Approach:- 2

You can't take extra Space

arr1 =

1	4	7	8	10
---	---	---	---	----

arr2 =

2	3	9
---	---	---

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

if yes then
Swap($arr1[i], arr2[i]$)

1	4	7	8	10
2	3	9		

1	2	7	8	10
4	3	9		

$$T.C = O(n_1 \times m_1)$$

$$S.C = O(1)$$

Sort

1	2	7	8	10
3	4	9		

1	2	3	8	10
7	4	9		

⇒

1	2	3	8	10
4	7	9		

1	2	3	4	10
8	7	9		

1	2	3	4	10
7	8	9		

⇒

1	2	3	4	7
10	8	9		

Sort

1	2	3	4	7
8	9	10		

✓ 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:- $\text{nums} = [1, 3, 4, 2, 2]$
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];
```

$\Rightarrow O(n \log n)$
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)  $\rightarrow$  freq[nums[i]] ++;
```

```
    else  $\rightarrow$  return nums[i];
```

most-Optimized method :-

Linked-List Cycle method

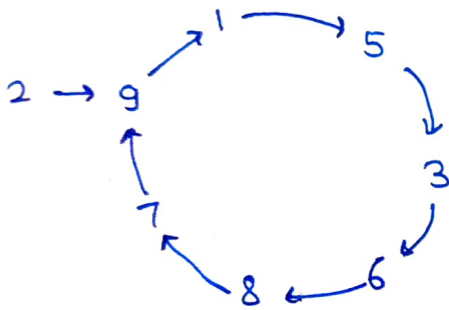
0	1	2	3	4	5	6	7	8	9
2	5	9	6	9	3	8	9	7	1

2 $\rightarrow$ then we take element present at index $\rightarrow$ 2
 $\rightarrow$ 9

2 $\rightarrow$ 9

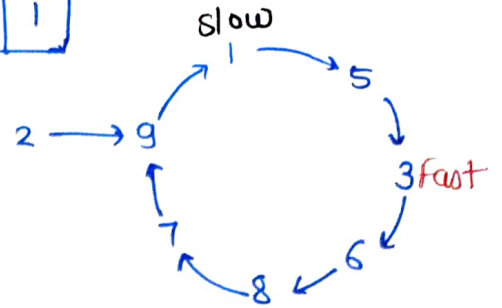
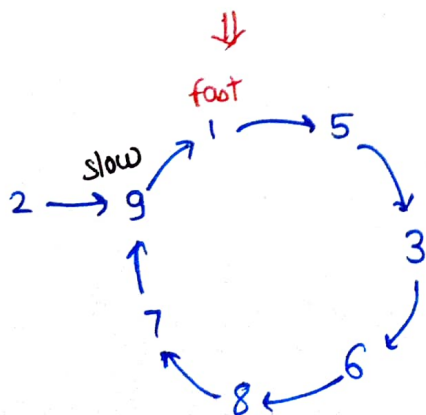
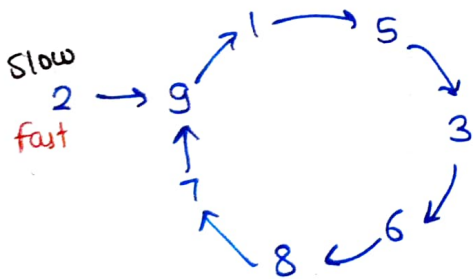
then take element of 9th index

2 $\rightarrow$ 9 $\rightarrow$ 1
like wise

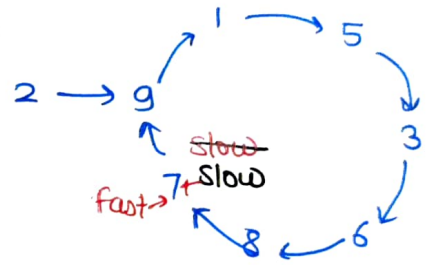


Now, we will apply tortoise-method

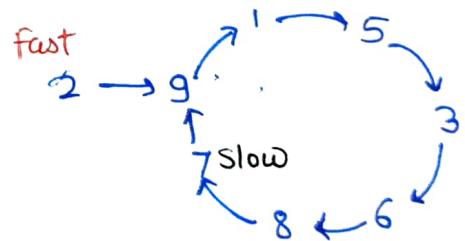
slow $\rightarrow$ it moves 1 step
pointer $\rightarrow$ it moves 2 step



