# ARRAYS [Level-1]

Page No.
Date

* It is a kind of container / data structure which can store similar types of elements.

Why we need Arrays?

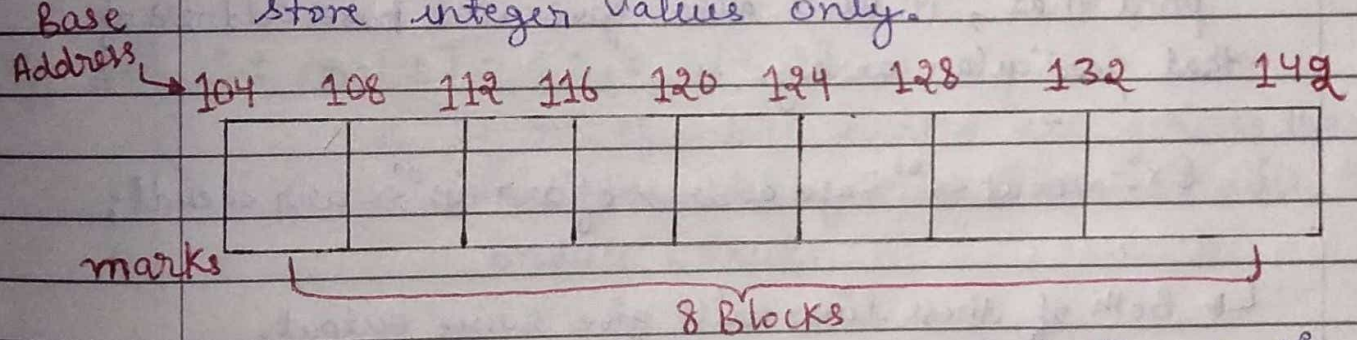
→ If we want sum of 10000 numbers and we have to store them in memory without writing 10000 lines of code to make variables and store them. Then array came into picture which can able to create continuous memory allocations with one single line of code.

Syntax:-

datatype array_name [size];

Ex:- int marks [8];

* This line of code will allocate 8 continuous memory block for the array named 'marks' and it can store integer values only.




Base Address:- It is also called as the starting address of the array

→ We can also refer the first block of array with its name so, we can also name the block with address 104 as 'marks'.

→ Int take 4 bytes to store integers in memory, so this array will take 8 × 4 = 32 bytes space in memory.

→ int arr[20];

↳ This type of array creation is called as static array. which takes fixed number of blocks of memory.

↳ Here, we fixed 20 blocks for array named 'arr'. It will take 20×4 = 80 byte storage space in 64 bit OS.

→ Address of operator :- '&'

* To get the base address of any array, we can use this operator with array name and it will return the hexa-decimal value of that array.

Ex:- int arr[10];
cout << "Base address of arr is:" << &arr << endl;

* Keep in mind - If we only write array name and print that, it will also give the base address of that particular array.

Ex:- cout << "Base address of arr is:" << arr << endl;

↳ Both of these lines will give same output.

→ Size of function :- 'sizeof(array_name)'

↳ This is an inbuilt function of C++ which gives the size of any variable or array passed to it.

Ex:- int a = 5;
int arr[5];
cout << "Size of a:" << sizeof(a) << endl;
cout << "Size of arr:" << sizeof(arr) << endl;

Output:-
Size of a: 4
Size of arr: 20

} int take 4 byte
So, 5×4 = 20 (size of the array)

Page No.
Date

# Array Initialisation :-

**Type I :-** `int arr[] = { 1, 3, 2, 6, 8 }`

∴ Size is not mentioned here, so it will automatically detect the size of array from no. of values.

**Type II :-** `int brr[5] = { 1, 2, 4, 3, 7 }`

∴ Size is mentioned, so we just initialise 5 values in it.

**Type III :-** `int crr[5] = { 2, 5 }`

∴ If no. of values inserted are less than the mentioned size, then it will insert '0' zero in empty blocks.

| 2 | 5 | 0 | 0 | 0 |
|---|---|---|---|---|

**Type IV :-** `int drr[2] = {1, 3, 5, 4, 8}`

∴ If no. of values inserted are greater than the mentioned size of array, then it will produce ERROR.

* If we want to create an array of user specified size, like :-

```
int n;
cin >> n;
int arr[n];
```

∴ This is compiler dependent, it might execute seamlessly or throw error.

↳ But writing this kind of code is considered as BAD PRACTICE, if you have a choice then must specify the size of array.

↳ Because OS will allocate some memory to each program if that integer input exceeds that memory limit then program show WRONG BEHAVIOUR and array can't be created !!

It might happen that our program don't have that much continuous memory which user demanded.
To prevent this case, we use DYNAMIC ARRAYS.

# Indexing in Array :-

→ Array indexing is the same as accessing an array element.

→ You can access an array element by referring to its index number.

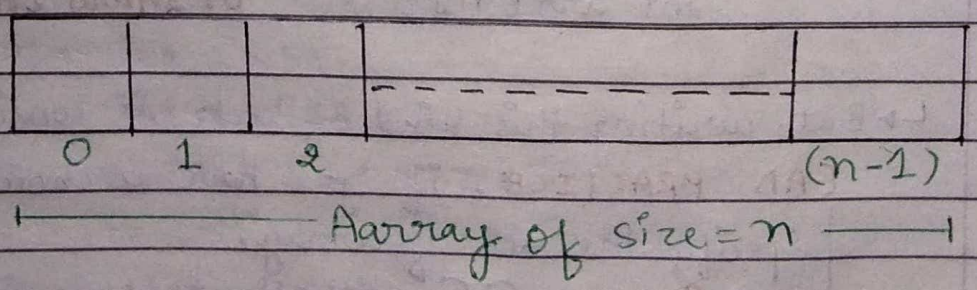
→ Indexing is same as the address of our home in continuous street where we live.

→ To reach at any place, we have to know its address. similarly in arrays, to access any array element, we can use index to get its value.

→ Every block of array have its hex address (hexa-decimal address) which is difficult to learn so, that's why we can use indexing in array.

→ Remember that, indexing starts with '0' in arrays.

So,
if we have an array of 'n' size, then first memory block have 0 index and the last block will have (n-1) index always.



Page No.
Date

Start n=5 End

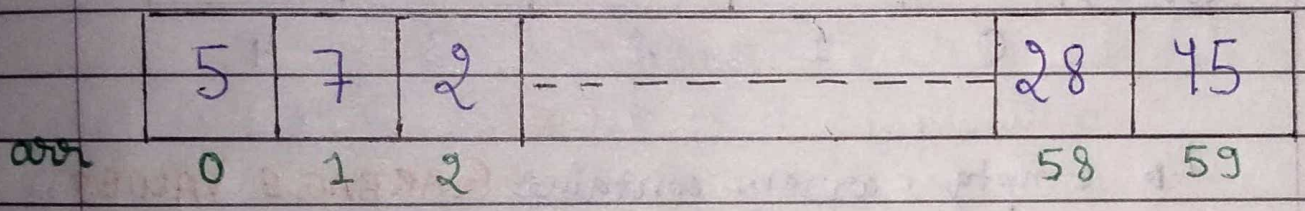
1-Based Indexing: 1 2 3 4 5 → n

0-Based Indexing: 0 1 2 3 4 → (n-1)

∴ To access any memory block of array using indexing, we can just follow this syntax and get the value inside particular block.

array_name [ index_number ]

Ex:- int arr[60];




To access the values of particular index we can write:-

arr[0] = 5

arr[1] = 7

arr[2] = 2

arr[58] = 28

arr[59] = 45

Page No.
Date

Ex:- `int arr[5] = {5, 8, 9, 12, 13}`
`int n = 5;`

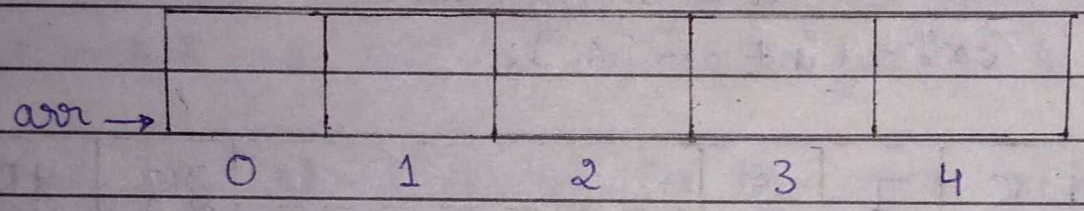
```
for (int i=0; i<n; i++) {
    cout << arr[i] << " ";
}
```

Output:-
5 8 9 12 13

→ Taking input in an Array :-

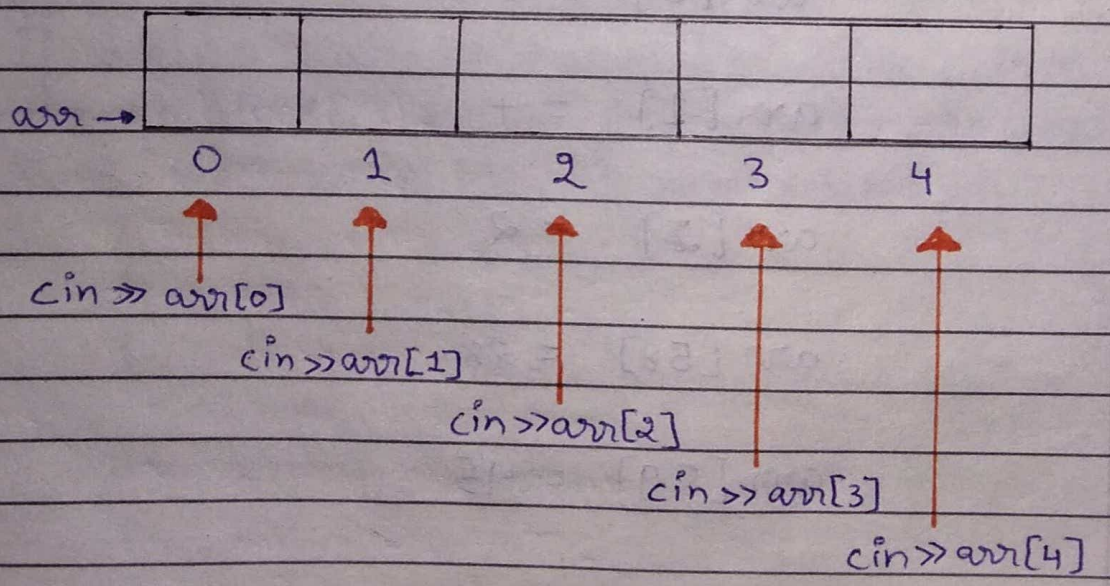
`int arr[5];`

∴ This line will create the continuous memory blocks like :-




↳ Empty array contains GARBAGE VALUES.

