HOMEWORK DAY: 10

13 September 2023 17:21

HW CLASS: 10 12/09/2023

https://www.linkedin.com/in/manojoffcialmj/

1. Re-arrange array elements (Leetcode-2149)

Input

nums

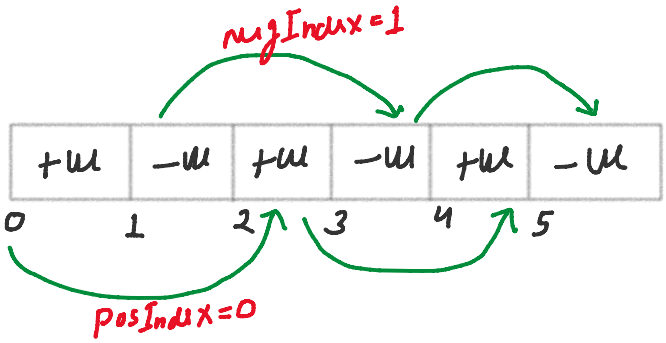
0	1	2	3	4	5
3	1	-2	-5	2	-4

size = 6

Output

0	1	2	3	4	5
3	-2	1	-5	2	-4

Observation



Note

① Array is size always even

② Negative No's == positive No's

③ pairs of number always start from positive No.

DryRun

iteration:0

nums

0	1	2	3	4	5
3	1	-2	-5	2	-4

index = 0

posIndex = 0

negIndex = 1

① nums[index] > 0

3 > 0

ans[posIndex] = nums[index];

ans[0] = 3

posIndex = posIndex + 2;

= 0 + 2

= 2

ans

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

iteration:1

nums

0	1	2	3	4	5
3	1	-2	-5	2	-4

index = 1

posIndex = 2

negIndex = 1

① nums[index] > 0

1 > 0

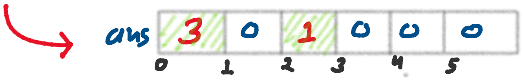
ans[posIndex] = nums[index];

ans[2] = 1

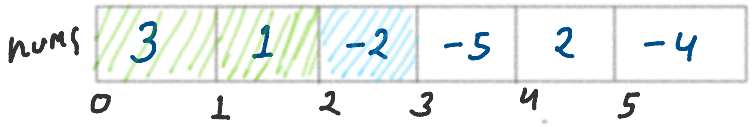
posIndex = posIndex + 2;

= 2 + 2

= 4



Iteration: 2



index = 2

posIndex = 4

negIndex = 1

① nums[index] < 0

-2 < 0

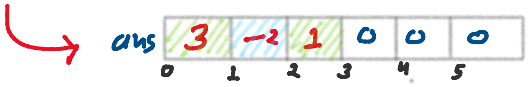
ans[negIndex] = nums[index];

ans[1] = -2

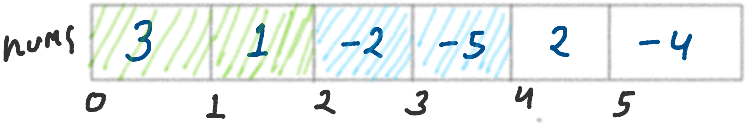
negIndex = negIndex + 2;

= 1 + 2

= 3



Iteration: 3



index = 3

posIndex = 4

negIndex = 3

① nums[index] < 0

-5 < 0

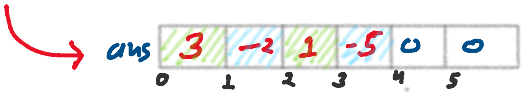
ans[negIndex] = nums[index];

ans[3] = -5

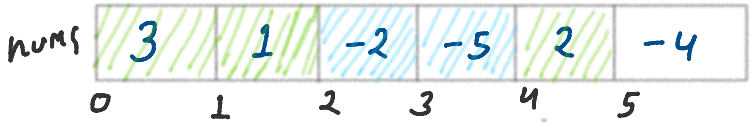
negIndex = negIndex + 2;

= 3 + 2

= 5



Iteration: 4



index = 4

posIndex = 4

negIndex = 5

① nums[index] > 0

2 > 0

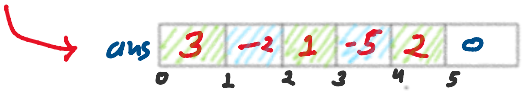
ans[posIndex] = nums[index];

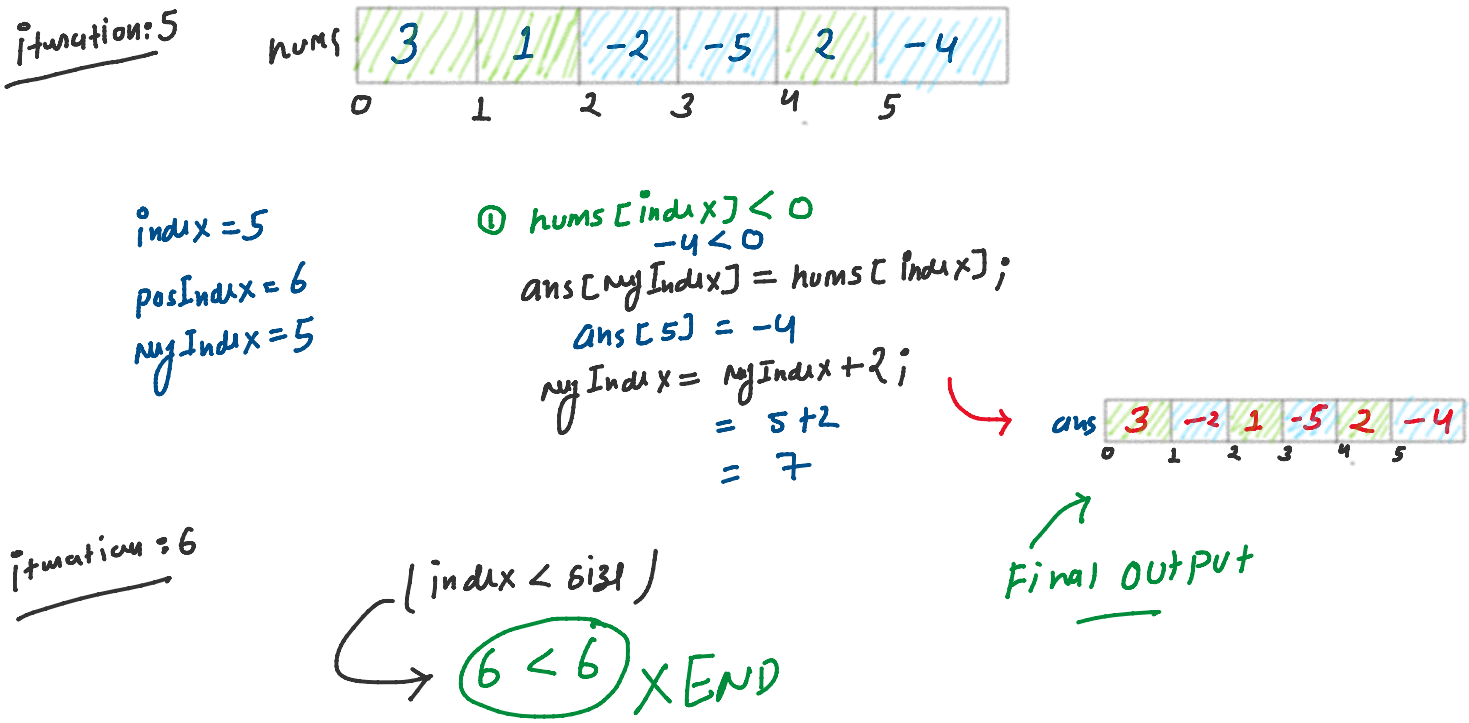
ans[4] = 2

posIndex = posIndex + 2;

= 4 + 2

= 6






```cpp
// HW 00: Re-arrange array elements (Leetcode-2149)

class Solution {
public:
    vector<int> rearrangeArray(vector<int>& nums) {
        // Size of nums
        int size=nums.size();

        // Create new vector array
        vector<int> ans(size,0);

        // Create positive index and negative index
        int posIndex=0;
        int negIndex=1;

        // Iterate the entire array
        for(int index=0;index<size;index++){
            // When element is positive then use even index in ans
            if(nums[index]>0){
                ans[posIndex]=nums[index];
                posIndex+=2;
            }
            // When element is positive then use even index in ans
            else{
                ans[negIndex]=nums[index];
                negIndex+=2;
            }
        }

        // Return new array
        return ans;

    }
};
```

T.C. => O(N)

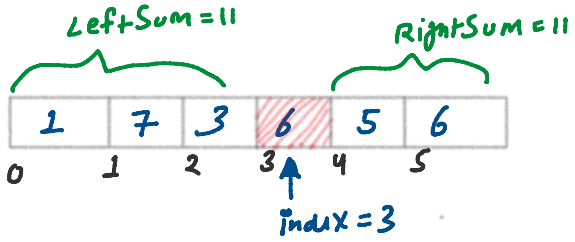
2. Find pivot element (Leetcode-724)

Example 1:
Input: nums = [1,7,3,6,5,6]
Output: 3
Explanation:
The pivot index is 3.
Left sum = nums[0] + nums[1] + nums[2] = 1 + 7 + 3 = 11
Right sum = nums[4] + nums[5] = 5 + 6 = 11

Example 2:
Input: nums = [1,2,3]
Output: -1
Explanation:
There is no index that satisfies the conditions in the problem statement.

DRY RUN

Example 1:



APPROACH PSEUDO CODE:
Step 01: find total sum of array as right sum
Step 02: subtract element one by one from right sum until left sum is equal to right sum
Step 03: return the index which terminate the loop

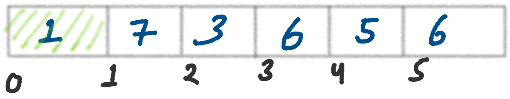
(1)

TotalSum = 1+7+3+6+5+6

RightSum = 28

(2)

Iteration: 0



RightSum = 28
LeftSum = 0
index = 0

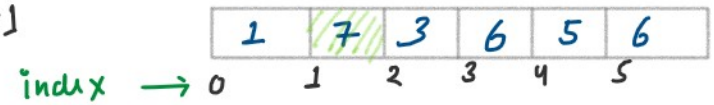
↳ RightSum = RightSum − nums[index]
= 28 − 1
= 27