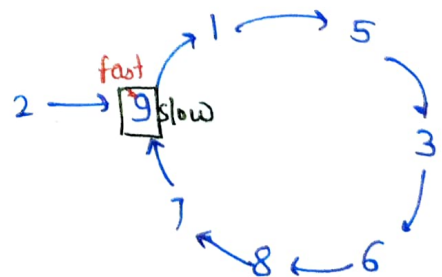
this process takes place 9 again
4 again



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 $= 0 \rightarrow$ missing No.
frequency $> 1 \rightarrow$ Repeating No.

```
for (i=0 to n)
{
    Substitute[Array[i]]++;
}
for (i to n)
{
    if (Substitute[i] == 0 || Substitute[i] > 1)
    {
        ans.push-back(i);
    }
}
```

Time Complexity = $O(N) + O(N) \approx O(N)$

Space Complexity = $O(N)$



Approach 2 :-

Using Maths

let the missing no. $\Rightarrow x$
& the repeating no. $\Rightarrow y$

$S =$ 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 $\Rightarrow S_1$ (iii)

& sum of Squares $\Rightarrow P_1$ (iv)

then Subtract the

$$\begin{array}{l} \text{Sum of elements} \\ \text{of Array} \end{array} - \begin{array}{l} \text{Sum of natural} \\ \text{No. from 1 to } N \end{array}$$

$\Downarrow$

gives

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

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

&

$$x^2 - y^2 = P - P_1 = P'$$

$$(x+y)(x-y) = P'$$

$$(x+y) * S' = P'$$

$$x+y = \frac{P'}{S'}$$

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

$$S.C = O(1)$$

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 $(arr[i], arr[j]) \Rightarrow$ said to be an inversion when

a) $\boxed{arr[i] > arr[j]}$

and $\boxed{i < j}$

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++;

$$TC = O(n^2)$$

$$Sc. = O(1)$$

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

Output = True

Approach

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

TC = $O(m \times n)$
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

$$\text{mid} = \frac{0+15}{2} = 7$$

so

$$7 / \text{column} = 7 / 4 = 1$$

$$7 \% 4 = 3$$

⇒ (1, 3)

so it will present in the right part ($20 < \text{target}$)

low
8

high
15

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

$$\frac{12}{4} = 3 \rightarrow 56 \text{ present at } (3,0)$$

$$12 \% 4 = 0$$

But our target element present at left part

low
8

high
11

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

Ⓐ matches with the target element
return true.

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

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

```
int low = 0, high = (n*m) - 1;
while (low <= high) {
```

```
    if (matrix[
        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;
```


(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 = $\begin{cases} +ve \\ -ve \end{cases}$

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

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

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

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

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

$$= 256$$

$$= 4 \times 256 = \underline{\underline{1024}}$$

if (n % 2 == 0)

x = x * x

n = n / 2

(n % 2 == 1)

ans = ans * x

n = n - 1;

```
double ans = 1.0
```

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

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 $N=3$ $\text{nums}=[3,2,3]$