↳ If some blocks have values in it, then other empty blocks will initialised with '0'.



→ But, if we want to store 1000 values in an array then we can use loops to store the values without writing 1000 lines of code.

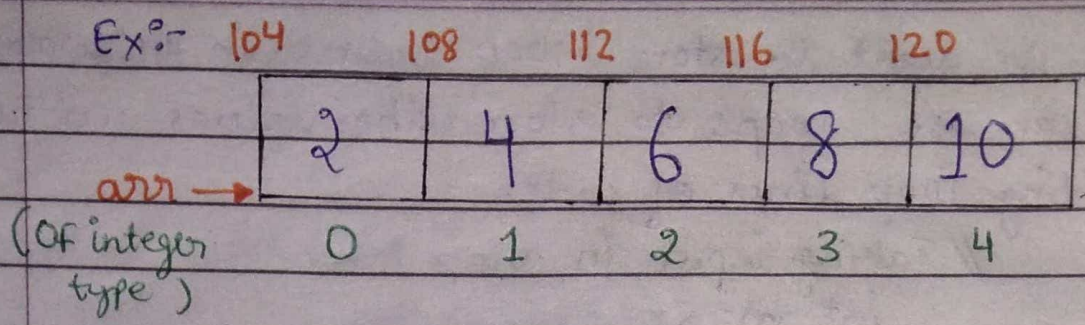
```
// Taking input in arr
    int m=5;
for (int i=0; i<m; i++){
    cin >> arr[i];
    cout << endl;
}
// Printing the arr array
cout<<"Printing the array"<<endl;

for (int i=0; i<m; i++){
    cout << arr[i] << " ";
}
```

Output

```
10
20
30
40
50
Printing the array
10 20 30 40 50
```

# Formula to calculate value of arr[i] -

$$arr[i] = \text{Value at}\left(\underbrace{\text{Base Address} + (\text{Datatype size} * \text{index})}_{\text{Address Calculation corresponding to arr[i].}}\right)$$




If we want to access value of arr[2] then,

arr[2] → Value at (Base Address 104 + (Data Type 4 × Index 2))

→ Value at (104 + 8)

→ Value at (112)

↳ Address of index 2

∴ So, behind the scene memory block is accessed using this formula and then returned the value corresponding to index numer.

Q 1. Make an array of size = 10 then take input values from user and double-up each value of that array.

```
int arr[10];        // Array of size = 10
int n = 10;
// Taking values from user
For (int i=0; i<n; i++) {
    cin >> arr[i];
}
// Double-Up the array
For (int i=0; i<n; i++){
    arr[i] = arr[i] * 2;
}
```

```
// Printing the doubled-Up array
for( int i=0; i<n; i++){
    cout << arr[i] << " ";
}
```

Output :-

1 2 3 4 5 6 7 8 9 10

2 4 6 8 10 12 14 16 18 20

Q2. Create an array of size = 5 the take values in input and print the total sum.

```
// Array of size = 5
int arr[5];
// Take input
int n=5;
cout<<" Enter the input :" <<endl;
for (int i=0; i<n; i++) {
    cin >> arr[i];
}
// Calculate sum
int sum =0;
for( int i=0; i<n; i++){
    sum = sum + arr[i];
}
// Printing total sum
cout<<" Total sum : "<<sum;
```

Output :-

Enter the input :

2 4 6 8 10

Total sum : 30

# Linear Search in Array :-

* In linear search, we traverse the array from starting to end or vice-versa to perform specific task.

Q1. Find the value of target in an array.

```
int arr[5] = { 5, 6, 7, 8, 9};
int n=5;
int target = 8;
bool flag = 0;
// 0 ⟶ Not found
// 1 ⟶ Found

for(int i=0; i<n; i++) {
  if (target == arr[i]) {
      flag = 1;
      break;
  }
}
// Checking the value of flag
if (flag == 1) {
  cout<< " Target found";
}
else {
  cout << " Target Not found";
}
```

Output:-

Target Found

Explanation:-

- We got array and target in input, to find the target in array we used a boolean variable named 'flag'.
- Flag = 0 means target not found and flag = 1 means target found.
- for loop is checking every value of array with the target value, if they got matched then flag becomes 1 and loop will breaks.
- finally we are checking the value of flag.

# Arrays & functions :-

↳ While passing arrays to a function, keep in mind to pass the size of array.

↳ Size here means the total no. of values present in array.

```
int main()
{
    int arr[5];
    int size = 5;
    solve(arr, size);
}
```

function call

```
solve(int arr[], int size)
{
    //logic
}
```

∴ Here, we passed a 1-dimensional array and integer size as arguments in function named 'solve()'.

Ex:-

function which print values of array :-

```
void printArray(int arr[], int size) {
    for(int i=0; i<size; i++) {
        cout << arr[i] << " ";
    }
    cout << endl;
}
int main() {
    int arr[5] = {2, 4, 6, 8, 10};
    int size = 5;
    // Function call
    printArray(arr, size);
}
```

Output :-

2 4 6 8 10

Page No.
Date

Q1. Finding the value of target present in an array using function which takes array in argument.

```
bool linearSearch (int arr[], int size, int target){

    for (int i=0; i<size; i++) {
        if (arr[i] == target) {
            // Target found
            return true;
        }
    }
    // Not found
    return false;
}

int main () {
 int arr[5] = { 5, 4, 2, 6, 1 };
 int size = 5;
 int target = 2;
 // function call
 bool ans = linearSearch (arr, size, target );

 if(ans == 1){
    cout << "Target found";
 }
 else {
    cout<<"Target Not found";
 }
}
```

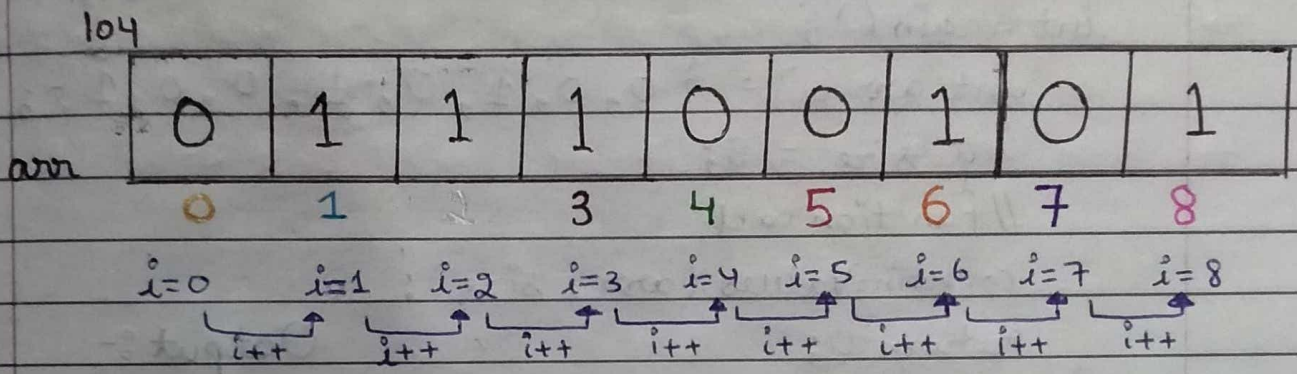
Output:-
Target Found

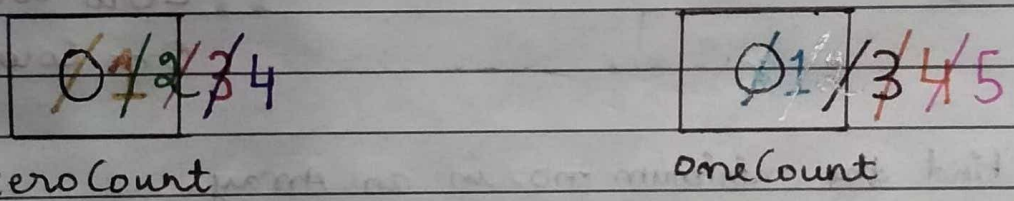
↳ Finding the target value without using flag.

Page No.
Date

Q2. Count 0's and 1's in an Array.

To solve this, we create two variables which count no. of 0's and 1's seperately. for complete array traversal we use loop which check array values and compare them with '0' and '1'. when a value matched with them, it will increment its corresponding count by 1. Both variables are initialised with 0.