↳ (RightSum == LeftSum)
27 == 0 ✗ FALSE

↳ LeftSum = LeftSum + nums[index]
= 0 + 1

= 1

Situation: 1



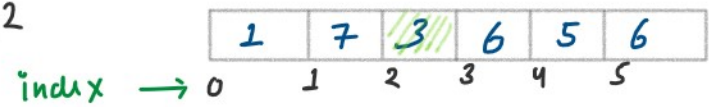
RightSum = 27
LeftSum = 1
index = 1

↳ RightSum = RightSum − nums[index]
= 27 − 7
= 20

↳ (RightSum == LeftSum)
20 == 1 ✗ FAIL

↳ LeftSum = LeftSum + nums[index]
= 1 + 7
= 8

Situation: 2



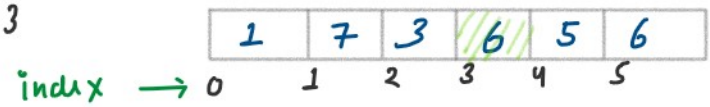
RightSum = 20
LeftSum = 8
index = 2

↳ RightSum = RightSum − nums[index]
= 20 − 3
= 17

↳ (RightSum == LeftSum)
17 == 8 ✗ FAIL

↳ LeftSum = LeftSum + nums[index]
= 8 + 3
= 11

Situation: 3



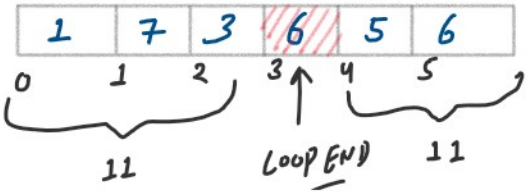
RightSum = 17
LeftSum = 11
index = 3

↳ RightSum = RightSum − nums[index]
= 17 − 6
= 11

↳ (RightSum == LeftSum)
11 == 11 ✓ TRUE

↳ return index

Output : 3



```cpp
// HW 01: Find pivot element (Leetcode-724)
class Solution {
public:
    int pivotIndex(vector<int>& nums) {
        int n=nums.size();
        int rightSum=0;
        int leftSum=0;

        // Step 01: total sum as right sum
        for(int i=0;i<n;i++){
            rightSum+=nums[i];
        }

        for(int i=0;i<n;i++){
            rightSum-=nums[i]; // Subtract element one by one from right sum until left sum equal to right sum
            if(rightSum==leftSum){
                return i; // when leftsum and rigt sum are equal then return index to break the loop
            }
            leftSum+=nums[i]; // left sum is increment by nums[i] element
        }
        return -1; // when there is no index that satisfies the conditions in the problem statement, then return -1
    }
}
```

T.C => O(n)
T.C => O(n)
Time complexity = O(n) + O(n) = O(n)

3. Find Duplicate Number (Leetcode-287)

Example 1:
Input: nums = [1,3,4,2,2]
Output: 2

Example 2:
Input: nums = [3,1,3,4,2]
Output: 3

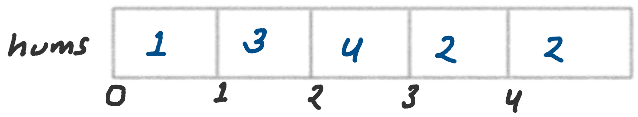
Example 3:
Input: nums = [2,2,2,2,2]
Output: 2

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

APPROACH:

Method 01: Sorting approach
Method 02: Negative marking approach
Method 03: Position and swapping approach

Method 01: Sorting approach

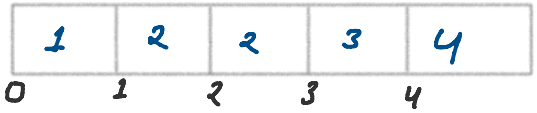


① nums[i] ∈ [1,N]
② size = N+1

output: 2

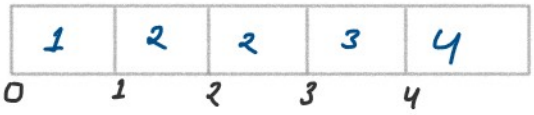
DRY RUN

Step 01: Sort the array



Step 02: Iterate the sorted array

Iteration 0



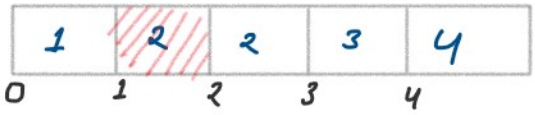
index = 0

(nums[index] != nums[index + 1])

↳ 1 != 2 True

index++;

Iteration 1



index = 1

(nums[index] != nums[index + 1])

↳ 2 != 2 False

return nums[index];

Duplicate = 2 at index 1

```cpp
// Solution 01: Sorting method
class Solution {
public:
    int findDuplicate(vector<int>& nums) {

        // Step 01: Sort nums
        sort(nums.begin(), nums.end());

        // Step 02: Iterate sorted array
        int duplicate=-1;
        for(int i=0;i<nums.size()-1;i++){
            // Check duplicate number
            if(nums[i]==nums[i+1]){
                duplicate = nums[i];
                break;
            }
        }
        return duplicate;
    }
};

// T.C. = O(NlogN)
// S.C. = O(N)
```

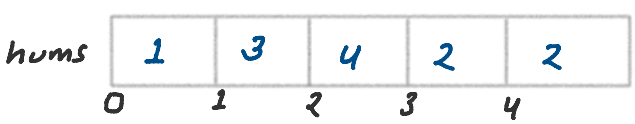
T.C. = O(NlogN)

T.C. = O(N-1)

⇒ O(NlogN) + O(N-1)

⇒ O(NlogN)

***Method 02:** Negative marking approach*



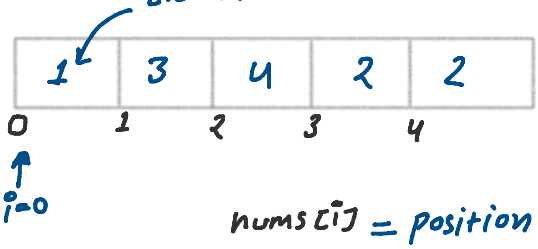
① nums[i] ∈ [1, N]
② size = N + 1

Output: 2

```
Step 01: Iterate the array
Step 02: Mark visited
Step 03: Already visited position then return duplicate
```

DRY RUN

Iteration 0



nums[i] = position = 1

nums[nums[i]] = element = 3

index = 1

index = abs(nums[i])

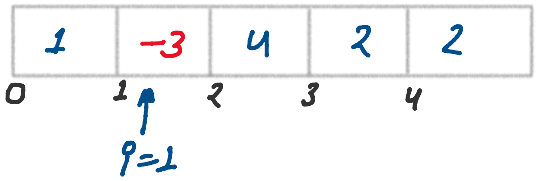
(nums[index] > 0)

3 > 0

nums[index] *= -1;

nums[1] = -3

Iteration 1



index = 2

index = abs(nums[i])

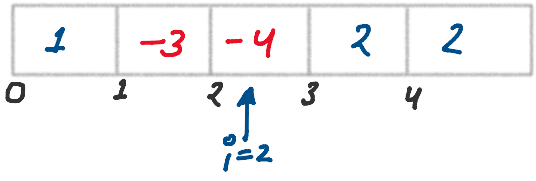
(nums[index] > 0)

4 > 0

nums[index] *= -1;

nums[2] = -4

Iteration 2

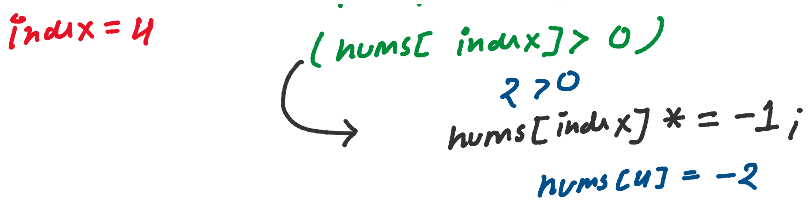


index = 4

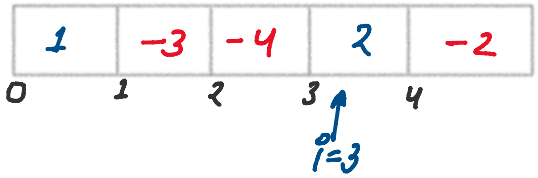
index = abs(nums[i])

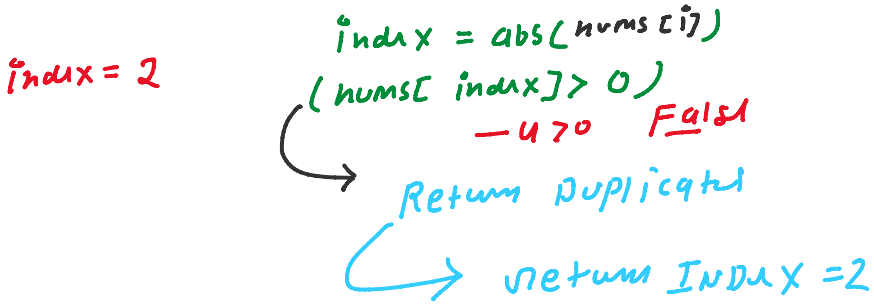
(nums[index] > 0)

2 > 0



Iteration 3

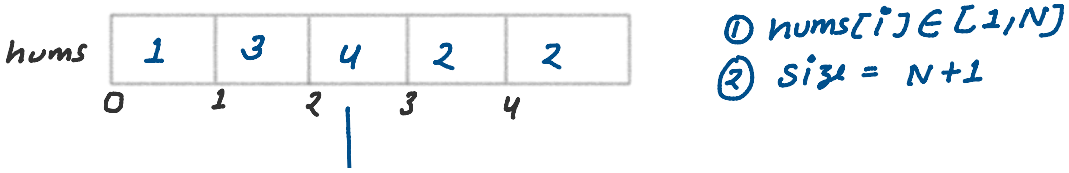





```cpp
// Solution 02: Negative marking method
class Solution {
public:
    int findDuplicate(vector<int>& nums) {

        int duplicate=-1;
        for(int i=0;i<nums.size();i++){

            // Store absolute position temporary
            int index=abs(nums[i]);

            // Not visited position
            if(nums[index]>0){
                nums[index]*=-1;
            }
            // Already visited position
            else{
                duplicate=index;
                return duplicate;
            }
        }
        return duplicate;
    }
};

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

T.C. ⇒ O(n)

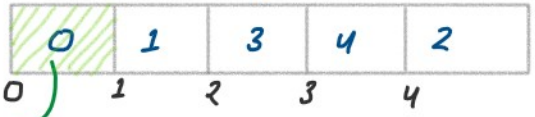
Method 03: *Position and swapping approach* *(Position index==Element)*



(2) Size - N+1

Output: 2

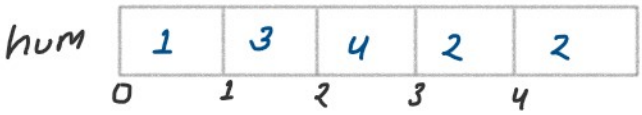
NORMAL



size = 5

Agar 0th index par zero ke alawa koi or element dobara se aata hai to iska matlb wo ek duplicate number hona chaiye

DRY RUN

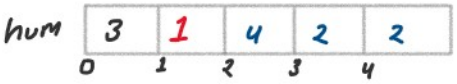


While (num[0] != num[num[0]])

itr:0

1 != 3

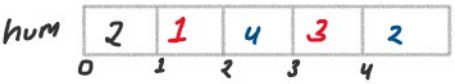
swap (1,3)



itr:1

3 != 2

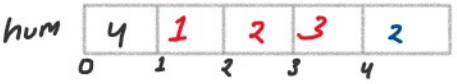
swap (3,2)



itr:2

2 != 4

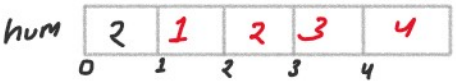
swap (2,4)



itr:3

4 != 2

swap (4,2)



itr:4

2 != 2 False

return num[0] 2 is Duplicate

```cpp
// Solution 03: Position and swaping marking method
class Solution {
public:
    int findDuplicate(vector<int>& nums) {
        while(nums[0]!=nums[nums[0]]){
            swap(nums[0],nums[nums[0]]);
        }
        return nums[0];
    }
};

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

T.C. => O(N)

4. Missing Element From An Array With Duplicates (GFG)

Problem: *Given an array arr[] of size N having integers in the range [1, N] with some of the elements missing. The task is to find the missing elements.*