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

result = 3

Solution

① Brute force :-

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

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

If not, proceed with the next element in the 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 : mp)
    if (i.second > (n/2))
        return i.first;
```

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



Why this intuition worked!

there are 4 partitions

7 7 5 7 5 1

5 7

5 5 7 7

Majority Ele = minority Element = 3

7 = 3 times

5 = 2 times

1 = 1 time

RAISON GROUP
a vision beyond

5 5 5 5

majority Ele = 5 (4 times appeared)

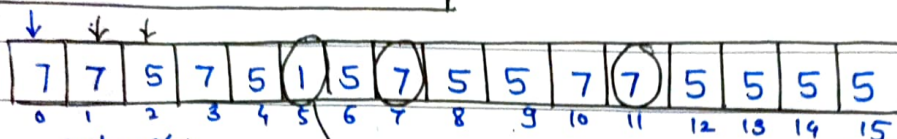
③ Most Optimal Solution

This is the ans.

T.C = $O(N)$

S.C = $O(1)$

Moore's Voting Algorithm



n = 16

cnt = 0

el = 7

Here count will be zero

if (cnt == 0)

el = a[i];

if (el == a[i])

cnt++;

else

cnt--;

Now i++; i=1 $\Rightarrow$ cnt = 2

a[1] = 7

el = 7

Now i++; i=2 $\Rightarrow$ cnt = 1

a[2] = 5

el = 7

Now i++; i=3

a[3] = 7

cnt = 2

el = 7

i++; i=4

a[4] = 5

cnt = 1

el = 7

i=5, a[5] = 1

cnt = 0

el = 7

count = 0

i++; i=6

a[6] = 5

cnt = 1

el = 5

i=7

a[7] = 7

cnt = 0

el = 5

count = 0

i=8

a[8] = 5

cnt = 1

el = 5

i=9, a[9] = 5

el = 5

cnt = 2

i=10, a[10] = 7

cnt = 1

el = 5

i=11, a[11] = 7

cnt = 0

el = 5

count = 0

i=12, a[12] = 5

cnt = 1

el = 5

i=13, a[13] = 5

cnt = 2

el = 5

i=14, a[14] = 5

cnt = 3

el = 5

i=15, a[15] = 5

cnt = 4

el = 5

el = 5

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

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 = count2 = 0;

if (i=0; i<sz; i++) {

if (nums[i] == num1)

count1++;

else if (nums[i] == num2)

count2++;

}

if (count1 > (sz/3))

ans.push_back(num1);

if (count2 > (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 = $O(1)$

① 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

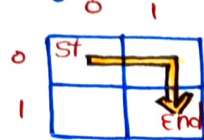
Input = m:2 n:2

output = 2

Explanation :-

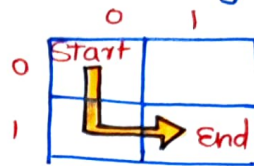
①

Right $\Rightarrow$ Down



②

Down $\Rightarrow$ right



Total ② ways

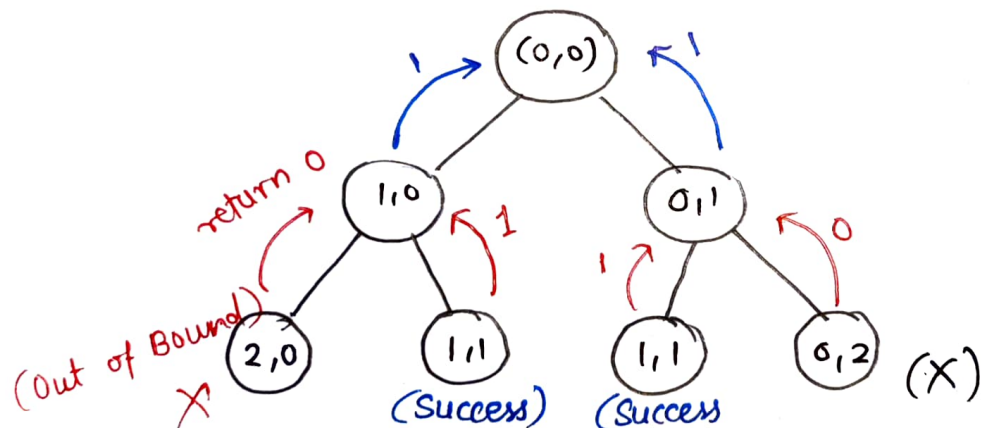
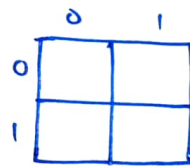
Approach

① ~~Back~~ 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



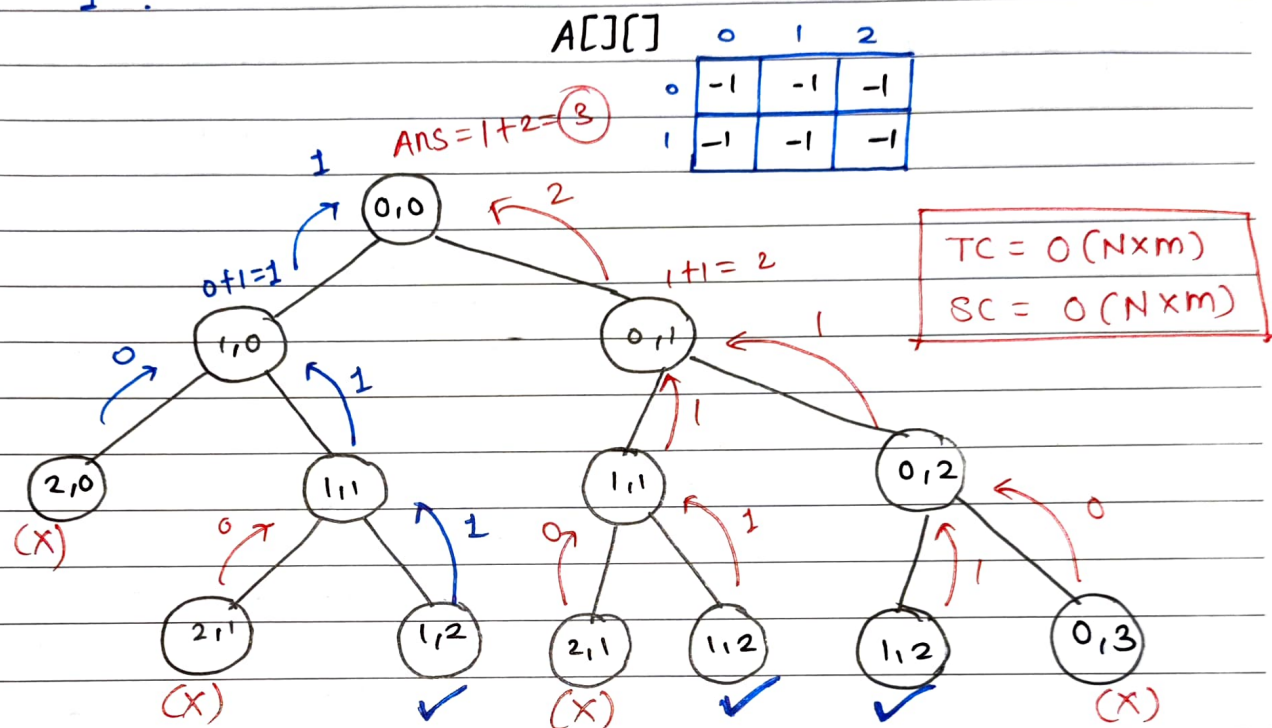
Ans = 2

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);
}
```