∴ finally zeroCount = 4 (which means 4 zeros in array i.e True)

And oneCount = 5 (which means 5 ones in array i.e True)

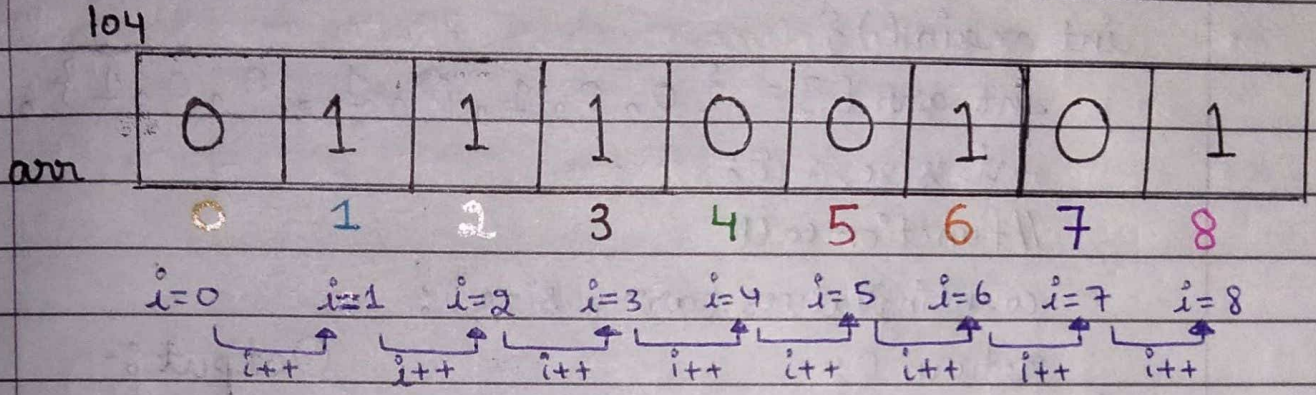
Code :-

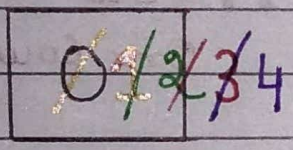
```
void countingValues(int arr[], int n){
    int zeroCount = 0;
    int oneCount = 0;
    // loop to traverse array
    for(int i=0; i<n; i++){
        if(arr[i] == 0){
            zeroCount++;
```

Page No.
Date

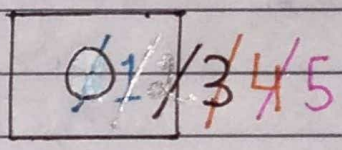
Q2. Count 0's and 1's in an Array.

To solve this, we create two variables which count no. of 0's and 1's seperately. for complete array traversal we use loop which check array values and compare them with '0' and '1'. when a value matched with them, it will increment its corresponding count by 1. Both variables are initialised with 0.




zeroCount




oneCount

∴ finally zeroCount = 4 (which means 4 zeros in array i.e True)

And oneCount = 5 (which means 5 ones in array i.e True)

Code :-

```
void countingValues(int arr[], int n){
    int zeroCount = 0;
    int oneCount = 0;
    // loop to traverse array
    for(int i=0; i<n; i++){
        if(arr[i] == 0){
            zeroCount++;
```

```
        }
        if (arr[i] == 1) {
            oneCount++;
        }
    }
    cout << "Zero Count = " << zeroCount << endl;
    cout << "One Count = " << oneCount << endl;
}

int main(){
    int arr[] = {0,0,1,0,1,0,0,1};
    int size = 8;
    //function call
    countingValues(arr, size);
    return 0;
}
```

Output :-
Zero Count = 5
One Count = 3

Q3. Find the minimum no. in an Array.

→ Range of signed integer is from $-2^{31}$ to $2^{31}-1$

→ We have 2 utility values in "limits.h" header file named INT_MIN and INT_MAX.

→ So, INT_MIN represents :- $-2^{31} = -2147483648$
and INT_MAX represents :- $2^{31}-1 = 2147483647$

→ Best Practice :-
  └→ To find min no., make a variable to store the minimum value & initialise it with → INT_MAX.
∴ When we compare any value with this, always that value is smaller and keep comparing it till the end of array.

↳ Similarly, always initialise the variable which is going to store the maximum answer with → INT_MIN.
∴ while comparing array values with this 'int min', it will always gives the greater value and thats how we can retrieve the maximum higher value in an array.

Examples:-

```
int findMinimumInArray (int arr[], int size) {
    // Variable to store minimum answer
    int minAns = INT_MAX;

    // Loop, to traverse the array
    for (int i=0; i<size; i++) {
        if (arr[i] < minAns) {
            minAns = arr[i];
        }
    }
    return minAns;
}

int main() {
    int arr[] = {10, 8, 4, 7, 2, -3, 1};
    int size = 7;

    int minimum = findMinimumInArray (arr, size);
    cout << " Minimum number is: " << minimum << endl;
    return 0;
}
```

Output:-

Minimum number is: -3

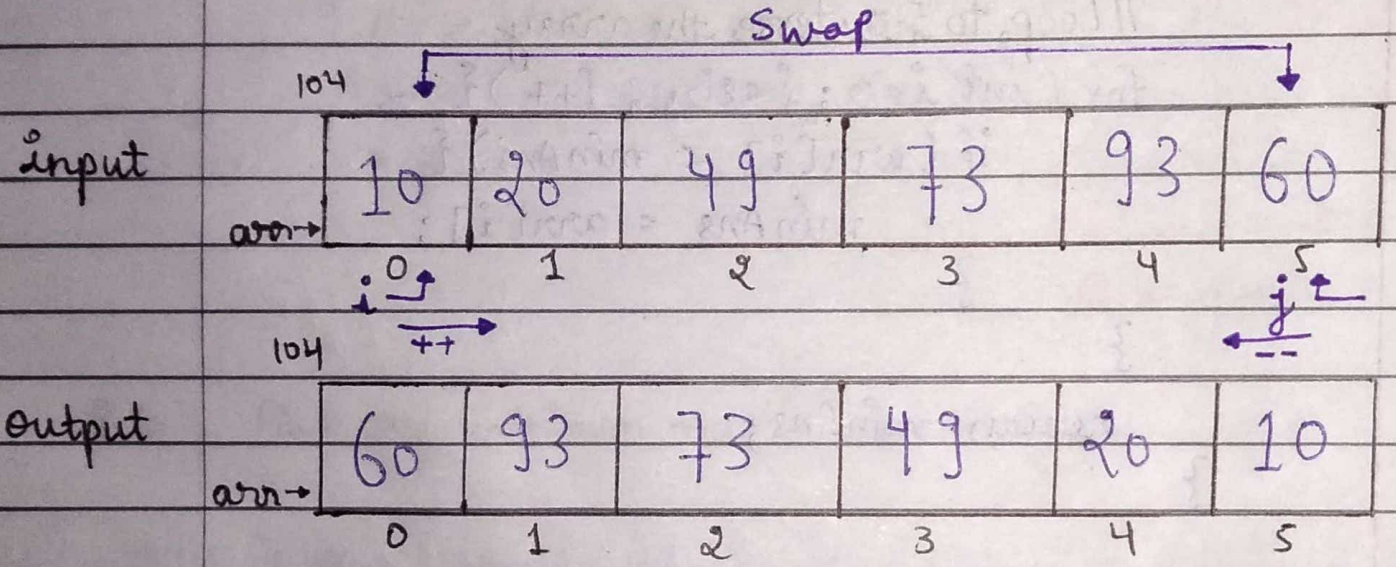
Q4. Reverse the array given in input.

→ swap() is a utility function in C++, which takes two values as input and swap them with eachother.

→ Take two iterators, one is initialised with '0' and traverse array from left to right and initialise other iterator with 'size-1' and traverse array from right to left.

→ Compare both values of each iterator and swap them with each other.

→ Both iterators will iterate until left <= right.




Example:-

```
void reverseArray (int arr[], int size){
    int left = 0;
    int right = size-1;
    while (left <= right){
        swap(arr[left], arr[right]);
        left++;
        right--;
    }
```

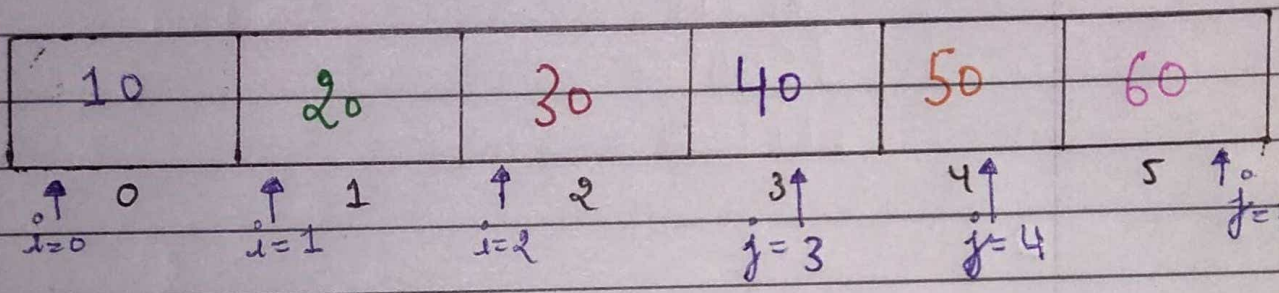
```
// Printing reversed array
for(int i=0; i<size; i++) {
    cout << arr[i] << " ";
}
}
int main(){
    int arr[] = { 10, 20, 49, 73, 93, 60 };
    int size = 6;
    reverseArray(arr, size);
    return 0;
}
```

Output:-

60 93 73 49 20 10

Qs. Extrem print in an Array.

input →




output → 10 60 20 50 30 40

Code:-

```
void extremPrint(int arr[], int size){
    int left = 0;
    int right = size-1;
    while( left <= right ){
        if(arr[left] == arr[right]){
// when both iterators are at same value, in odd no. of
// values we print that central value only at once.
            cout << arr[left];
        }
        else{
        cout << arr[left] << "_" << arr[right] << "_";
        }
```

```
            i++;
            j--;
        }
    }
int main() {
    int arr[] = {10, 20, 49, 73, 93, 60};
    int size = 6;
    // function call
    extremePrint(arr, size);
    return 0;
}
```

Output:-

10 60 20 93 49 73

# ARRAYS [LEVEL-2]

Page No. _____
Date 08 09 23

**Pass By Value :-** Whenever a variable is passed into a function by value, it means copy of that passed variable is created in the memory for that particular function.

Ex:-

```
int main(){
    int a=5;
    solve(a);
    cout<<a<<endl;
}
```

```
void solve(int a){
    a++;
    cout<<a<<endl;
}
```

104 a [5]

328 a [56]

**Output :-**

6
5

↳ Inside main, variable named 'a' is created and after passing 'a' into 'solve' function. Copy of 'a' is created. Here, both variables have same name 'a', but they both are different memory allocations. Thats why output is different bcz scope of 'a' present in main function is limid to main only and 'a' present in solve() function have its scope only inside solve().

Ex:-

```
int main(){
    int mark=90;
    mark++;
    solve(mark);
    cout<<mark<<endl;
    return 0;
}
```

```
void solve(int m){
    m--;
    m=m*10;
    cout<<m;
}
```

104 mark [90] [91]

423 m [91] [90]

**Output :-**

900 → (m=m×10 ⇒ 90×10 = 900)
91

Page No.
Date

↳ Here, copy of mark is created inside solve() with variable name 'm'. Both the variables have different memory blocks and have different addresses.