Note: *There can be duplicates in the array.*

EXAMPLE 01:
Input: arr[] = {1, 3, 5, 3, 4}, N = 5
Output: 2

EXAMPLE 02:
Input: arr[] = {1, 3, 3, 3, 5}, N = 5
Output: 2 4

EXAMPLE 03:
Input: arr[] = {1, 2, 3, 4, 4, 7, 7}, N = 7
Output: 5 6

Approach:
- *Negative marking approach*
- *Sorting approach*

1. Negative marking approach

Observation
- N = 5
- arr[i] ∈ [1, 5]

arr

1	3	5	3	4
0	1	2	3	4

Step 01: Apply Visited Method

-1	3	-5	-3	-4
0	1	2	3	4

Step 02: All positive index are missing

↳ OUTPUT: 2

DRY RUN

Step 01: Apply Visited Method

Iteration:0

1	3	5	3	4
0	1	2	3	4

i = 0

index = 1

index = abs(arr[index]) 1

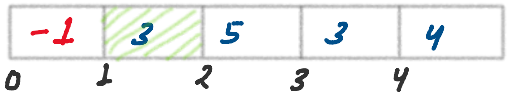
if(arr[index-1] > 0) {

1 > 0

index = 1

```
if( arr[index-1] > 0){
        1 > 0
    arr[index-1] *= -1;
}                arr[0] = -1
```

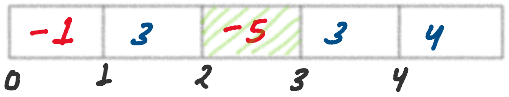
Iteration:1



i = 1
index = 3

```
index = abs(arr[index]) 3
if( arr[index-1] > 0){
        5 > 0
    arr[index-1] *= -1;
}                arr[2] = -5
```

Iteration:2



i = 2
index = 5

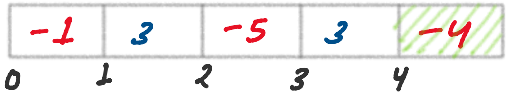
```
index = abs(arr[index]) 5
if( arr[index-1] > 0){
        4 > 0
    arr[index-1] *= -1;
}                arr[4] = -4
```

Iteration:3

i = 3
index = 3

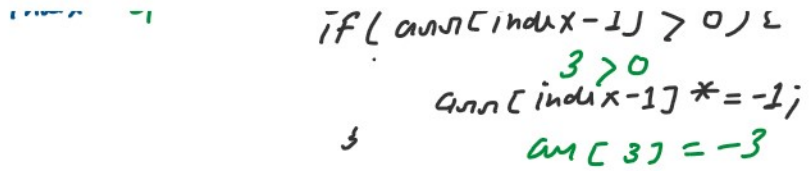
```
index = abs(arr[index]) 3
if( arr[index-1] > 0){
        -5 > 0 ⟶ FAIL
    arr[index-1] *= -1;
}

i++;
```

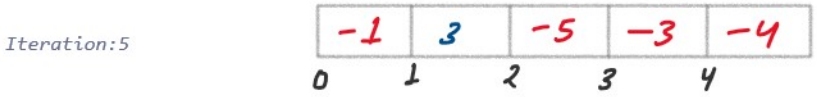
Iteration:4

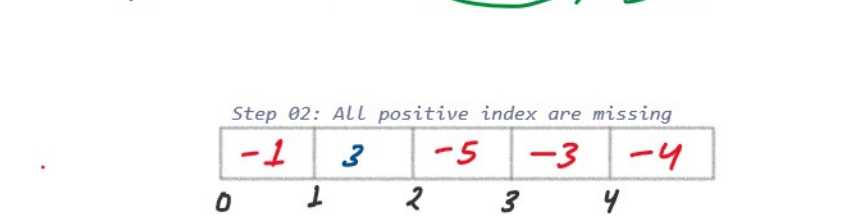


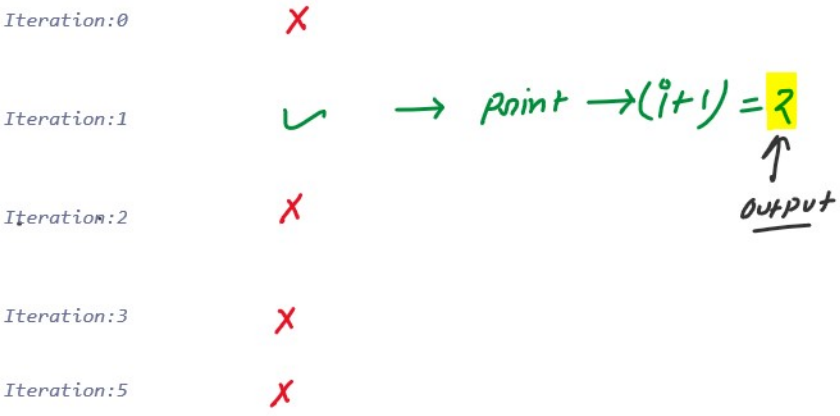
i = 4
index = 4

```
index = abs(arr[index]) 4
if( arr[index-1] > 0){
        3 > 0
    arr[index-1] *= -1;
```










```cpp
// Solution 01: Negative marking method
void findMissing(vector<int> arr){
    int n= arr.size();

    // Step 01: Apply Visited Method
    for(int i=0;i<n;i++){
        int index=abs(arr[i]);
        if(arr[index-1]>0){
            arr[index-1]*=-1;
        }
    }

    // Step 02: All positive index are missing
    for(int i=0;i<n;i++){
        if(arr[i]>0){
            cout<<i+1<<" ";
        }
    }
}
```

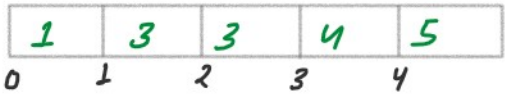
T.C. = O(N)

T.C. = O(N)

T.C. ⇒ O(N) + O(N) = O(N)

2. Sorting approach

Step 01: Sorting array



Step 02: Printing missing element

$(arr[i] != i+1)$

Iteration:0 $i=0$ — $1 != 1$ — False

Iteration:1 $i=1$ — $3 != 2$ — TRUE → print $(i+1) = 2$

Output

Iteration:2 $i=2$ — $3 != 3$ — False

Iteration:3 $i=3$ — $4 != 4$ — False

Iteration:5 $i=4$ — $5 != 5$ — False

```cpp
// Solution 02: Sorting method
void findMissing(vector<int> arr){
    int n= arr.size();

    // Step 01: Sorting array
    sort(arr.begin(),arr.end());

    // Step 02: Printing missing element
    for(int i=0;i<n;i++){
        if(arr[i]!=i+1){
            cout<<i+1<<" ";
        }
    }
}
```

T.C. ⇒ $O(N \log N)$

T.C ⇒ $O(N)$

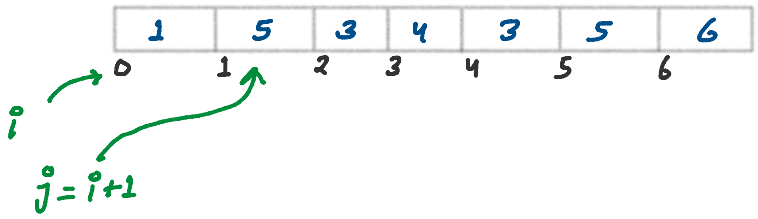
T.C. ⇒ $O(N \log N) + O(N) \Rightarrow O(N \log N)$

5. Find First Repeating Element (GFG)

EXAMPLE 01: ***INPUT:*** *arr{1,5,3,4,3,5,6} and N=7* ***OUTPUT:*** *2* ***Explanation:*** *Value 5 is at index 2*	***EXAMPLE 02:*** ***INPUT:*** *arr{1,2,3,4,5,6,7} and N=7* ***OUTPUT:*** *-1* ***Explanation:*** *No repeated value*

Note: *Position return should be according to 1-based indexing*

Brute force approach



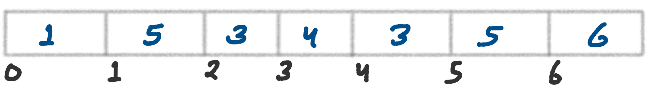
i	j = i+1	arr[i] == arr[j]
0	1	1 == 5 X
	2	1 == 3 X
	3	1 == 4 X
	4	1 == 5 X
	5	1 == 6 X
	6	
1	2	5 == 3 X
	3	5 == 4 X
	4	5 == 3 X
	5	5 == 5

return (i+1);

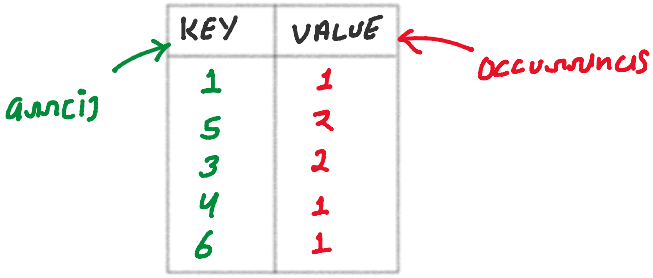
```
// BRUTE FORCE APPROACH
int bruteForceSol(int *arr, int n) {

    for(int i=0;i<n;i++){ ⟶ T.C. = O(N)

        for(int j=i+1;j<n;j++){ ⟶ T.C. = O(N)

            if(arr[i]==arr[j]){
                return i+1;
            }
        }

    }
    return -1;
}
// TIME COMPLEXITY: O(N^2)
// SPACE COMPLEXITY: O(1)
```

⇒ O(N*N)
⇒ O(N²)
T.C ⇒ O(N²)

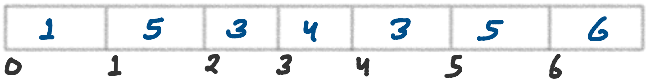
Optimal approach with hashing technique

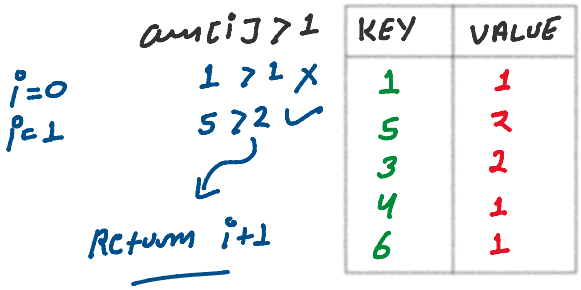


Step 01: *Traverse array and store it's element as hashing*



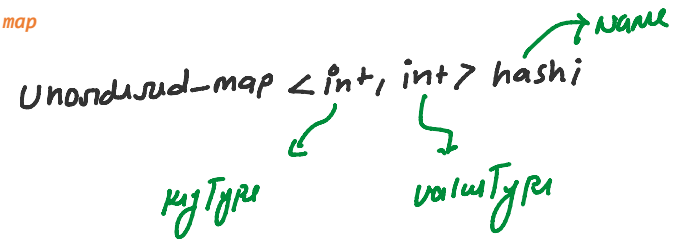
Step 02: *Traverse array to check each element if it has occurrence in future*





Hashing: *We can use unordered map to store array's element in pair of key and value*

Declaration of unordered map



Read more from this resource: https://www.geeksforgeeks.org/unordered_map-in-cpp-stl

```cpp
// OPTIMAL SOLUTION WITH HASHING APPROACH
int optimalSol(int arr[],int n){

    // Declared unordered_map
    unordered_map<int,int> hash;

    // Step 01: Traverse array and store it's element as hashing
    for(int i=0;i<n;i++){
        hash[arr[i]]++;
    }

    // Step 02: Traverse array to check each element if it has occurrence in future
    for(int i=0;i<n;i++){
        if(hash[arr[i]] > 1){
            return i+1;
        }
    }
    return -1;
}
// TIME COMPLEXITY: O(N)
// SPACE COMPLEXITY: O(N)
```