② Dynamic Programming Solution

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



```

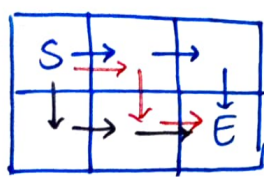
int countPaths(int i, int j, int n, int m, vector<vector<int>> &dp)
{
    if (i == (n-1) && j == (m-1)) return 1;
    if (i >= n || j >= m) return 0;
    if (dp[i][j] != -1) return dp[i][j];
    else
        return dp[i][j] = countPaths(i+1, j, n, m, dp)
            +
            countPaths(i, j+1, n,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

$$\begin{aligned}\text{Total moves} &= (m-1 + n-1) \\ &= \boxed{m+n-2} \text{ moves}\end{aligned}$$

for given Example

$$\begin{aligned}m &= 2 \\ n &= 3\end{aligned}$$

$$\text{So } m+n-2 = 2+3-2 = \boxed{3} \text{ places}$$

so

nC_r

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)

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

$$\begin{aligned}\text{T.C} &= O(m-1) \text{ or } O(n-1) \\ \text{S.C} &= O(1)\end{aligned}$$

```
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;
```


⑮ Count Reverse Pairs Pairs :- (HARD)

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

Reverse pairs :- $\text{arr}[i] > 2 * \text{arr}[j] \quad \& \quad i < j$

Approach :-

① Brute force :-

Two nested loops