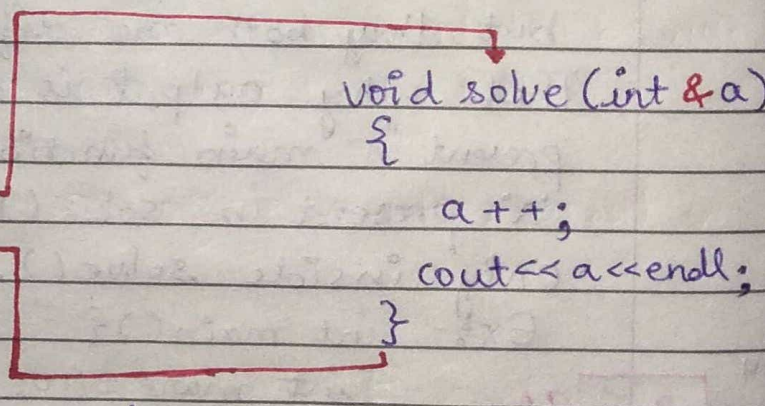
# Pass By Reference :-

→ When a variable is passed by reference, it means we are sending the original variable into the function and further updation of that variable into that function will also change the value of that variable into the memory block of that variable.

→ In short, further updation of passed variable will gets reflected into its original memory block too.

→ To pass any variable by reference, we write an ampersand '&' sign before the variable name while passing the variable in argument list of particular function.

Ex:-
```
int main(){
    int a=5;
    solve(a);
    cout<<a;
    return 0;
}

void solve(int &a)
{
    a++;
    cout<<a<<endl;
}
```




Output :-

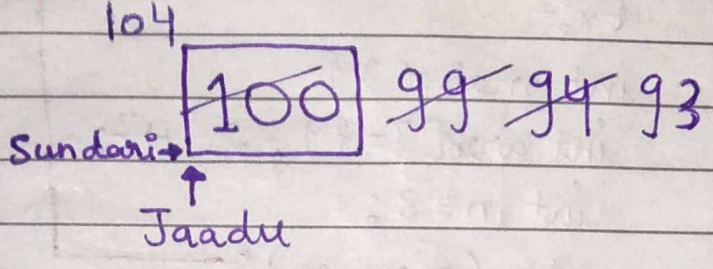
6 → printed by cout written in solve

6 → printed by cout written in main

Page No.
Date

Ex:- 
```
int main(){
    int sundari = 100;
    sundari--;
    sundari -= 5;
    solve(sundari);
    cout << sundari;
    return 0;
}
```

```
void solve(int & Jaadu)
{
    Jaadu--;
    cout << Jaddu + 10;
    return;
}
```




Output:-
103
9-

# Reference Variable:-

↳ In the above example, variable name 'sundari' and 'Jaadu' both are pointing to same memory block.

↳ Compiler will create a symbol table where both of them are mapped to a same memory address.

Ex:-
```
int a = 5;
int &b = a;
```

∴ let address of a = 1743@2

∴ We can say that, same memory block

| Variable Name | Memory Address |
|---|---|
| Sundari → | 104 |
| Jaadu → | 104 |
| a → | 1743@2 |
| b → | 1743@2 |

Symbol Table

# Passing arrays in a function :-

↳ **Keep in Mind** that, array is by default passed by reference. That means original array is passed as an argument every time and any change will get reftect back in its continuous memory blocks.

Ex:-

```
int main(){
 int arr[]={1, 2, 4};
 int n=3;
 solve(arr, n);
 for(int i=0; i<n; i++)
 {
   cout<<arr[i];
 }
 return 0;
}
```

```
solve(int arr[], int size)
 {
  arr[0]=100;
 }
```

Output

100 2 4

↳ **Remember that** :- While passing array as argument, we have to pass its size as an argument too because we don't have any explicit way to find its size.

↳ Here, size means no. of elements present in that array.

Q1. Find unique element in an array. Each element occurs twice except one.

Truth Table of XOR →

| a | | b | | output |
|---|---|---|---|---|
| 0 | ↔ | 0 | → | 0 |
| 0 | ↔ | 1 | → | 1 |
| 1 | ↔ | 0 | → | 1 |
| 1 | ↔ | 1 | → | 0 |

∴ OBSERVE - $0 \wedge a \rightarrow a$

XOR with '0' gives us the other element

```
int findUnique(int arr[], int size){
    int ans = 0;
    for(int i=0; i<size; i++){
        ans = ans ^ arr[i];
    }
    return ans;
}
int main(){
    int arr[] = {1,4,4,1,2,5,6,5,6};
    int size = 9;
    int finalAns = findUnique(arr, size);
    cout << "final ans = " << finalAns;
    return 0;
}
```

Output :-

final ans = 2

Explanation :-

Here, we are storing ans ^ arr[i] in ans variable i.e. after completing the loop ans will take XOR of every array element and calculate its XOR. Same values will be cancelled out and when the remaining value is going to calculate XOR with '0' then we get that unique element.

ans = 0 ^ 1 ^ 4 ^ 4 ^ 1 ^ 2 ^ 5 ^ 6 ^ 5 ^ 6

ans = 0 ^ 2 (same value will get cancelled or we can say, it will give '0' bcz XOR with same value will gives 0 in return)

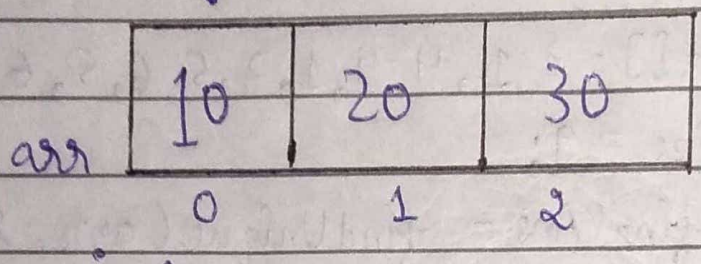
ans = 2

Page No.
Date

Q2. Print all pairs of an array.

Input Array ⟶ {10, 20, 30}

Output ⟶ (10,10), (20,10), (30,10)
(10,20), (20,20), (30,20)
(10,30), (20,30), (30,30)




for i=0, j=0 → print arr[i], arr[j] → (10,10)
j=1 → (10,20)
j=2 → (10,30)

i=1, j=0 → (20,10)
j=1 → (20,20)
j=2 → (20,30)

i=2, j=0 → (30,10)
j=1 → (30,20)
j=2 → (30,30)

∵ for every value of "i" we have to travers the complete array every time.

Code:-

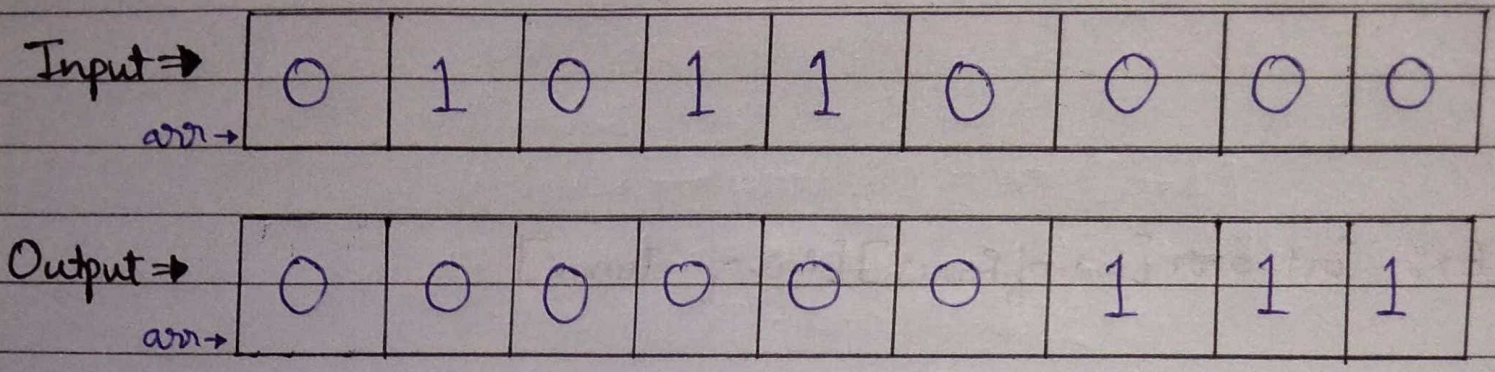
```
void printPairs(int arr[], int size){
    for (int i=0; i<size; i++){
        for (int j=0; j<size; j++){
            cout << arr[i] << ", " << arr[j] << endl;
        }
        cout << endl;
    }
}
int main(){
    int arr[] = {10, 20, 30};
    int size = 3;
    //function call
    printPairs(arr, size);
```

Output:-

```
10, 10
10, 20
10, 30

20, 10
20, 20
20, 30

30, 10
30, 20
30, 30
```

Date ..../..../...

Q30. Sort 0's and 1's.