T.C. = O(N)

T.C. = O(N)

T.C. => O(N) + O(N)
=> O(N)

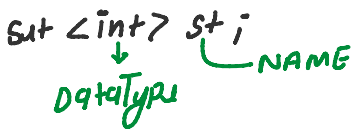
6. Common Element in 3 Sorted Array (GFG)

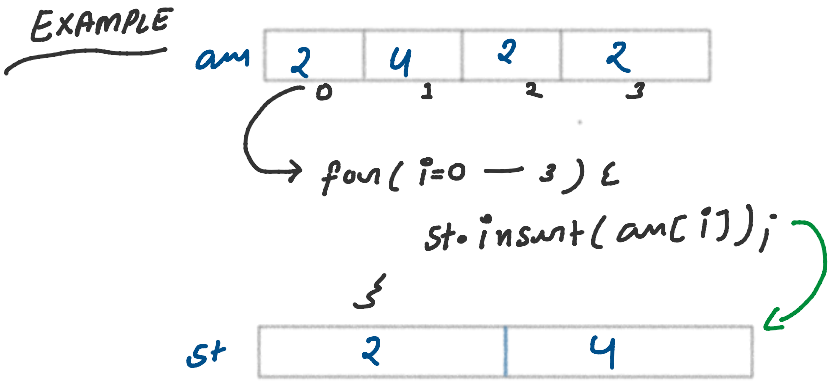
Example 01:	Example 02:
Input: n1 = 6; A = {1, 5, 10, 20, 40, 80} n2 = 5; B = {6, 7, 20, 80, 100} n3 = 8; C = {3, 4, 15, 20, 30, 70, 80, 120} **Output:** 20 80	**Input:** n1 = 3; A = {3, 3, 3} n2 = 3; B = {3, 3, 3} n3 = 3; C = {3, 3, 3} **Output:** 3

Optimal Approach:
Step 01: remove duplicates from sorted array
Step 02: store common value of all three arrays in new array

Step 01: store unique element in new array with the help of C++ STL Data Structure is **set(dataType)**

Declaration of set

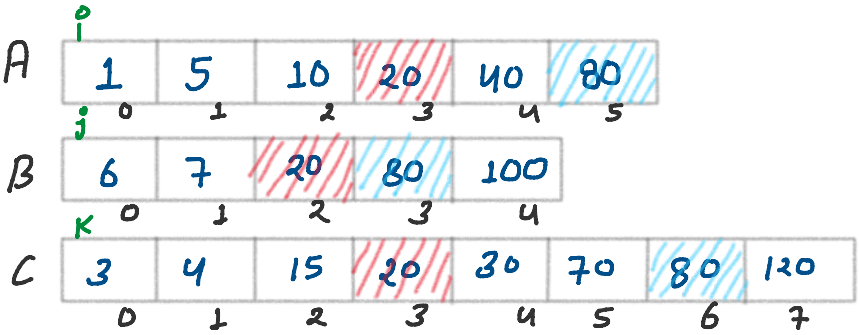




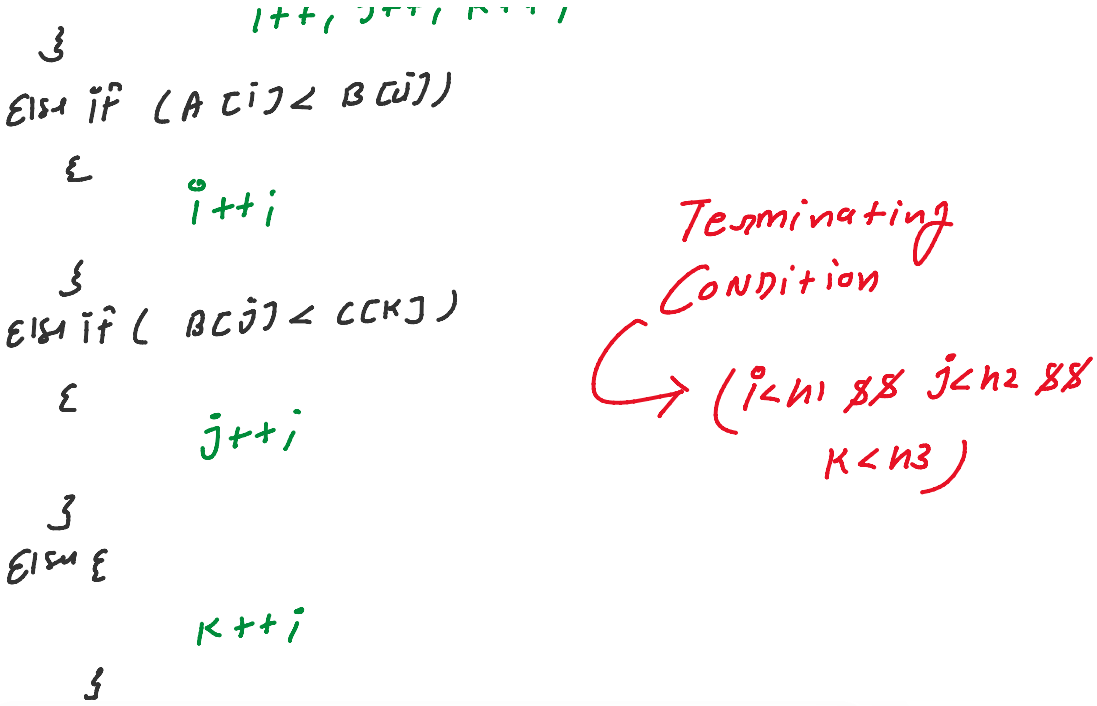
Read more from this resource: https://www.geeksforgeeks.org/set-in-cpp-stl/

DRY RUN

Step 02: store common value of all three arrays in new array

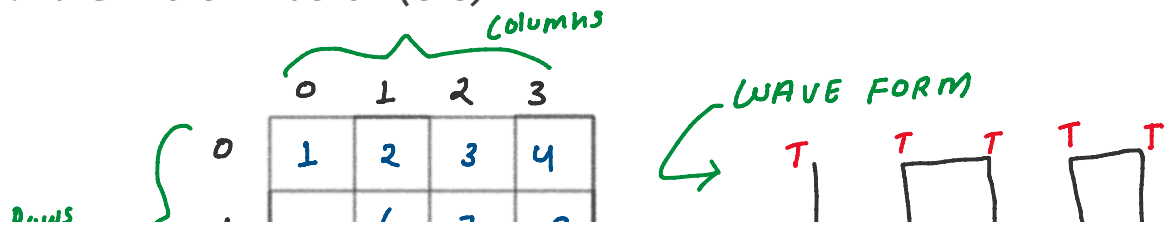


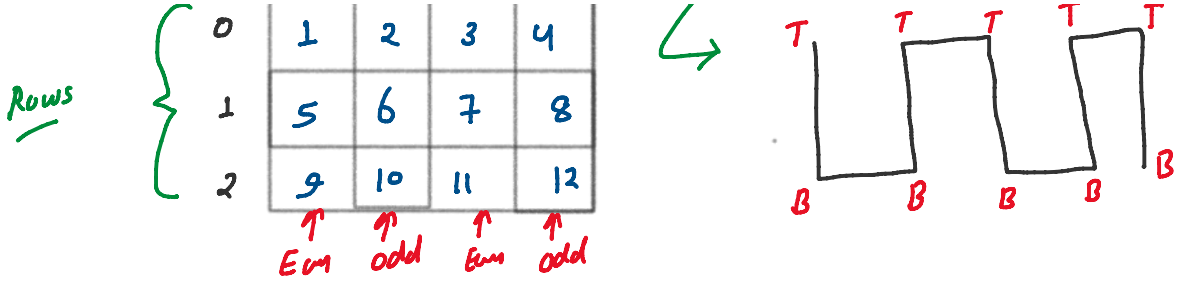

```
if(A[i]==B[j] && B[j]==C[k] && C[k]==A[i])
{
    st.insert(A[i]);
    i++, j++, k++;
}
else if(A[i]<B[j])
```




```cpp
// Common Element in 3 Sorted Array (GFG)
class Solution
{
    public:
       vector <int> commonElements (int A[], int B[], int C[], int n1, int n2, int n3)
        {
            vector<int> ans;
            set<int> st;
            int i=0,j=0,k=0;

            while(i<n1 && j<n2 && k<n3)
            {

                if(A[i]==B[j] && B[j]==C[k] && C[k]==A[i])
                {
                    st.insert(A[i]);
                    i++, j++, k++;
                }
                else if(A[i]<B[j])
                {
                    i++;
                }
                else if(B[j]<C[k])
                {
                    j++;
                }
                else
                {
                    k++;
                }
            }

            for(auto i: st)
            {
                ans.push_back(i);
            }

            return ans;
        }

};
// TIME COMPLEXITY: O(N1+N2+N3)
// SPACE COMPLEXITY: O(N1+N2+N3)
```

7. Wave Print A Matrix (GFG)



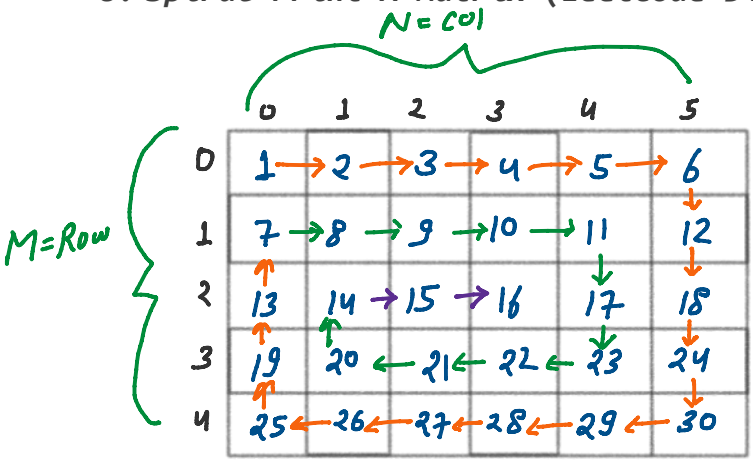


Approach:
- *When number of column is **even** then print row **top to bottom***
- *When number of column is **odd** then print row **bottom to top***

```cpp
// Print wave matrix
void printWaveMatrix(vector<vector<int>> matrix){
    int row = matrix.size();
    int col = matrix[0].size();

    // Iterate array column wise
    for(int startCol=0; startCol<col; startCol++){

        // when startCol==even then print row top to bottom
        if((startCol & 1)==0){
            for(int i=0;i<row;i++){
                cout<<matrix[i][startCol]<<" ";
            }
        }
        // when startCol==odd then print row bottom to top
        else{
            for(int i=row-1;i>=0;i--){
                cout<<matrix[i][startCol]<<" ";
            }
        }
    }
}
```

T.C. => O(M x N)

8. Spiral Print A Matrix (Leetcode-54)



Output:

1 2 3 4 5 6 12 18 24 30 29 28 27 26 25

19 13 7 8 9 10 11 17 23 22 21 14 15 16

startingRow	= 0
endingCol	= 5
endingRow	= 4
startingCol	= 0

startingRow	= 1
endingCol	= 4
endingRow	= 3
startingCol	= 1

startingRow	= 2
endingCol	= 3
endingRow	= 2
startingCol	= 2

Optimal Approach:
Step 01: find total elements
Step 02: iterate matrix till end of the total element and print
- *Print startingRow*
- *Print endingCol*
- *Print endingRow*