```
for (i = 0 to n)
```

```
  for (j = i + 1 to n)
```

```
    if ( $\text{arr}[i] > 2 * \text{arr}[j]$ )
```

```
      count++;
```

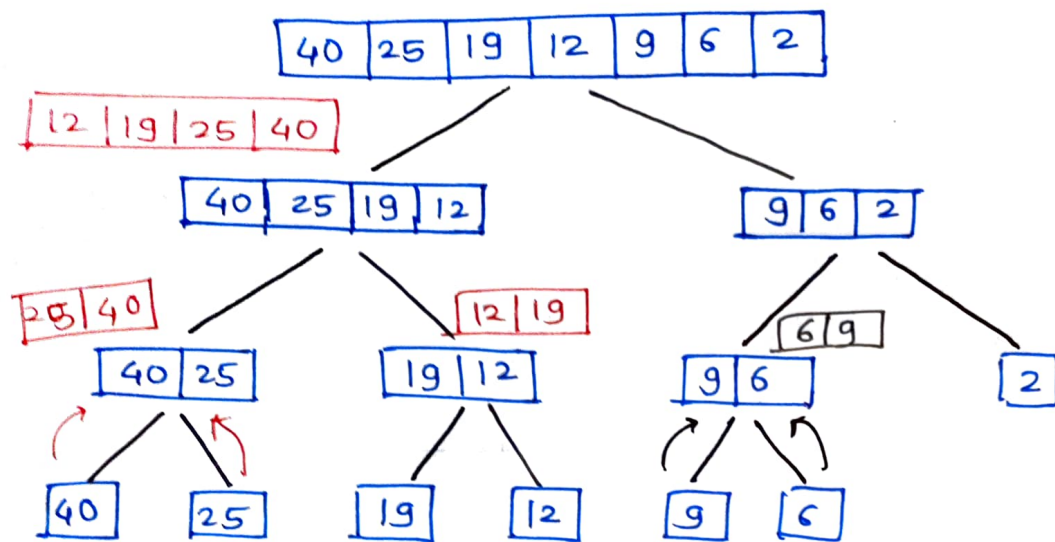
```
return count;
```

Time Complexity = $O(N^2)$

Space Complexity = $O(1)$

② Optimal Approach

We will use merge sort.



(i) $\rightarrow$ 40 $\downarrow$ (j) 25

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

$\Rightarrow$ j stays over there, so 25 will never contribute to our ans.

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

Now

$\downarrow$ (i) 25 $\downarrow$ (i) 40 $\downarrow$ (j) 12 $\downarrow$ (j) 19

perform merge

$25 \leq 2 * 12$ (false)
 move $j \Rightarrow j++ \Rightarrow 19$
 $25 \leq 2 * 19$

} j moves by one step
So ans = 1

$40 \leq 2 * 19$ (false)

Stop. (Right array exhausted)
left array exhausted $\Rightarrow +$ (2)

perform merge operation

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

perform merge step

(i) (j) $\rightarrow j++$ (Right array exhausted)
6 9 2

$6 \leq 2 \times 2$ (false) (No contribution)

(+1) ans

This is how we have to check.

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

(j) (j)
2 6 9

$12 \leq 2 \times 2$

move j

$12 \leq 2 \times 6$ (✓) (add no. of elements on left of
j) $\Rightarrow$ (+1)

move i $\Rightarrow$ to 19

$19 \leq 2 \times 6$ (X)

move j

$19 \leq 2 \times 9$ (X) move j $\Rightarrow$ Right array exhausted

Add # elements on left of j pointer

for (19) $\Rightarrow$ (+3) ans

move i to 25

$25 \leq 2 \times 2$ (X)

$25 \leq 2 \times 6$ (X)

$25 \leq 2 \times 9$ (X)

} Right array exhausted
Add (+3) ans

for 40 $\Rightarrow$ (+3)

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

ans = 15 $\Rightarrow$ This is no. of pairs

Now merge step

2	6	9	12	19	25	40
---	---	---	----	----	----	----

Time Complexity = $O(n \log n)$ + $O(n)$ + $O(n)$
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] > 2 * LL * 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) {  
        temp.push_back(nums[left++]);  
    }  
    while (right <= high) {  
        temp.push_back(nums[right++]);  
    }  
}
```

} if left array is left out.

// Copy back the temp array to nums array.

```
for(int i = low ; i <= high ; i++) {
```

```
    nums[i] = temp[i - low];
```

```
}
```

```
return cnt;
```


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 = $O(n \log n)$

S.C = $O(n)$

③ Hashing (Most Efficient)

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

If $(\text{target} - i)$ found in the map we return the current index & index stored at $(\text{target} - \text{nums}[\text{index}])$ 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

$$\text{nums}[a] + \text{nums}[b] + \text{nums}[c] + \text{nums}[d] == \text{target}$$

Example:- $\text{nums} = [1, 0, -1, 0, -2, 2]$, $\text{target} = 0$

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

Solutions $\Rightarrow$

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 $\Rightarrow$

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

starts

$i = 0$
 $j = i + 1$
 $k = j + 1$

$\uparrow \quad \uparrow \quad \uparrow \quad \uparrow$
 $i \quad j \quad k_x$

so $\text{nums}[i] + \text{nums}[j] + \text{nums}[k] = 1 + 1 + 1 = 3$

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 $(i + j + k) = 1 + 1 + 2 = 4 \Rightarrow 9 - 4 = 5$ find X

Now when

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

 $\uparrow \quad \uparrow \quad \uparrow$
 $i \quad j \quad k$

do right-half (Binary Search)

$\text{nums}[i] + \text{nums}[j] + \text{nums}[k] = 1 + 1 + 3 = 5$

we have to find = $9 - 5 = 4$

so we have got

1	1	3	4
---	---	---	---

as one quad

push

1	1	3	4
---	---	---	---

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

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

left
right

$$\text{nums}[i] + \text{nums}[j] = 1 + 1 = 2$$

To find :- 7 $\Rightarrow x$

and now $(\text{nums}[\text{left}] + \text{nums}[\text{right}])$
 $(1 + 4 < 7) \Rightarrow$ so no soln

so $\text{left}++;$ $(2 + 4 < 7) \Rightarrow x$

Now

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

check for this
↓

we have
checked for this
↑

There is no need to check
for this '2'

$$(\text{nums}[\text{left}] + \text{nums}[\text{right}] == x)$$

$$(3 + 4 == 7) \checkmark$$

quad $\Rightarrow$

1	1	3	4
---	---	---	---

 $\Rightarrow$

1	1	3	4
---	---	---	---

Now eliminating duplicates and shifting
left & right pointers

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

right
skip
skip

skip
skip
left

left pointer crosses the right pointer

Now 'j' will jump

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

left
right

Remaining value we are = $9 - (1 + 2) = 6 \Rightarrow x$
 looking for

$$(\text{nums}[\text{left}] + \text{nums}[\text{right}] == x)$$

$$(2 + 4 == 6) \checkmark$$

quad $\Rightarrow$

1	2	2	4
---	---	---	---

1	2	2	4
1	1	3	4

Now skip the duplicates

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

$(3+3==6) \checkmark$
quad

1	2	3	3
---	---	---	---

 $\Rightarrow$

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)