Approaches → ① Counting
→ ② 2 pointer approach → H/W
→ ③ sort() → covered in further lectures

① Counting :- Logic :- (i) Count 0 and 1
(ii) place 0
(iii) place 1

Code :-

```
void sortZeroOne(int arr[], int size){
    int zeroCount = 0;
    int oneCount = 0;
// Step A: Count zeros and ones
    for(int i=0; i<size; i++){
        if(arr[i] == 1){
            oneCount++;
        }
        if(arr[i] == 0){
            zeroCount++;
        }
    }
// place 0 and ones in array
    int i;
```

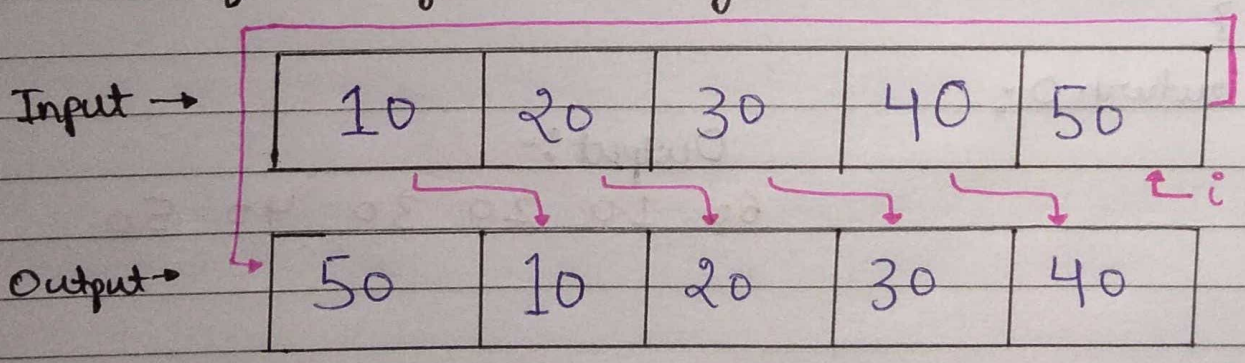
Date ..../..../.....

```
for(i=0; i<zeroCount; i++){
    arr[i]=0;
}

for(int j=i; j<size; j++){
    arr[j]=1;
}
}
int main(){
    int array[]={0,1,1,0,0,1,1,0,0};
    int n=9;
    // function call
    sortZeroOne(array, n);
    // Printing the array
    for(int i=0; i<n; i++){
        cout<<array[i]<<" ";
    }
    return 0;
}
```

Output :-

0 0 0 0 0 1 1 1 1

Q4. Shift arrays element by 1.




Logic :-

(A) Store last array value in a temp variable.
(B) Iterate from i=n-1 to i=1 & store value of 'i-1' in 'i'.
(C) Store value of temp in arr[0] i.e first place.

Spiral

Code:-

```
void shiftArray(int arr[], int n){
    // step 1
    int temp = arr[n-1];
    //step2
    // shift → arr[i] = arr[i-1]
    for(int i = n-1; i>=1; i--){
        arr[i] = arr[i-1];
    }
    // Step3 → copy temp into 0th index
    arr[0] = temp;
}

int main(){
    int arr[] = {10, 20, 30, 40, 50, 60};
    int size = 6;
    // function call
    shiftArray(arr, size);

    // printing the array
    for(int i=0; i<size; i++){
        cout << arr[i] << " ";
    }
    return 0;
}
```

Output:-

60 10 20 30 40 50

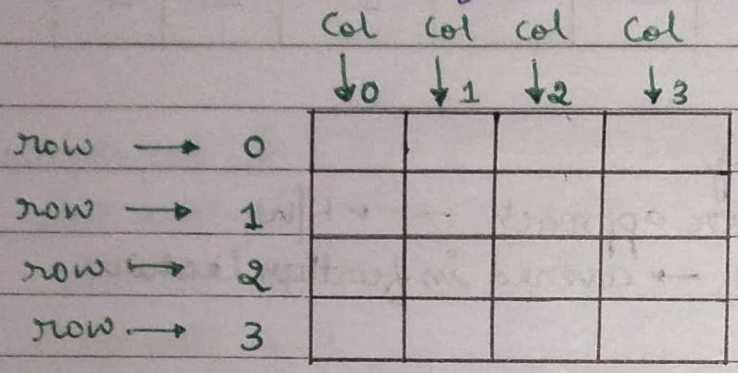
# ARRAY [Level-3]

Date 11/09/23

# 2-D-Array :- 2D-Structure which contain rows and columns its similar to matrix.

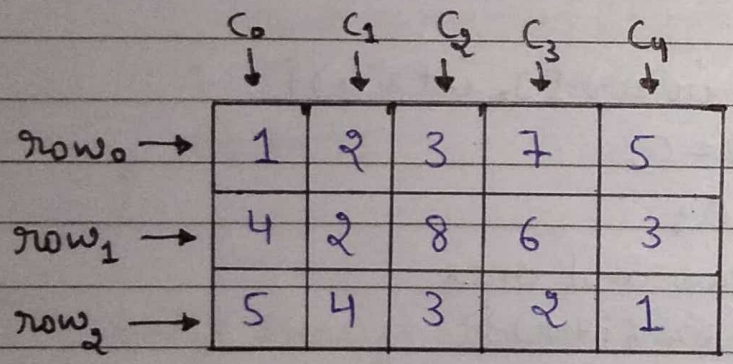
# Creation of array :-

Ex :- int arr [No. of Rows] [No. of Columns]




# Initialisation :-

Ex :- 3 rows, 5 cols → int arr[3][5] = {
{1, 2, 3, 7, 5},
{4, 2, 8, 6, 3},
{5, 4, 3, 2, 1}
}




☆ Specify no. of colums for initialising 2-D array

# Access array values in 2-D Array :-

Syntax = array_name [Row_Index] [Column_Index]

Ex :-

| | 0 | 1 | 2 |
|---|---|---|---|
| 0 | 10 | 20 | 30 |
| 1 | 40 | 50 | 60 |
| 2 | 70 | 80 | 90 |

arr

arr[0][0] = 10, arr[0][1] = 20, arr[0][2] = 30

arr[1][0] = 40, arr[1][1] = 50, arr[1][2] = 60

arr[2][0] = 70, arr[2][1] = 80, arr[2][2] = 90

Spiral

Date ..../..../.....

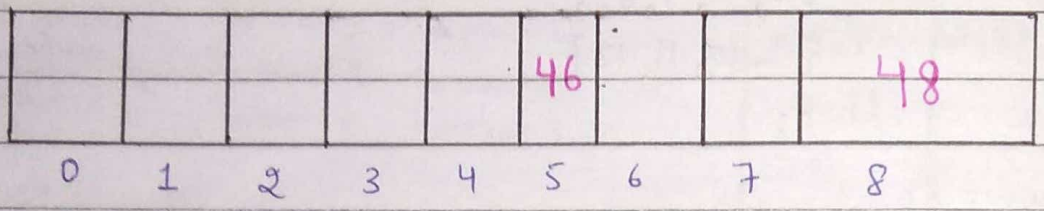
# Storage of 2-D array in memory :-

We are visualising 2-D array as matrix but its actually stored as a linear array in the memory.

Ex:- int arr[3][3]

arr →

| | 0 | 1 | 2 |
|---|---|---|---|
| 0 | 96 | 25 | 100 |
| 1 | 67 | 52 | 46 |
| 2 | 44 | 60 | 48 |




↳ formula is used to convert 2-D array into linear array :-

$C \times i + j$

- $C$ : Total no. of columns
- $i$ : Current Row no. of that element
- $j$ : Current column no. of that element

Ex:-

arr[2][2] is at $(C \times i + j)^{th}$ index of linear array

i.e. $3 \times 2 + 2$

$= 8^{th}$ index

arr[1][2] is at $(C \times i + j)^{th}$ index

i.e $3 \times 1 + 2$

$= 5^{th}$ index

↳ Keep in mind :- (i) 2-D array initialise karte time atleast column size specify krna mandatory hai.

(ii) 2-D array ko function mein pass krte time bhi same condition follow hogi i.e. atleast column size specifically batana mandatory hai.

Date ..../..../.....

Q Print 2-D array.

```
int main(){
 int arr[3][4] = {
                  {1, 2, 3, 4},
                  {5, 6, 7, 8},
                  {9, 10, 11, 12}.
                  };
  int row = 3;
  int col = 4;
  // function call
  printArray (arr, row, col);
  return 0;
}

void printArray ( int arr[ ][4], int row, int col){
    for ( int i=0; i<row; i++ ){
      for ( int j=0; j<col; j++ ) {

          cout << arr[i][j] << " ";
      }

      cout << endl;
    }
}
```

Output

1 2 3 4
5 6 7 8
9 10 11 12

∴ Row wise access

Date ..../..../.....

→ Column wise access

for each column, every row is printing.

Col = 0
- ↳ row = 0
- ↳ row = 1
- ↳ row = 2
- ↳ row = 3

Col = 1
- ↳ row = 0
- ↳ row = 1
- ↳ row = 2
- ↳ row = 3

Col = 2
- ↳ row = 0
- ↳ row = 1
- ↳ row = 2
- ↳ row = 3

| | 0 | 1 | 2 |
|---|---|---|---|
| 0 | (0,0) 1 | (0,1) 2 | (0,2) 3 |
| 1 | (1,0) 10 | (1,1) 20 | (1,2) 30 |
| 2 | (2,0) 11 | (2,1) 21 | (2,2) 13 |
| 3 | (3,0) 20 | (3,1) 22 | (3,2) 53 |

Outer loop → i=0 to i<col

↳ Inner loop → j=0 to j< row

Printing → arr[col][row]

**Code**

```
void colWisePrint(int arr[][3], int row, int col){
    for(int i=0; i<col; i++){
        for(int j=0; j<row; j++){
            cout<< arr[j][i]<<" ";
        }
        cout<<endl;
    }
int main(){
    int arr[4][3] = {
                        {1,2,3},
                        {10,20,30},
                        {11,21,13},
                        {20,22,53}
                    };
    int row=4; int col=3;
    colWisePrint(arr, row, col);
    return 0;
}
```

Output :-

```
1 10 11 20
2 20 21 22
3 30 13 53
```

Spiral

Date ..../..../.....

Q Search the value of target in given array and return true if target present and return false if target is absent.

target = 70

| | 0 | 1 | 2 |
|---|---|---|---|
| 0 | 10 | 20 | 30 |
| 1 | 40 | 50 | 60 |
| 2 | 70 | 80 | 90 |

Code :-

```
bool findTarget (int arr[][3], int row,
    int col, int target ){

  for (int i=0; i< row; i++){
      for (int j=0; j< col; j++){
          if (arr[i][j] == target) {
              return true;
          }
      }
  }
  // Iss line par tabhi aa sakte ho, jab saare element check
  // ho chuke or target nhi mila hoga
  // return krdo false;
  return false;
}
int main (){
  int arr[][3] = { {10, 20, 30},
                   {40, 50, 60},
                   {70, 80, 90}
                 };
  int row = 3;
  int col = 3;
  int target = 70;
  // Function Call
  Cout << " found or Not : " << findTarget (arr, row, col, target);
  return 0;
}
```

Output :-

Found or Not : 1

Spiral

Date ..../..../.....