- Print startingCol

```cpp
// HW 08: Spiral Print A Matrix (Leetcode-54)
class Solution {
public:
    vector<int> spiralOrder(vector<vector<int>>& matrix) {

        vector<int> ans;
        int m=matrix.size();
        int n=matrix[0].size();

        // Step 01: Total elements of matrix
        int totalElements=m*n;

        int startingRow = 0;
        int endingCol = n-1;
        int endingRow = m-1;
        int startingCol = 0;

        // Step 02: Iterate matrix till end of the total element
        int count=0;
        while(count<totalElements){
            // Print startingRow
            for(int i=startingRow;i<=endingCol && count<totalElements;i++){
                ans.push_back(matrix[startingRow][i]);
                count++;
            }
            startingRow++;

            // Print endingCol
            for(int i=startingRow;i<=endingRow && count<totalElements;i++){
                ans.push_back(matrix[i][endingCol]);
                count++;
            }
            endingCol--;

            // Print endingRow
            for(int i=endingCol;i>=startingCol && count<totalElements;i--){
                ans.push_back(matrix[endingRow][i]);
                count++;
            }
            endingRow--;

            // Print startingCol
            for(int i=endingRow;i>=startingRow && count<totalElements;i--){
                ans.push_back(matrix[i][startingCol]);
                count++;
            }
            startingCol++;
        }
        return ans;
    }
};


/*
Example 01:
Input:  {{1,   2,   3,   4},
        {5,   6,   7,   8},
        {9,   10,  11,  12},
        {13,  14,  15,  16 }}

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

Example 02:
Input: {{1,   2,   3,   4,  5,   6},
        {7,   8,   9,  10,  11,  12},
        {13,  14,  15, 16,  17,  18}}

Output: 1 2 3 4 5 6 12 18 17 16 15 14 13 7 8 9 10 11
*/
```

15. Add two numbers represented by two array (GFG)

Example 01:
Input : A[] = {1, 2}, B[] = {2, 1}
Output : 33

Example 02:
Input : A[] = {9, 5, 4, 9}, B[] = {2, 1, 4}
Output : 9763

Example 03:
Input : A[] = {2, 1, 4}, B[] = {9, 5, 4, 9}
Output : 9763

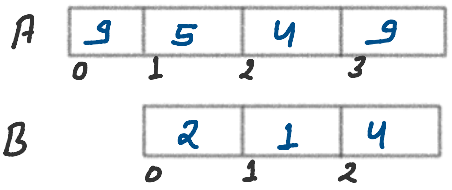
Example 04:
Input : A[] = {9, 1, 4}, B[] = {8, 1, 4}
Output : 1728

Example 05:
Input : A[] = {0, 0, 2, 1, 4}, B[] = {0, 0, 2, 1, 4}
Output : 428

DRY RUN

DRY RUN

Example 02:

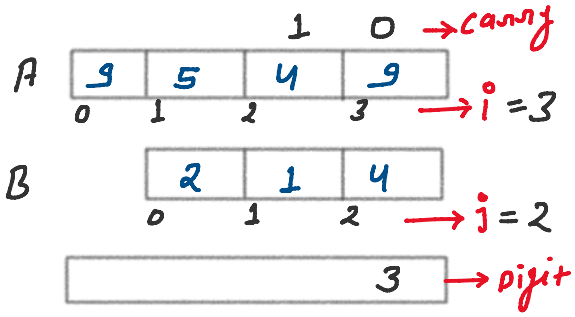


N = 4

M = 3

Output: 9763

Iteration 0



(i>=0 && j>=0)

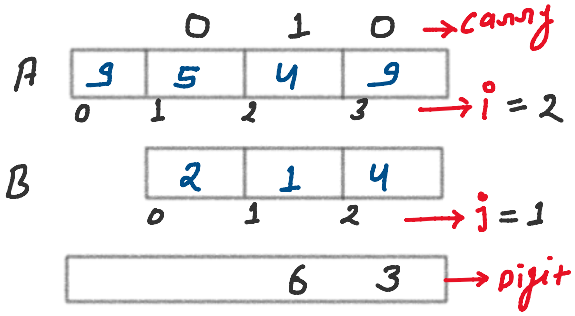
carry = 0

X = A[i] + B[j] + carry
= 9 + 4 + 0
= 13

digit = X % 10 i
= 3

carry = X / 10 i
= 1

Iteration 1



(i>=0 && j>=0)

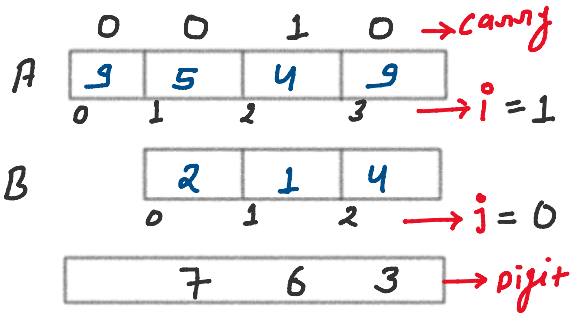
carry = 1

X = A[i] + B[j] + carry
= 4 + 1 + 1
= 6

digit = X % 10 i
= 6

carry = X / 10 i
= 0

Iteration 2



(i>=0 && j>=0)

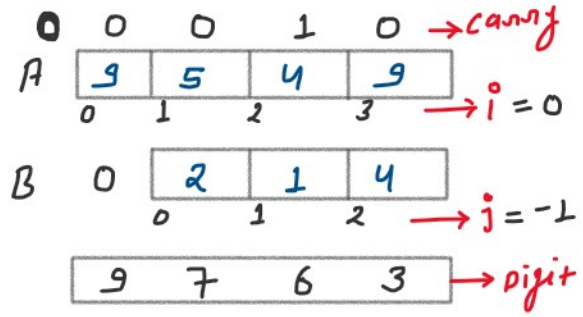
carry = 0

X = A[i] + B[j] + carry
= 5 + 2 + 0
= 7

digit = X % 10 i
= 7

carry = X / 10 i
= 0

Iteration 3



(j >= 0)

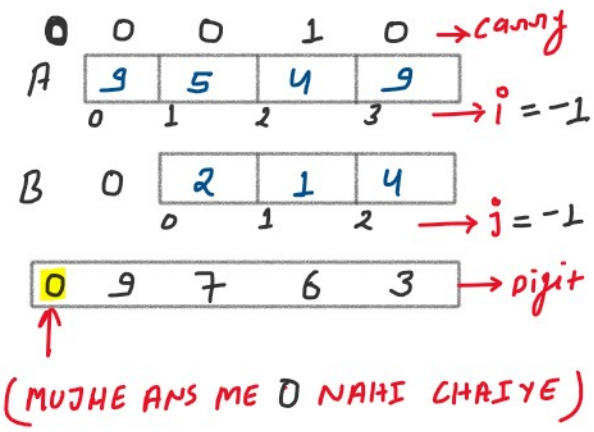
carry = 0

X = A[i] + 0 + carry
= 9 + 0 + 0
= 9

digit = X % 10 i
= 9

carry = X / 10 i
= 0

Iteration 4



(MUJHE ANS ME 0 NAHI CHAIYE)

if(carry > 0) FALSE

push karo ANS ME AGAR TRUE HAI

Yeh Ans actual me kis tarike se save hai ans ke andar

ANS | 3 6 7 9 |

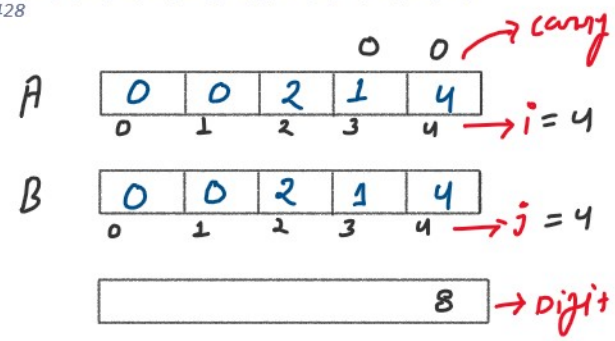
REVERSE

Final ANS | 9 7 6 3 |

DRY RUN

Example 05:
Input : A[] = {0, 0, 2, 1, 4}, B[] = {0, 0, 2, 1, 4}
Output : 428

Iteration 0



(i >= 0 && j >= 0)

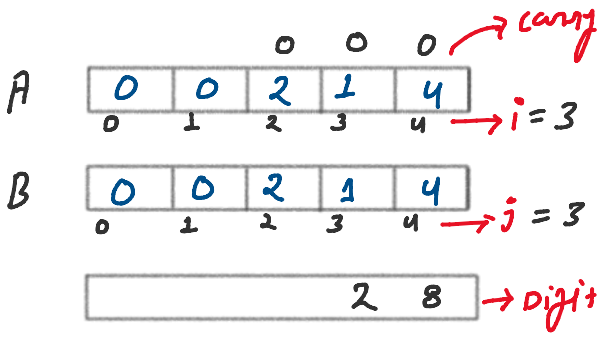
carry = 0

X = A[i] + B[j] + carry
= 4 + 4 + 0 = 8

digit = X % 10
= 8

carry = X / 10
= 0

Iteration 1



(i>=0 && j>=0)

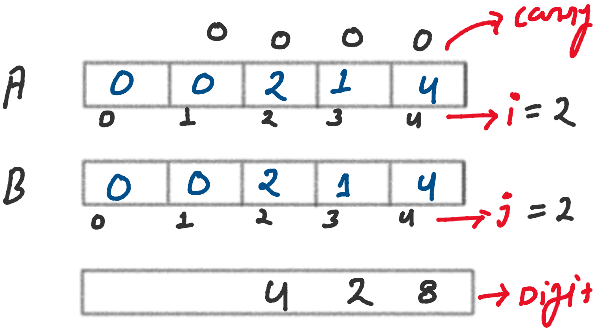
carry = 0

X = A[i] + B[j] + carry
= 1 + 1 + 0 = 2

digit = X % 10
= 2

carry = X / 10
= 0

Iteration 2



(i>=0 && j>=0)

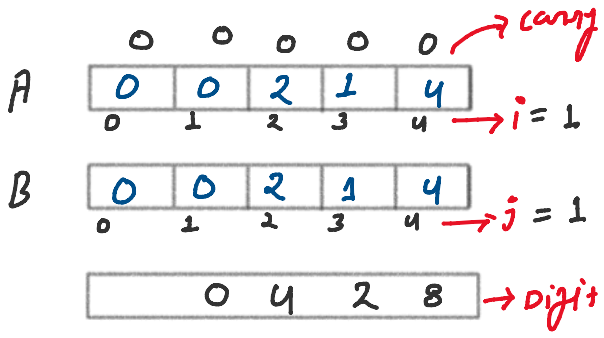
carry = 0

X = A[i] + B[j] + carry
= 2 + 2 + 0 = 4

digit = X % 10
= 4

carry = X / 10
= 0

Iteration 3



(i>=0 && j>=0)