$x = \text{nums}[\text{left}] + \text{nums}[\text{right}]$

if ($x < \text{target}_2$) $\Rightarrow$ left++ ;

else if ($x > \text{target}_2$) $\Rightarrow$ right-- ;

else {

vector<int> quad(4,0)

push all elements (i, j, left, right)

Avoiding duplicates

while (left < right && nums[left] == quad[2]) $\Rightarrow$ left++ ;

while (left < right && nums[right] == quad[3]) $\Rightarrow$ right-- ;

}

while (j+1 < n && ~~nums[j]~~ nums[j+1] == nums[j])

j++ ;

while (i+1 < n && nums[i+1] == nums[i]) $\Rightarrow$ 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:- $\text{nums} = [100, 200, 1, 2, 3, 4]$
 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 Complexity = $O(n \log n) + O(n)$
 $\approx O(n \log n)$

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~~ the currentStreak until the
(currentNum+1) element is present in the
HashSet.

~~ma~~ $longestStreak = \max(longestStreak, currentStreak);$

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

T.C = $O(N)$

S.C = $O(N)$

```
longestStreak = 0;
```

```
for(int num: hashSetnums) {
```

```
    if(!hashSet.count(num-1)) {
```

```
        currentNum = num;
```

```
        currentStreak = 1;
```

```
        while(hashSet.count(numcurrentNum+1)) {
```

```
            currentNum += 1;
```

```
            currentStreak += 1;
```

```
        }
```

```
        longestStreak = max(longestStreak, currentStreak);
```

```
    }
```

```
}
```

```
return longestStreak;
```


(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] ⇒ ^{O/P} len = 5• Solutions :-① Naïve 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.

int maxlen = 0

for (int i = 0; i < n; i++)

sum = 0

for (int j = i; j < n; j++)

sum += nums[j];

if (sum == 0)

maxlen = max(maxlen, j - i + 1);

return maxlen;

T.C = $O(N^2)$
S.C = $O(1)$



RAISONI GROUP
— a vision beyond —

② Optimized Approach :-

nums =

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

Maintain (maxi) to store the maxLen of Subarray.
HashMap

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

At i=0

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

(1,0)
(key, Value)

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)

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

At i=3

sum = 5 → Add to hashmap (5,3)

At i=4

sum = 5 - 2 = 3 (It 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

0	1	2	3	4
---	---	---	---	---

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

$$\begin{aligned}\text{sum} &= 3 - 8 \\ &= -5\end{aligned}$$

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

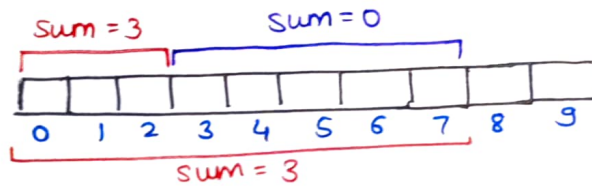
(-4, 6)
(-5, 5)
(5, 3)
(3, 2)
(1, 0)

At i=6

$$\begin{aligned}\text{sum} &= -5 + 1 \\ &= -4\end{aligned}\quad \text{Add } (-4, 6)$$

At i=7

$$\begin{aligned}\text{sum} &= -4 + 7 \\ &= 3 \Rightarrow \text{is present}\end{aligned}$$



$$\begin{aligned}\text{len} &= 7 - 2 \\ &= 5\end{aligned}$$

$$\begin{aligned}\text{maxi} &= \\ \text{len} &= \max(\text{len}, \text{maxi}) \\ &= \max(5, 2)\end{aligned}$$

$$\boxed{\text{maxi} = 5}$$

At i=8

$$\begin{aligned}\text{sum} &= 3 + 10 \\ &= 13\end{aligned} \quad \rightarrow \text{Add } (13, 8) \text{ in hashmap}$$

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

hashMap

At i=9

$$\begin{aligned}\text{sum} &= 13 + 23 \\ &= 36\end{aligned} \quad \rightarrow \text{Add } (36, 9)$$