Q find maximum no. in an array.

Initialise answer variable with INT_MIN and then compare it with all elements. If "no." > maxAns then store that "no." in maxAns. Traverse array using 2 loops

Code:-

```
int findMaxInArray(int arr[][3], int row, int col){
    int maxAns = INT_MIN; //Initialisation with INT_MIN
    for(int i=0; i<row; i++){
        for(int j=0; j<col; j++){
            if( arr[i][j] > maxAns){ //comparison
                maxAns = arr[i][j]; //updation
            }
        }
    }
    return maxAns;
}
```

Q find minimum no. in an array.

Initialise answer variable with INT_MAX and traverse the array if any element found less than answer variable then store it in answer.

Code:-

```
int findMinInArray(int arr[][3], int row, int col){
    int minAns = INT_MAX; //Initialisation with INT_MAX
    for(int i=0; i<row; i++){
        for(int j=0; j<col; j++){
            if( arr[i][j] < minAns){ //comparing
                minAns = arr[i][j] //updating it with arr[i][j]
            }
        }
    }
    return minAns;
}
```

Spiral

Date ..../..../.....

Q Find row wise sum in an array.

Traverse the 2-D array row wise and initialise sum = 0 for each iteration of outer loop. Store the sum of each element with sum itself and store it into sum variable and then print sum.

Code:-

```
void rowWiseSum(int arr[][3], int row, int col){
    for(int i=0; i<row; i++){
        int sum = 0;
        for(int j=0; j<col; j++){
            sum += arr[i][j];
        }
        cout<<"Sum of row "<<i<<" is "<<sum<<endl;
    }
}

int main(){
    int arr[][3] = {{1, 2, 3},
                    {40, 5, 60},
                    {7, 0, 2}
                   };
    int row = 3;
    int col = 3;
    rowWiseSum(arr, row, col);
    return 0;
}
```

Output:-

```
Sum of row 0 is 6
Sum of row 1 is 105
Sum of row 2 is 9
```

Q Find column wise sum in array. HW

Traverse the array column wise and initialise sum with 0. for each column add every current row element in sum variable then print the sum when all rows are traversed.

Date ..../..../.....

Code:-

```
void colWiseSum (int arr[][3], int row, int col) {
    for (int i=0; i< col; i++){
        int sum=0;
        for (int j=0; j< row ; j++) {
            sum += arr[j][i];
        }
        cout << "sum of col " << i << " is " << sum << endl;
    }
}
int main(){
    int arr[][3] = { {1, 2, 3},
                     {40, 5, 60},
                     {7, 0, 2}
                   };
    int row = 3, col = 3;
    colWiseSum (arr, row, col);
    return 0;
}
```

Output:-

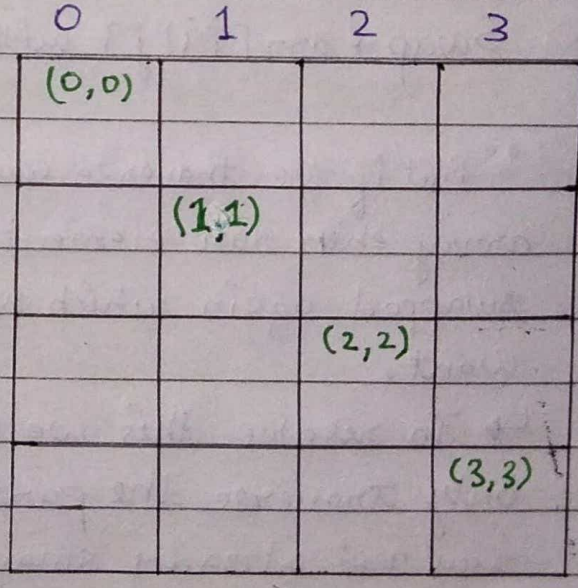
Sum of Col 0 is 48
Sum of Col 1 is 7
Sum of col 2 is 65

Q Print diagonal of 2-D array. Assuming no. of rows = no. of columns.

∵ for this diagonal we have elements whose row and column index are same.

Code:-

```
void diagonalPrint(int arr[][3], int row){
    for (int i=0; i< row; i++){
        cout << arr[i][i] << " ";
    }
}
```




Spiral

Date ..../..../.....

Q Print diagonal sum of 2-D array. Assuming no. of rows = no. of columns.

∴ Initialise a sum variable with 0.

↳ Then traverse the array and add only those elements in sum whose row and column index are the same.

| | 0 | 1 | 2 |
|---|---|---|---|
| 0 | (0,0) | | |
| 1 | | (1,1) | |
| 2 | | | (2,2) |

Code:-

```
void diagonalSum (int arr[][3], int row)
  {
      int sum = 0;
      for ( int i = 0; i<row; i++ ){
          sum += arr[i][i];
      }
      cout << sum;
  }
```

Imp

Q Print transpose of 2-D matrix. Assuming no. of rows = no. of columns.

Transpose means $i^{th}$ row becomes $i^{th}$ column.

Approach:-

Swap → arr[i][j] with arr[j][i]

∴ But if we traverse complete array then our elements gets swapped again which we don't want.

↳ To resolve this we should only traverse the part which was not already swapped

↳ Inner Loop will initialised with j==i, so that we can skip the swapped column.

| | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| 0 | (0,0) 10 | (0,1) 20 | (0,2) 30 | (0,3) 40 |
| 1 | (1,0) 5 | (1,1) 4 | (1,2) 3 | (1,3) 2 |
| 2 | (2,0) 1 | (2,1) 7 | (2,2) 19 | (2,3) 22 |
| 3 | (3,0) 28 | (3,1) 6 | (3,2) 31 | (3,3) 9 |

Spiral

Date ..../..../.....

Code:-

```
void transpose (int arr[][3], int row, int col) {
    // Outer Loop
    for(int i= 0; i< row; i++) {
    // Inner Loop → from j==i to j< col
        for(int j=i; j< col; j++){
            swap( arr[i][j], arr[j][i]);
        }
    }
}
```

# Vector :-

A vector stores elements of a given type in a linear arrangement and allow fast random access to any element.

↳ Syntax to create a vector :-

```
vector<data_type> vector_name(size);
```

Ex :-

```
vector<int> v(5);
```

↳ To check the size of vector :-

size() - It's a function which return the size of vector.

Ex :-

```
cout << v.size(); // It will print the size of 'v' vector i.e 5
```

* By default, all vector elements are assigned with '0'.

↳ To initialise vector elements :-

Ex :-

```
vector<int> arr(5, 40);
```

∴ Here, all the elements of vector 'arr' will initialise with '40'

Date ..../..../.....

↳ Insert elements into 1-D vector :-

We can use inbuilt function push_back() to push any value.

Ex:- arr.push_back(1);
arr.push_back(2);

∴ This will push/insert 1 and 2 in the vector named "arr"

# 2-D vector :-

↳ Syntax to create 2-D vector :-

```
Vector < vector < int > > arr
```

int → Datatype
space
arr → vector_name

↳ Visualisation of 1-D and 2-D vectors :-

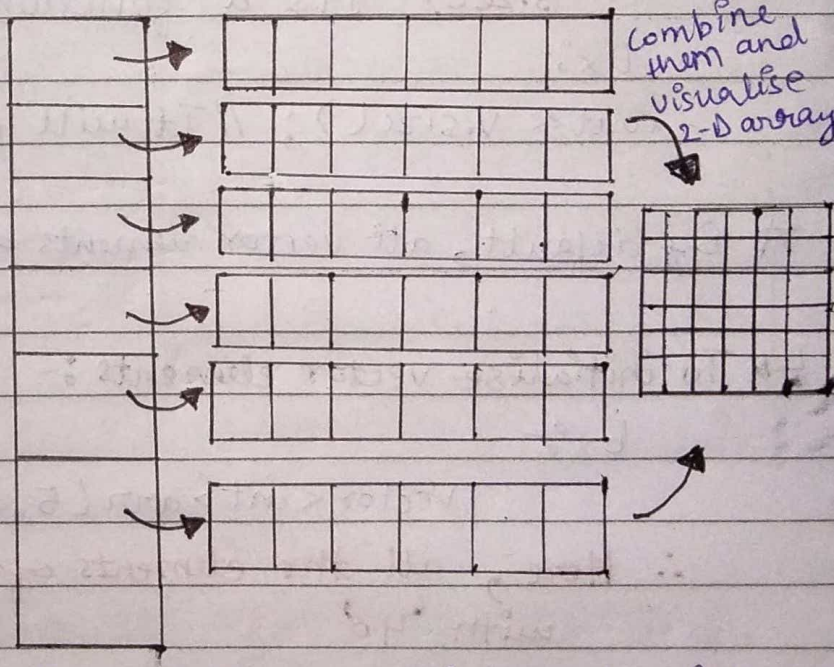
Ex:-

vector <int> v

1D | 2 | 4 | 6 | 8 | 9 | 10 |

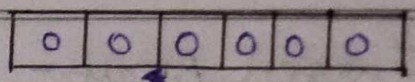
↳ 1-D array which have integer values in it.

vector <vector <int> >




∴ A 1-D array contain <vector<int> > type of elements in each place

Date ..../..../.....