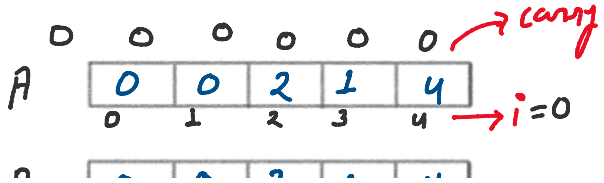
carry = 0

X = A[i] + B[j] + carry
= 0 + 0 + 0 = 0

digit = X % 10
= 0

carry = X / 10
= 0

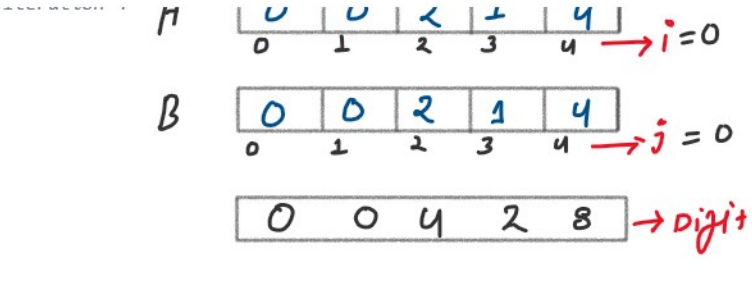
Iteration 4



(i>=0 && j>=0)

carry = 0

X = A[i] + B[j] + carry
= 0 + 0 + 0 = 0

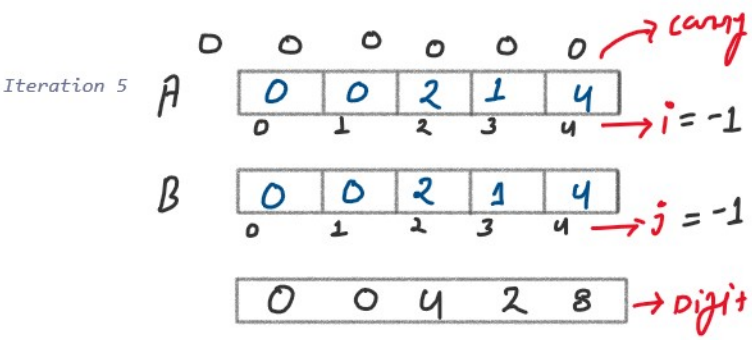


$X = A[i] + B[j] + carry$
$= 0 + 0 + 0 = 0$

$digit = X \% 10$
$= 0$

$carry = X / 10$
$= 0$

Iteration 5



if(carry > 0) FAIL

↳ push karado ANS ME AGAR TRUE hai

Yeh Ans actual me kis tarike se save hai ans ke andar

ANS 8 2 4 0 0

↓ REVERSE

Final ANS 0 0 4 2 8

WRONG ANS

FIR Right ANS kaise AYEGA

Solution

ANS 8 2 4 0 0
0 1 2 3 4

while (ans[ans.size()-1] == '0')

↳ ans.pop(); Delete Last Element

① size = 5 ans[4] = 0

8 2 4 0
0 1 2 3

② size = 4 ans[3] = 0

8 2 4

↓ REVERSE

RUK jao

4 2 8

CORRECT ANS

```cpp
// HW 15: Add two numbers represented by two array (GFG)
#include<iostream>
#include<vector>
#include <bits/stdc++.h>
using namespace std;

// Add two numbers represented by two array
```

```cpp
// HW 15: Add two numbers represented by two array (GFG)
#include<iostream>
#include<vector>
#include <bits/stdc++.h>
using namespace std;

// Add two numbers represented by two array
vector<int> calc_Sum(int a[],int n,int b[],int m){
    int i=n-1;
    int j=m-1;
    vector<int> ans;
    int carry = 0;

    // Case: 01 Jab i and j both are greater than or equal to 0 honge
    while(i>=0 && j>=0){
        int x = a[i] + b[j] + carry;
        int digit = x % 10;
        // Push digit in ans
        ans.push_back(digit);
        carry = x / 10;
        i--,j--;
    }
    // Case 02: Jab i greater than or equal to 0 hoga
    while(i>=0){
        int x = a[i] + 0 + carry;
        int digit = x % 10;
        // Push digit in ans
        ans.push_back(digit);
        carry = x / 10;
        i--;
    }
    // Case 03: Jab j greater than or equal to 0 hoga
    while(j>=0){
        int x = 0 + b[j] + carry;
        int digit = x % 10;
        // Push digit in ans
        ans.push_back(digit);
        carry = x / 10;
        j--;
    }
    // Case 04: Jab Carry greater than 0 hoga
    if(carry > 0){
        ans.push_back(carry);
    }
    // Case 05: Jab actual ans ke last position par zero ho tab tak use delete karte rahenge
    while(ans[ans.size()-1] == 0){
        ans.pop_back();
    }
    // Actual ans ko reverse krdo correct ans ke liye
    reverse(ans.begin(), ans.end());
    return ans;
}

int main(){
    int a[]={9, 5, 4, 9};
    int b[]={2, 1, 4};
    int n = sizeof(a)/sizeof(a[0]);
    int m = sizeof(b)/sizeof(b[0]);

    vector<int> currectAns = calc_Sum(a,n,b,m);
    for (auto i : currectAns) {
        cout<<i;
    }

    return 0;
}

/*
Time Complexity: O(N + M)
Space Complexity: O(N + M)

Where N and M are size of array of a and b respectively
*/
```

9. Factorial of A Large Number (GFG)

```
Example 01:
Enter number: 5
120
```

```
Example 02:
Enter number: 100
933262154439441526816992388562667004907159682643816214685929638952175999932299156089414639761565182862536
```

```
Example 01:
Enter number: 5
120
```

```
Example 02:
Enter number: 100
93326215443944152681699238856266700490715968264381621
46859296389521759999322991560894146397615651828625 36
97920827223758251185210916864000000000000000000000000
```

DRY RUN

$N = 5! \Rightarrow 1 * 2 * 3 * 4 * 5 = 120$

i ↷

carry = 0

x = ans[jj] * i + carry

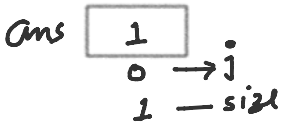
ans[jj] = x % 10

carry = x / 10

ans initially 1 element store karega

Situation: 01

carry = 0 i = 2 jj = 0 size = 1



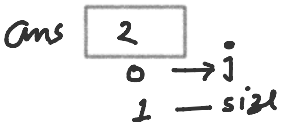
x = ans[jj] * i + carry
= 1 * 2 + 0
= 2

ans[jj] = x % 10
= 2

carry = x / 10
= 0

Situation: 02

carry = 0 i = 3 jj = 0 size = 1



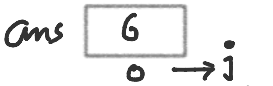
x = ans[jj] * i + carry
= 2 * 3 + 0
= 6

ans[jj] = x % 10
= 6

carry = x / 10
= 0

Situation: 03

carry = 0 i = 4 jj = 0 size = 1



carry = 0 i = 4 j

ans | 6 | (0 → j, 1 — size)

X = ans[j] * i + carry
= 6 * 4 + 0
= 24

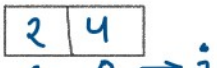
ans[j] = X % 10
= 4

carry = X / 10
= 2

⇒ carry = 2 i = 4 j = 1 size = 1

while (carry > 0) { ans.push_back(carry % 10); ⇒ 2
carry = carry / 10 ; ⇒ 0
}

Jab tak mera carry equal to zero nahi ho jayega tab tak me ek new element add karta rahunga



Situation : 04

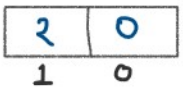
carry = 0 i = 5 j = 0 size = 2

ans | 2 | 4 | (1 0, size = 2)

X = ans[j] * i + carry
= 4 * 5 + 0
= 20

ans[j] = X % 10
= 0

carry = X / 10
= 2



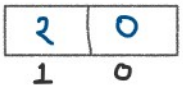
carry = 2 i = 5 j = 1 size = 2

ans | 2 | 0 | (1 0, size = 2)

X = ans[j] * i + carry
= 2 * 5 + 2
= 12

ans[j] = X % 10
= 2

carry = X / 10
= 1

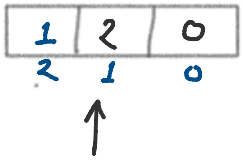


carry = x / 10
= 1

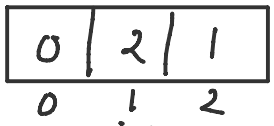
⇒ carry = 1 i = 5 j = 3 size = 2

while (carry > 0)
{
ans.push_back (carry % 10); ⇒ 1
carry = carry / 10 ; ⇒ 0
}

i = 6
N = 5
} RUK jaao... (i < N)



Actual



REVERSE

```cpp
// HW 09: Factorial of A Large Number (GFG)
#include<iostream>
#include<vector>
#include <bits/stdc++.h>
using namespace std;

vector<int> factorial(int N){
    vector<int> ans;
    ans.push_back(1);
    int carry = 0;

    for(int i=2;i<=N;i++){
        for(int j=0;j<ans.size();j++){
            int x=ans[j]*i+carry;
            // Jo pahle se element rakhe hue hai une me update karta rahunga jab tak j<arr.size() hai
            ans[j]=x%10;
            carry=x/10;
        }
        // Jab tak mera carry equal to zero nahi ho jayega tab tak me ek new element add karta rahunga
        while(carry>0){
            ans.push_back(carry%10);
            carry/=10;
        }
    }

    // Reverse the ans to get the correct value of factorial
    reverse(ans.begin(), ans.end());

    // Returning correct ans
    return ans;
}

int main(){
    int N;
    cout<<"Enter number: ";
    cin>>N;

    vector<int> ans = factorial(N);
    for (auto factorialVal : ans) {
        cout<<factorialVal;
    }

    return 0;
}

/*

Factorial of A Large Number (GFG)
Example 01:
Enter number: 5
120


Example 02:
Enter number: 100
933262154439441526816992388562667004907159682643816214685929638952175999932299156089414639761565182862536
979208272237582511852109168640000000000000000000000000
*/
```

T.C. => O(N)

10. Key Pair/Two Sum (GFG and Leetcode-1)

Example 01: *Input:* *N = 6, X = 16* *Arr[] = {1, 4, 45, 6, 10, 8}* *Output: Yes* *Explanation: Arr[3] + Arr[4] = 6 + 10 = 16*	***Example 02:*** *Input:* *N = 5, X = 10* *Arr[] = {1, 2, 4, 3, 6}* *Output: Yes* *Explanation: Arr[2] + Arr[4] = 4 + 6 = 10*

Example:01

arr

1	4	45	6	10	8
0	1	2	3	4	5

X = 16

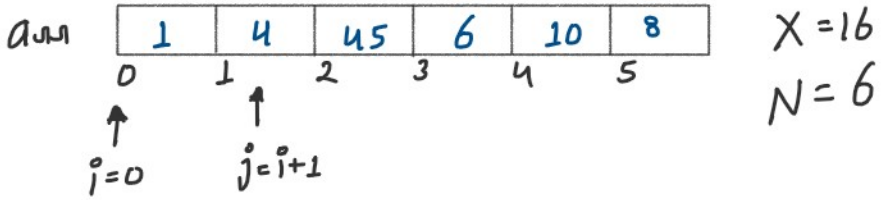
N = 6

Total pairs

(1,4) (4,45) (45,6) (6,10) (10,8)
(1,45) (4,6) (45,10) (6,8)
(1,6) (4,10) (45,8)
(1,10) (4,8)
(1,8)

(6,10) → SUM = 16, Target = 16 → YES

Brute Force Approach:
Step 01: find all pairs
Step 02: check all pair's sum




```
for(int i=0 ; i<N ; i++){
    for(int j=i+1 ; j<N ; j++) {
        if( arr[i] + arr[j] == X)
        {
            return True;
        }
    }
}
return false;
```