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 = ~~$O(n \log n)$~~ $O(n) + O(n \log n) \approx O(n \log n)$
Space Complexity = $O(n)$

(23)

Count the number of Subarrays with given XOR k

Given an array of integers A and 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 $\Rightarrow$

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[j];
        if (curr-xor == B) {
            count++;
        }
    }
}
return count;
```

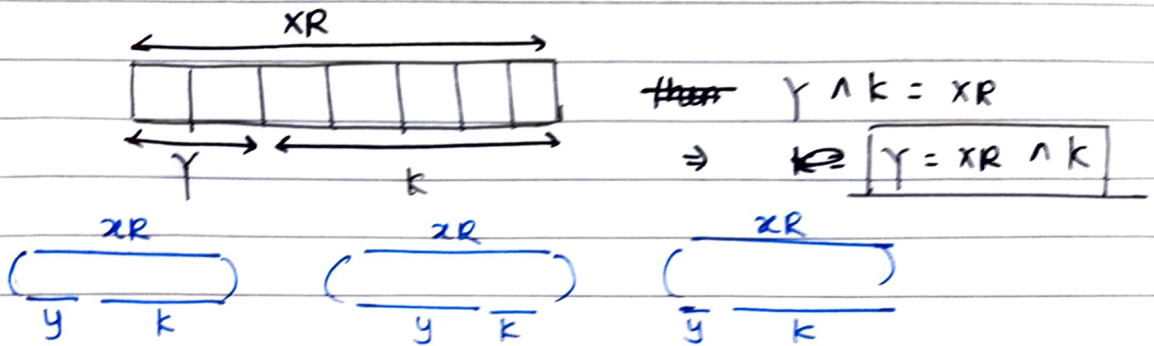
Time Complexity = $O(n^2)$

Space Complexity = $O(1)$



RAISONI GROUP
— a vision beyond —

② Optimized Approach :-



there are multiple 'y's' so

the no. of y's = no. of Subarrays

dry Run :-

nums =

0	1	2	3	4
4	2	2	6	4

 $K=6$

Assign $XOR = 0$, count = 0;

At i=0

$XOR = 0 \wedge 4 = 4 \rightarrow$ 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

$XOR = 4 \wedge 2$

$= 6 \Rightarrow$ Yes it is giving

4, 2, 2, 6, 4

increment count $\Rightarrow$ count++;

$y = XOR \wedge k$

$= 6 \wedge k$

$y = 0$

$\rightarrow$ Check if '0' prefix-xor is present in the Hashmap or not.

if it is present then we will increment the count
 $\uparrow$ else Add (6,1) to hashmap.

At i=2

$$\text{XOR} = 6 \wedge 2 \\ = 4 \rightarrow$$

$$y = \text{XOR} \wedge k \\ = 4 \wedge 6$$

$y = 2 \rightarrow$ check in the hashmap

~~Add(4, 1)~~ Add(4, cut)

But '4' is present already
so we'll just increment the count

(6, 1)
(4, 2)

At i=3

$$\text{XOR} = 4 \wedge 6$$

$$= 2 \rightarrow y = \text{XOR} \wedge k \\ = 2 \wedge 6$$

$y = 4$ (present!)

$\text{XOR} = 2$
[4 2 2 6] $\rightarrow$ one subarray $\rightarrow \text{cut}++;$
 $\text{XOR} = 4$ $\text{XOR} = 6$ $= 2$

$\text{XOR} = 2$
[4 2 2 6] $\rightarrow$ another subarray $\rightarrow \text{cut}++;$
 $\text{XOR} = 4$ $\text{XOR} = 6$ $= 2$

$\text{count} = 3$

(2, 1)
(6, 1)
(4, 3)

At i=4

$$\text{XOR} = 2 \wedge 4 \\ = 6 \rightarrow y = \text{XOR} \wedge 6 \\ = 6 \wedge 6 \\ y = 0$$

cut++;

cut = 4

Count = 4

T.C = $O(n \log n)$
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]++;
}
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:- $s = \text{"abcabcbb"}$

o/p = 3

• Solutions

① Brute force :-

Take two loops $\Rightarrow$

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.

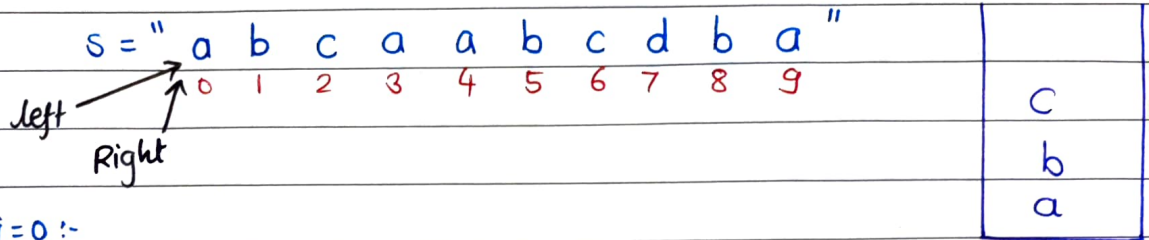
if the element is not found then we will store the element in hashset otherwise break from the loop & count it.

~~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:- $len=0$



At $i=0$:-

Check if $s[i]$ is present in the set (Not present) set
 so Range $(L-R) \Rightarrow$ 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

$$(L-R) \Rightarrow (r-l+1) \\ = 1-0+1 = 2$$

update $\Rightarrow len = 2$ (Insert b)

At $i=2$

'c' is not present in set

$$(L-R) \Rightarrow (r-l+1) \\ = (2-0+1) \\ = 3$$

update $\Rightarrow len = 3$ (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 $left++$

$$(L-R) \Rightarrow (r-l+1) \\ = [(3-1)+1] = 2+1 = 3 \quad \text{len=3} \quad \text{push 'a' to set}$$

At $i=4$

a b c a a b c d b a
0 1 2 3 4

↳ This has repeating characters

remove $\Rightarrow s[\text{left}] \Rightarrow$ remove b

then $\text{left}++;$

Now c a a $\rightarrow$ It still has repeating characters

remove c $\Rightarrow \text{left}++;$

$\downarrow$ left
a a

remove 'a'

$\downarrow$ L
a
 $\uparrow$ R

(It does not have repeating characters)

$\text{len} = 1$

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

$\text{len} = 2$ (No updation)

push b into set

At $i=6$

'c' is not present

$\text{len} = 3$ (No updation)

push 'c' into set

At $i=7$

'd' is not present

~~$\text{len} = 4$~~ $(L-R) \Rightarrow (7-4)+1$

$= 4$

update $\text{len} = 4$

push 'd' into set

Now At $i=8$

a b c d b

~~left++~~
remove [left] $\Rightarrow$ remove 'a'

$\text{left}++;$

b c d b

we still have 'b'

remove 'b'

$\text{left}++;$

c d b

$\text{len} = 3$ (No updation)

Now At $i=9$

'a' is not present

$\text{len} = 4 \Rightarrow (L-R)$

$= (9-6)+1$

$= 3+1 = 4$

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 i=0 ; i<n ; i++)
{
    if(st.find(s[i]) != st.end())
    {
        while(l < i && st.find(s[l]) != st.end())
        {
            st.erase(st[l]);
            l++;
        }
        st.insert(s[i]);
        maxlen = max(maxlen, i-l+1);
    }
}

return maxlen;
```