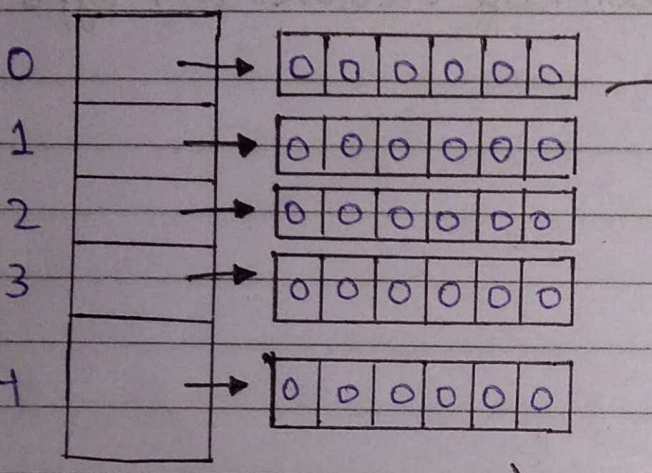
```
Vector < vector <int> > arr ( 5 , vector <int> (6,0) )
```

2-D array

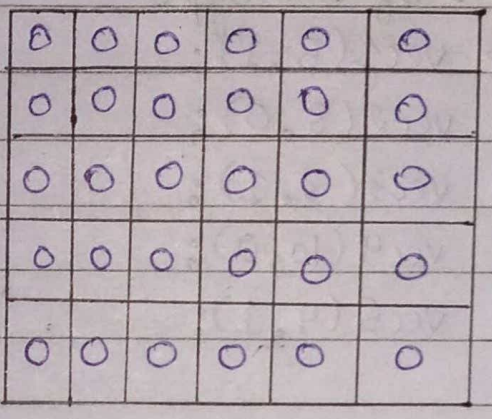
name

Row size

Row Item
↳ Initialise → with a vector of size 6 that is initialised with 0



1-D array with size '5' and initialised with vector of integer type



5

6

∵ A 2-D array with 5 rows and 6 columns which is initialised with zeros

**Code:-**

Row size ← Each Row item initialise with this vector → And this vector have size = 10 and initialise with 0.

```
vector< vector<int> > arr(5, vector<int>(10,0));

for(int i=0; i< arr.size(); i++){
    for( int j=0; j< arr[i].size(); j++){
        cout << arr[i][j] << " ";
    }
    cout<<endl;
}
```

**Output:-**

```
10 10 10 10 10 10 10 10 10 10
10 10 10 10 10 10 10 10 10 10
10 10 10 10 10 10 10 10 10 10
10 10 10 10 10 10 10 10 10 10
10 10 10 10 10 10 10 10 10 10
```

Spiral

Date ..../..../.....

# Jagged Array -

A 2-D array which has different no. of columns in its each row.

**Code:-**

```
int main(){
    vector< vector<int> > brr;
// This line above will make a 2-D vector of <vector<int>> type

// 1-D vectors of <int> type
  vector<int> vec1(5,1);
  vector<int> vec2(8,0);
  vector<int> vec3(3,1);
  vector<int> vec4(10,0);
  vector<int> vec5(4,1);

// Inserting 1-D vectors in 2-D vector
  brr.push_back(vec1);
  brr.push_back(vec2);
  brr.push_back(vec3);
  brr.push_back(vec4);
  brr.push_back(vec5);

// Printing 2-D vector
  for(int i=0; i<brr.size(); i++){
    for(int j=0; j<brr[i].size(); j++){
        cout<<brr[i][j]<<" ";
    }
    cout<<endl;
  }
return 0;
}
```

**Output:-**

```
1 1 1 1 1
0 0 0 0 0 0 0 0
1 1 1
0 0 0 0 0 0 0 0 0 0
1 1 1 1
```

Spiral

# ARRAY [EXTRA CLASS]

Date 12/09/23

Q1. We have an array with both positive and negative numbers. We have to modify the array and keep all -ve numbers in left and +ve in right side.

Approach :-

① Sorting → But takes O(nlogn) T.C

② 2-pointer approach → O(n) T.C

③ Make a temp array and first all -ve numbers in it and the store all +ve ones. → O(n) T.C
↳ Extra, O(n) space complexity.
Storage

↳ Using 2-pointer Approach

Ex:-

| | 4 | 2 | -8 | 7 | -6 | -12 | 3 |
|---|---|---|---|---|---|---|---|
| arr → | | | | | | | |

Logic :-

① 'index' is traversing the array

② 'j' represent the memory block where we can keep -ve numbers

③ arr[index] > 0 → ignore

④ arr[index] < 0 → swap (arr[index], arr[j])
then ↳ j++

Date ..../..../.....

Code :-

```
void swapNegativeOneSide (int arr[], int n){
  int j = 0;
  // j → memory block → where i can store negative numbers

  for( int i=0; i< n; i++){
  // i → looping variable for array traversal
    if( arr[i] < 0) {
        swap (arr[i] , arr[j]);
        j++;
    }
  }
}

int main(){
  int arr[] = { 4, 2, -8, 7, -6, -12, 3};
  int n = 7;
  // function call
  swapNegativeOneSide (arr, n);
  cout << "Printing the array" << endl;
  for( int i=0; i<n; i++){
    cout << arr[i] << " ";
  }
  return 0;
}
```

Output :-

Printing the array

-8 -6 -12 7 2 4 3

Date ..../..../.....

# Leetcode Questions

## 8 Sort Colors

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

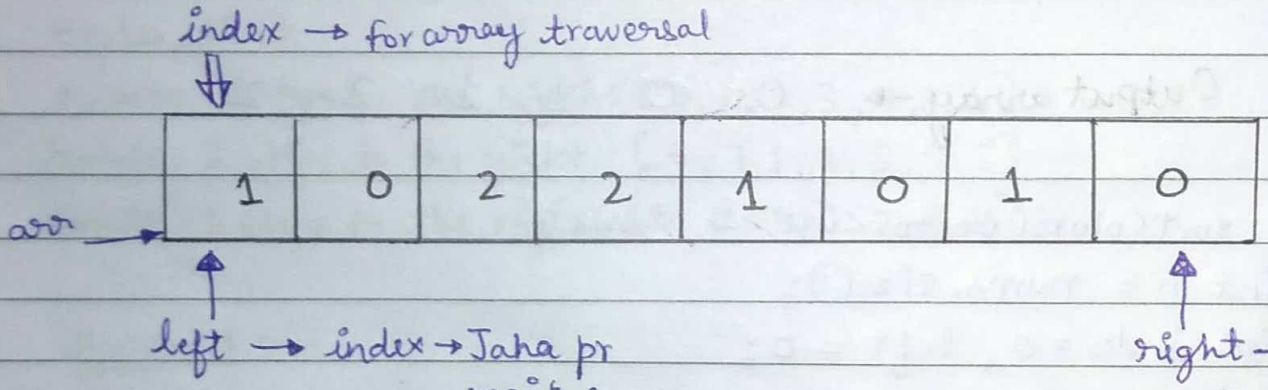
We will use the integers 0, 1, and 2 to represent the color red, white, and blue, respectively.

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

Example 1:-

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

Output: [0, 0, 1, 1, 2, 2]




Logic:-

- '0' mila to left ko dedunga
- 2 mila to right ko dedunga
- 1 mila → ignore and index++

Rule-1:- arr[index] = 1 ✗ (do nothing just → index++)

Rule-2:- arr[index] = 0 → (first) swap → (arr[index], arr[left])

After that → left++ → index++

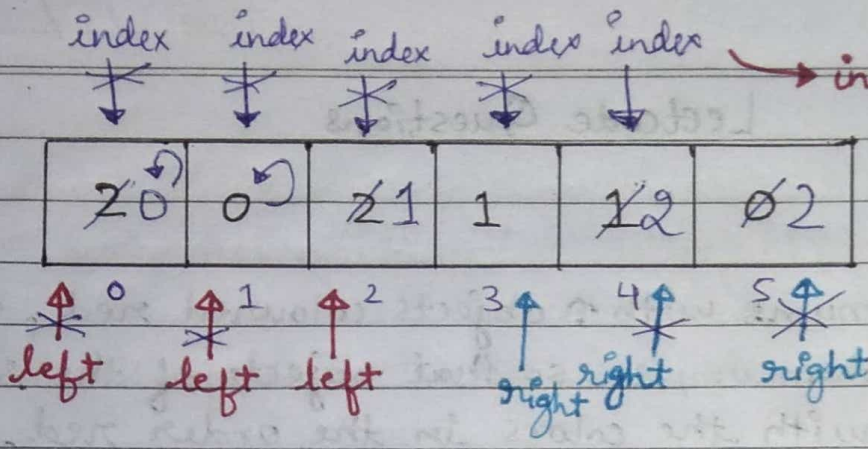
Rule-3:- arr[index] = 2 → swap → (arr[index], arr[right])

→ right-- → index++ ✗ Catch

Yaha mai galti Spiral karunga

Date ..../..../.....

↳ Dry Run :-




→ index > right
↳ stop!
Bcz now we have only 2's so, no need to swap

index=0 → arr[index] = 2 → swap(index, right) → right--
index=0 → arr[index] = 0 → swap(index, left) → left++ → index++
index=1 → arr[index] = 0 → swap(index, left) → left++ → index++
index=2 → arr[index] = 2 → swap(index, right) → right--
index=2 → arr[index] = 1 → index++
index=3 → arr[index] = 1 → index++
index=4
↳ index > right
↳ STOP!

Output array → 0 0 1 1 2 2

Code :-

```
void sortColors(vector<int>& nums){
    int n = nums.size();
    int index=0, left = 0;
    int right=n-1;

    while (index<= right){
        if(nums[index]==0){
            swap(nums[index],nums[left]);
            left++;
            index++;
        }
```

Spiral

Date ..../..../.....

```
else if (nums[index] == 2){
    swap(nums[index], nums[right]);
    right--;
    // catch → no need of index++
}
else{
    index++;
}
}
}
```

Q Rotate Array. (189) Imp ★★★ Amazon

Given an integer array 'nums', rotate the array to the right by 'k' steps, where 'k' is non-negative.

Example 1:-

Input:- nums = [1,2,3,4,5,6,7], K=3

Output:- [5,6,7,1,2,3,4]

Explanation:-

rotate 1 steps to the right: [7,1,2,3,4,5,6]

rotate 2 steps to the right: [6,7,1,2,3,4,5]

rotate 3 steps to the right: [5,6,7,1,2,3,4]

Approach:-

① Modulus

② Temp Array

| Input | 10 | 20 | 30 | 40 | 50 | 60 |
|---|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 4 | 5 |

| Output | 50 | 60 | 10 | 20 | 30 | 40 |
|---|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 4 | 5 |

K=2

n=6

∵ Taking extra space to make new vector space

| | i/p | o/p |
|---|---|---|
| index: | 0 | 2 |
| | 1 | 3 |
| | 2 | 4 |
| | 3 | 5 |
| | 4 | 0 |
| | 5 | 1 |

index + K ✗

(index + K) % n

(0+2)%6 → 2%6 → 2

(1+2)%6 → 3%6 → 3

(2+2)%6 → 4%6 → 4

(3+2)%6 → 5%6 → 5

(4+2)%6 → 6%6 → 0

Spiral

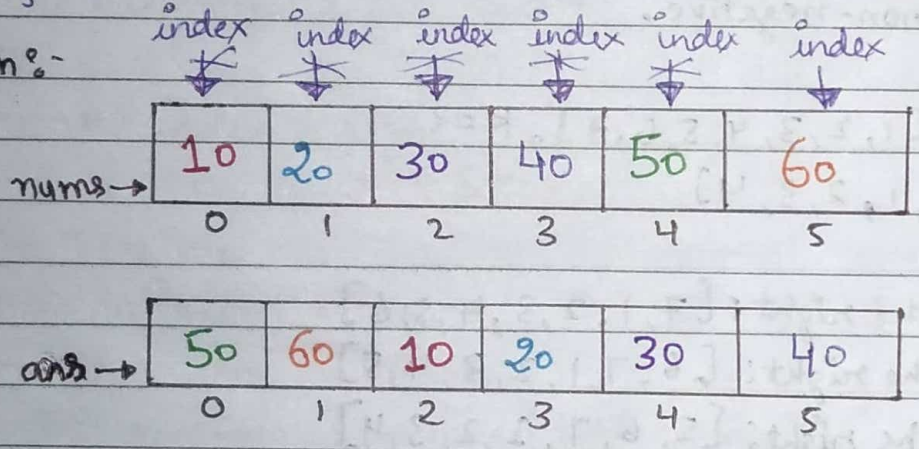
(5+2)%6 → 7%6 → 1

Date ..../..../.....

Code :-

```
Void rotate(vector<int>& nums, int K){
    int n = nums.size();
    // Taking additional space to create new shifted array
    vector<int> ans(n);

    for(int index =0; index<n; index++){
        // new index to store the shifted value
        int newIndex = (index + K) % n;
        ans[newIndex] = nums[index];
    }

    nums = ans; // bez nums mein he changes krne hai
}
```

Dry Run :-




K=2, n=6

index=0
newIndex = (0+2)%6 = 2
ans[newIndex] = nums[index]
ans[2] = nums[0]

index = 1
newIndex = (1+2)%6 = 3
ans[newIndex] = nums[index]
ans[3] = nums[1]

index = 2
newIndex = (2+2)%6 = 4
ans[newIndex] = nums[index]
ans[4] = nums[2]

index = 3
newIndex = (3+2)%6 = 5
ans[newIndex] = nums[index]
ans[5] = nums[3]

index = 4
newIndex = (4+2)%6 = 0
ans[newIndex] = nums[index]
ans[0] = nums[4]

index = 5
newIndex = (5+2)%6 = 1
ans[newIndex] = nums[index]
ans[1] = nums[5]

Spiral

Date ..../..../.....

Q Missing Number (268)

Given an array 'nums' containing 'n' distinct numbers in the range [0, n], return the only number in the range that is missing from the array.

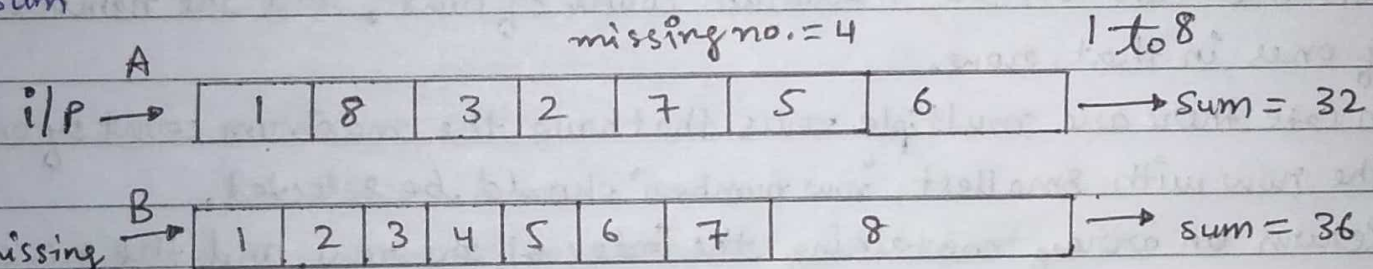
Example 1:- Input : nums = [3, 0, 1]

Output : 2

Explanation:- n = 3 since there are 3 numbers, so all numbers are in the range [0, 3]. 2 is the missing number in the range since it does not appear in nums.

Approaches:-

1. 2pointer → $O(n^2)$ T.C
2. sorting → $O(n \log n)$
   → Traverse & check diff of adjacent no.'s is 1 or not
3. Sum
4. XOR




B - A = 36 - 32

= 4 → missing no found

A.P → $sum = \frac{n}{2}(a + l)$

n → no. of terms

a → 1st term

l → Last term

Date ..../..../.....

Code:-

```
int missingNumber(vector<int>& nums){
    int n = nums.size();
    int sum = 0;
    for(int index=0; index<n; index++){
        sum += nums[index];
    }
    int totalSum = ((n)*(n+1))/2;
    int ans = totalSum - sum;
    return ans;
}
```

Q Row With Maximum Ones (2643)

Given a 'm x n' binary matrix 'mat', find the 0-indexed position of the row that contains the 'maximum' count of 'ones', and the number of ones in that row.

In case there are multiple rows that have the maximum count of ones, the row with smallest 'row number' should be selected.

Return an array containing the index of the row, and the no. of ones in it.

Ex:- Input: mat = [[0,1],[1,0]] Output:- [0,1]

Explanation:- Both rows have the same number of 1's. So we return the index of the smaller row, 0, and the maximum count of ones (1). So, the answer is [0,1].

oneCount = INT_MIN

rowNo = -1

Jab bhi new max oneCount milega, tab row no. store krayenge

| | 0 | 1 | 2 | 3 | |
|---|---|---|---|---|---|
| $r_0$ → | 1 | 0 | 0 | 0 | → 1's → 1 |
| $r_1$ → | 0 | 1 | 1 | 0 | → 1's → 2 |
| $r_2$ → | 0 | 1 | 1 | 0 | → 1's → 2 |
| $r_3$ → | 1 | 1 | 1 | 0 | → 1's → 3 → oneCount |
| $r_4$ → | 0 | 0 | 1 | 0 | → 1's → 1 |

rowNo

ans = {3, 3}

rowNo → OneCount

Spiral

Date ..../..../.....

Code :-

```
vector<int> rowAndMaximumOnes ( vector< vector<int>>& mat ){
    vector<int> ans;  // array to store answers
    int n = mat.size();
    // oneCount → will store max no. of 1's in a row
    int oneCount = INT_MIN;
    // rowNo → will store row No which contains max no. of 1's
    int row = -1 ;

    for (int i=0;  i< mat.size();  i++) {
    // har row start hone se pehle initialise count with 0
    int count =0;
        for ( int j=0;  j< mat[i].size();  j++ ){
            // If one found, the increment count
            if ( mat[i][j] == 1){
                    count ++;
            }
        }
    // After row completion, compare count with oneCount
    if ( count > oneCount) {
            oneCount = count;
            rowNo = i;
     }
}

ans.push_back( rowNo);
ans. push_back(oneCount);
return 0;
}
```

Date ..../..../.....

## Q Rotate Image (48) (Imp) ★★★

You are given an 'n x n' 2D 'matrix' representing an image, rotate the image by 90° (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 1:-

| | | |
|---|---|---|
| 1 | 2 | 3 |
| 4 | 5 | 6 |
| 7 | 8 | 9 |

⇨

| | | |
|---|---|---|
| 7 | 4 | 1 |
| 8 | 5 | 2 |
| 9 | 6 | 3 |

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

2 Step Process to get 90° rotated matrix

↳ 1. Transpose
2. Revers the rows

| | | |
|---|---|---|
| 1 | 2 | 3 |
| 4 | 5 | 6 |
| 7 | 8 | 9 |

→ Transpose

| | | |
|---|---|---|
| 1 | 4 | 7 |
| 2 | 5 | 8 |
| 3 | 6 | 9 |

↔

| | | |
|---|---|---|
| 7 | 4 | 1 |
| 8 | 5 | 2 |
| 9 | 6 | 3 |

Reverse

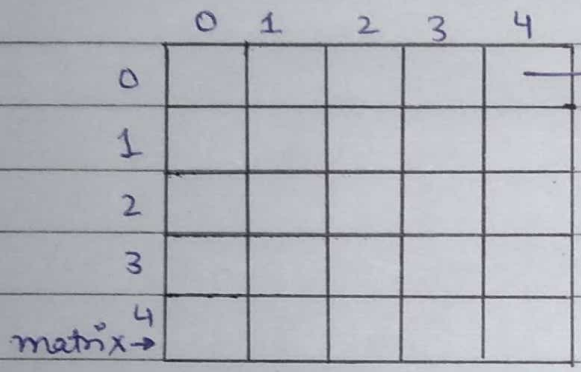
Code:-

```
void rotate(vector<vector<int>>& matrix){
    int n = matrix.size();
    // transpose
    for(int i=0; i<n; i++){
        for(int j=i; j<matrix[i].size(); j++){
            swap(matrix[i][j], matrix[j][i]);
        }
    }
```

Spiral

Date ..../..../.....

```
// reverse → 2D matrix ki sari rows ko
// kitni rows hai → 0 to n-1
for (int i = n-1; i<=0; i--){
    // har row ko reverse krna hai
    reverse (matrix[i].begin(), matrix[i].end());
  }
}
```



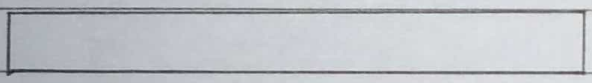

rowName → matrix[0]

→ matrix[1]

$i^{th}$ row → matrix[i]

```
for( i=0 ⟶ i<n)
{
    reverse ⟶ matrix[i]
}
```

↘ 1D Vector<int> arr



reverse (arr.begin(), arr.end())

⟶ reverse (matrix[i].begin(), matrix[i].end())