③ Most Optimized Approach

use map

$\begin{array}{c} L \\ \downarrow \\ " a \ b \ c \ a \ a \ b \ c \ d \ b \ a " \\ \uparrow \\ R \end{array}$

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

$\begin{array}{|l} (c, 2) \\ (b, 1) \\ (a, 0) \end{array}$

① push 'a' $\Rightarrow (a, 0)$

len = 1 R++

② push 'b' $\Rightarrow (b, 1)$

len = 2 R++

③ push 'c' $\Rightarrow (c, 2)$

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)~~

$\downarrow$
 1st index

$\begin{array}{c} L \\ b \ c \ a \\ 1 \ 2 \ 3 \end{array}$
 (Now update the index of a)

$\begin{array}{|l} (c, 2) \\ (b, 1) \\ (a, 3) \end{array}$

③

⑤

$\begin{array}{c} L \\ b \ c \ a \ a \\ 1 \ 2 \ 3 \ 4 \end{array}$

so we will

$l = 3 + 1 = 4$

$\begin{array}{c} L \\ a \\ \uparrow \\ R \end{array}$

$\begin{array}{|l} (c, 2) \\ (b, 1) \\ (a, 4) \end{array}$

⑥

$\begin{array}{c} L \\ a \ b \\ 4 \ 5 \end{array}$

range [4-5]

$\begin{array}{|l} (c, 2) \\ (b, 5) \\ (a, 4) \end{array}$



RAISONI GROUP
— a vision beyond —

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

↓ jump
a (b) c d b
4 5 6 7 8

$$l = 5 + 1 = \textcircled{6}$$

↓
c d b

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

Now at i=9

↓ r
c d b a
6 7 8 9

$$(6-9) \Rightarrow \text{length} = 4$$

(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;
```