Outer Loop
Inner Loop
Step:1
Step:2
{ Check Sum

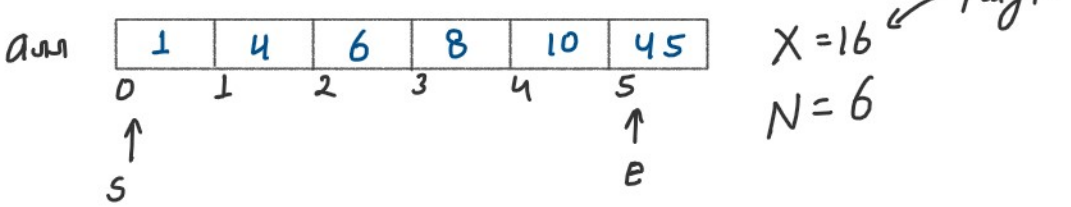
TC = O(NXN) = O(N²)
S.C = O(1)

Two Pointers Approach:
Step 01: Sort the array
Step 02: agar arr[s]+arr[e]==target hai to return YES kardo

Step 03: agar arr[s]+arr[e]<target to mujhe nearly answer s++ karne par jaldi mil skta hai

Step 04: agar arr[s]+arr[e]>target to mujhe nearly answer e-- karne par jaldi mil skta hai

Intuition:01



s = 0
e = 5
x = 16

(arr[s] + arr[e] > X)

1 + 45 > 16

e--

Iteration: 02

arr

1	4	6	8	10	45
0	1	2	3	4	5

s ↑ e ↑

X = 16 ← target

N = 6

s = 0
e = 4
x = 16

(arr[s] + arr[e] < X)

1 + 10 < 16

s++

Iteration: 03

arr

1	4	6	8	10	45
0	1	2	3	4	5

s ↑ e ↑

X = 16 ← target

N = 6

s = 1
e = 4
x = 16

(arr[s] + arr[e] < X)

4 + 10 < 16

s++

Iteration: 04

arr

1	4	6	8	10	45
0	1	2	3	4	5

↑ e ↑

X = 16 ← target

N = 6

s = 2
e = 4
x = 16

(arr[s] + arr[e] == X)

6 + 10 == 16

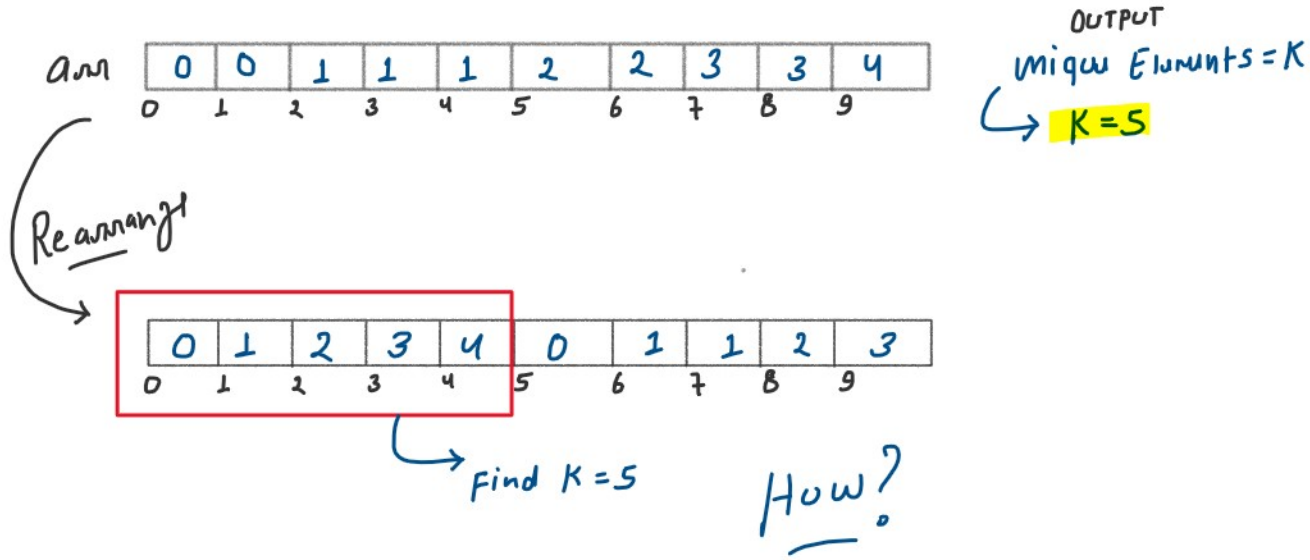
return true

OUTPUT: YES

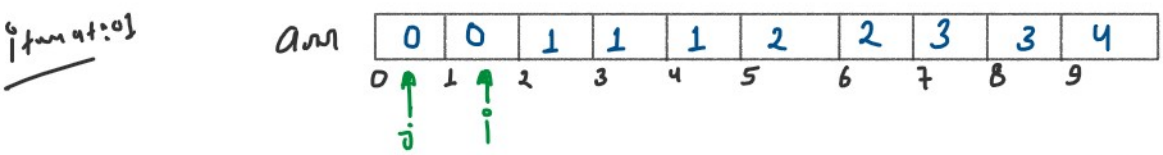
```cpp
// HW 10: Key Pair/two Sum (GFG and Leetcode-1)
#include<iostream>
#include<vector>
#include <bits/stdc++.h>
using namespace std;

// TWO SUM
bool twoSum(vector<int> arr,int X){

    // Step 01: Sort the array
    sort(arr.begin(),arr.end());

    int N=arr.size();
    int s=0;
    int e=N-1;
```

```cpp
// HW 10: Key Pair/Two Sum (GFG and Leetcode-1)
#include<iostream>
#include<vector>
#include <bits/stdc++.h>
using namespace std;

// TWO SUM
bool twoSum(vector<int> arr,int X){

    // Step 01: Sort the array
    sort(arr.begin(),arr.end());

    int N=arr.size();
    int s=0;
    int e=N-1;

    while(s<e){

        // Step 02
        if(arr[s]+arr[e]==X){
            return true;
        }
        // Step 03:
        // If pair sum is less than X, it means current pair is so small;
        // We have to move the starting index to right to consider larger pair.
        else if (arr[s]+arr[e]<X)
        {
            s++;
        }
        // Step 04:
        // If pair sum is greater than X, it means current pair is too large,
        // We have to move the ending index to left to consider smaller pair.
        else{
            e--;
        }
    }
    return false;
}

int main(){
    vector<int> arr{1, 4, 45, 6, 10, 8};
    int X=16;

    bool ans = twoSum(arr,X);

    if(ans){
        cout<<"YES"<<endl;
    }
    else{
        cout<<"NO"<<endl;
    }

    return 0;
}

/*
Time Complexity: O(NlogN)+O(N)->O(NlogN), Where N is size of array
Space Complexity: O(1)
*/
```

11. Remove Duplicates From Sorted Array (Leetcode-26)



Two Pointers Approach:



(where j is a position of unique element)

j=0
i=1

arr[i] == arr[j]
0 == 0
→ i++

j=0
i=1

arr[i] == arr[j]
0 == 0
i++

i from 2

arr

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

j, i

j=0
i=2

arr[i] != arr[j]
1 != 0
j++

arr[j] = arr[i]
i++

i from 3

arr

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

j, i

j=1
i=3

arr[i] == arr[j]
1 == 1
i++

i from 4

arr

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

j, i

j=1
i=4

arr[i] == arr[j]
1 == 1
i++

i from 5

arr

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

j, i

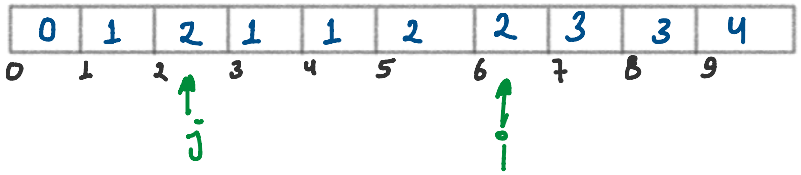
j=1
i=5

arr[i] != arr[j]
2 != 1

j = 1
i = 5

arr[i] != arr[j]
2 != 1
j++
arr[j] = arr[i]
i++

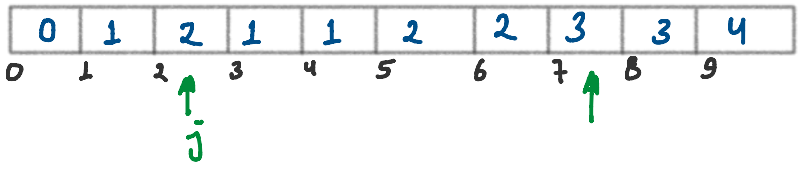
i from 6:



j = 2
i = 6

arr[i] == arr[j]
2 == 2
i++

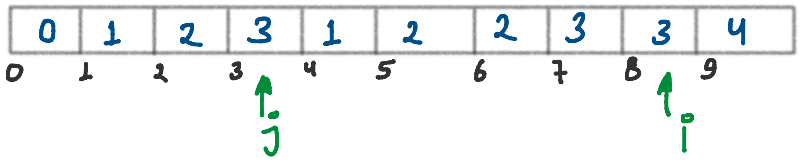
i from 7:



j = 2
i = 7

arr[i] != arr[j]
3 != 2
j++
arr[j] = arr[i]
i++

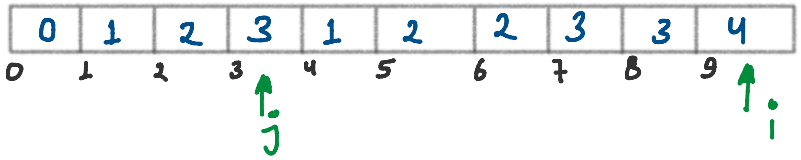
i from 8:

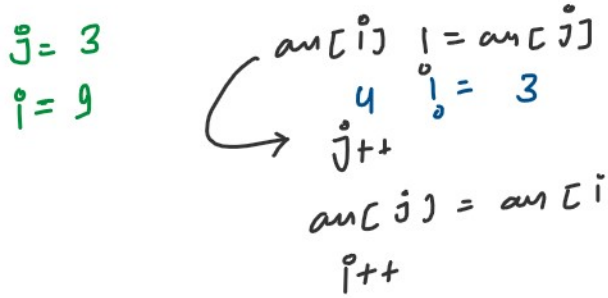


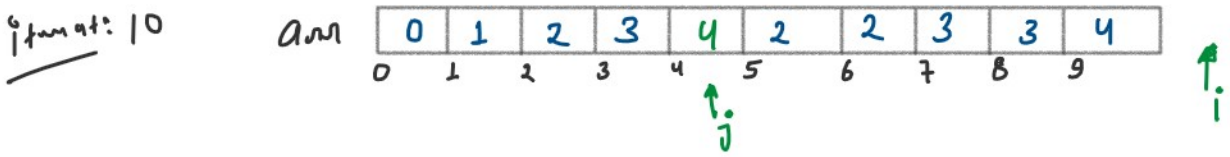
j = 3
i = 8

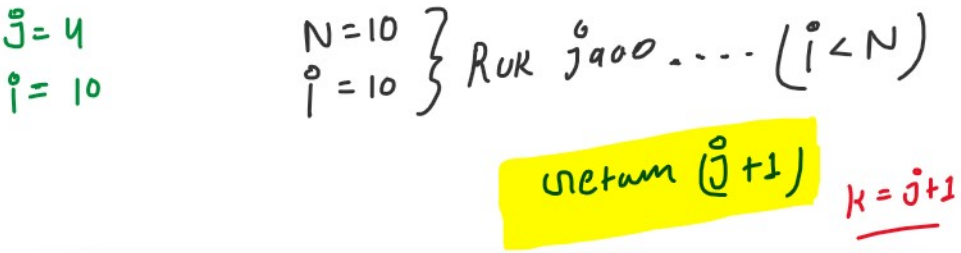
arr[i] == arr[j]
3 == 3
i++

i from 9:



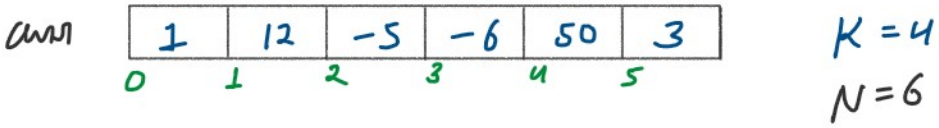







```cpp
// HW 11: Remove Duplicates From Sorted Array (Leetcode-26)
class Solution {
public:
    int removeDuplicates(vector<int>& nums) {
        int N=nums.size();
        int j=0; // J is a unique element's position
        int i=1;

        while(i<N){
            if(nums[i]==nums[j]){
                i++;
            }else{
                nums[++j]=nums[i++];
            }
        }
        return j+1;
    }
};
/*
Time Complexity: O(N), Where N is size of array
Space Complexity: O(1)
*/

/*
Example 1:
Input: nums = [1,1,2]
Output: 2, nums = [1,2,_]
Explanation: Your function should return k = 2, with the first two elements of nums being 1 and 2 respectively.
It does not matter what you leave beyond the returned k (hence they are underscores).

Example 2:
Input: nums = [0,0,1,1,1,1,2,2,3,3,4]
Output: 5, nums = [0,1,2,3,4,_,_,_,_,_,_]
Explanation: Your function should return k = 5, with the first five elements of nums being 0, 1, 2, 3, and 4
respectively.
It does not matter what you leave beyond the returned k (hence they are underscores).
*/
```

12. Maximum Average Subarray 1 (Leetcode-643)



K = 4
N = 6

METHOD:1

Brute Force Approach:
Step 01: *find all sub arrays sum of length K*
Step 02: *find maximum average from sub arrays*

Step:01

i=0 j=3 ⇒ maxSum = 1+12+(-5)+(-6) = 2

i=1 j=4 ⇒ maxSum = 12+(-5)+(-6)+50 = 51

i=2 j=5 ⇒ maxSum = (-5)+(-6)+50+3 = 42

i=3 j=6 Ruk jaaoo.... (j<N) END

Step:02

$$maxAvg = \frac{maxSum}{K} = \frac{51}{4} \Rightarrow 12.75$$ output

```cpp
// Method 01: Brute force solution
class Solution {
public:
    double findMaxAverage(vector<int>& nums, int k) {
        int N = nums.size();
        int maxSum = INT_MIN;
        int i = 0, j=k-1;

        while(j<N){
            int sum = 0;
            for(int x=i;x<=j;x++){
                sum = sum + nums[x];
            }
            maxSum = max(maxSum, sum);
            j++, i++;
        }


        // Find maximum average of maximum sum
        double maxAvg = maxSum/(double)k;
        return maxAvg;
    }
};
/*
Time Complexity: O(N^2), Where N is size of array
Space Complexity: O(1)
*/
```

arr

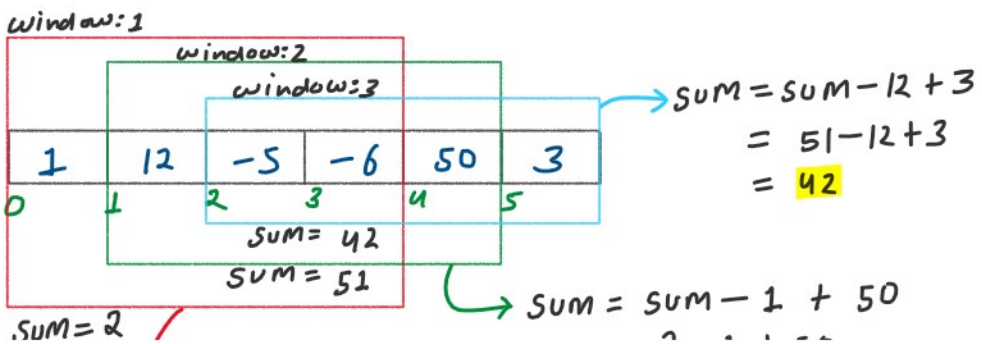
1	12	-5	-6	50	3
0	1	2	3	4	5

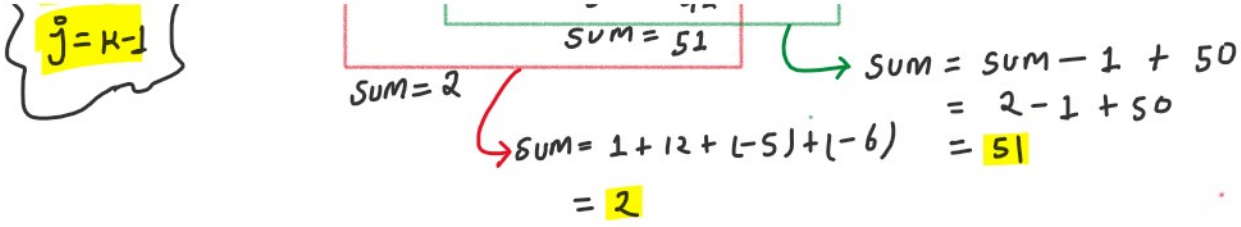
K = 4

N = 6

METHOD: 2

Optimal Solution: Sliding Window Approach
Step 01: sum of first window
Step 02: first window ke first element ko subtract kardo and next new element ko add krdo with sum of first windo me





GENERAL FORMULA

step- (1) sum = 2

step- (2) sum = sum - arr[i]
i++ and j++
sum = sum + arr[j]

```cpp
// Optimal Solution: Sliding Window Approach
class Solution {
public:
    double findMaxAverage(vector<int>& nums, int k) {
        int N = nums.size();
        int i = 0, j=k-1;

        int sum = 0;
        // Step 01: Find sumof first window
        for(int x=i;x<=j;x++){
            sum = sum + nums[x];
        }

        int maxSum = sum;
        // Step 02: Find maxSum
        // Me pahle hi check kar loonga ki j out of bound to nhi ho rha hai
        j++; // pahle aap j ko use karlo uske baad increment by one kardo
        while(j<N){
            sum = sum - nums[i++];
            sum = sum + nums[j++];
            maxSum = max(maxSum, sum);
        }

        // Find maximum average of maximum sum
        double maxAvg = maxSum/(double)k;
        return maxAvg;
    }
};

/*
Time Complexity: O(N), Where N is size of array
Space Complexity: O(1)
*/

/*
Example 1:
Input: nums = [1,12,-5,-6,50,3], k = 4
Output: 12.75000 26.50000
Explanation: Maximum average is (12 - 5 - 6 + 50) / 4 = 51 / 4 = 12.75

Example 2:
Input: nums = [5], k = 1
Output: 5.00000

Example 3:
Input: nums = [1,12,-5,-6,50,3], k = 2
Output: 26.50000
*/
```

13. Find Pivot Index with prefix sum approach (Leetcode-724)

Example 1: *Input: nums = [1,7,3,6,5,6]* *Output: 3* *Explanation:* *The pivot index is 3.* *Left sum = nums[0] + nums[1] + nums[2] = 1 + 7 + 3 = 11* *Right sum = nums[4] + nums[5] = 5 + 6 = 11*	*Example 2:* *Input: nums = [1,2,3]* *Output: -1* *Explanation:* *There is no index that satisfies the conditions in the problem statement.*

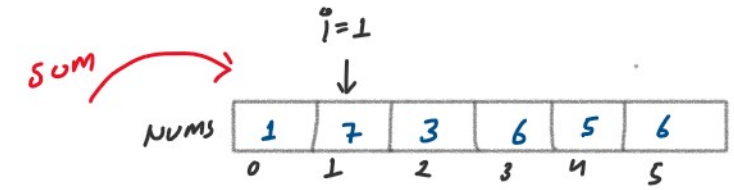
EX:01

APPROACH PSEUDO CODE:

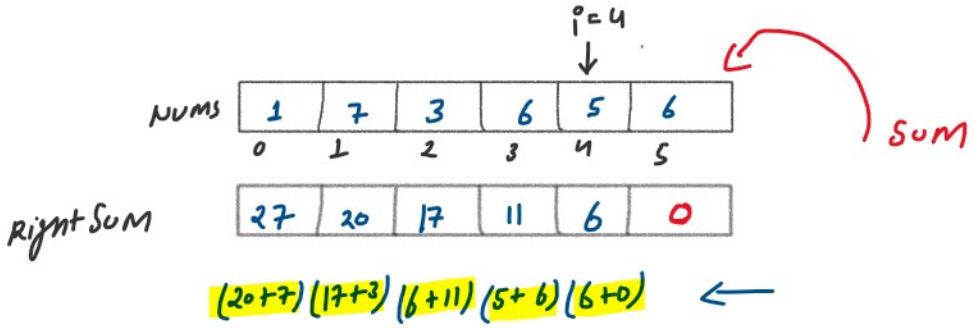
Ex:01

```
APPROACH PSEUDO CODE:
Step 01: create two vector rightSum and leftSum to prefix sum
Step 02: iterate nums till rightSum[i]==leftSum[i]
Step 03: return the index which terminate the loop
```

STEP:01

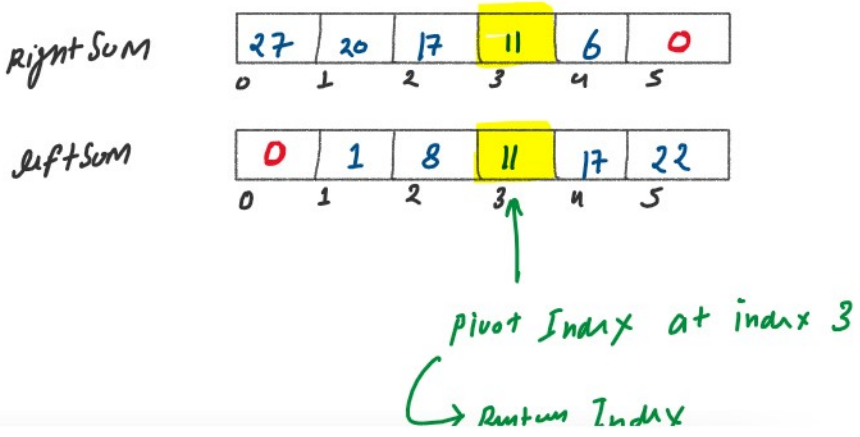



```
for(int i=1; i<n; i++){
    leftSum[i] = nums[i-1] + leftSum[i-1];
}
```




```
for(int i=n-2; i>=0; i--){
    rightSum[i] = nums[i+1] + rightSum[i+1];
}
```

STEP:02



↳ Return Index

```cpp
// Find Pivot Index with prefix sum approach (Leetcode-724)
class Solution {
public:
    int prefixSum(vector<int>& nums){
        int n=nums.size();
        vector<int> rightSum(n,0);
        vector<int> leftSum(n,0);

        // leftSum
        for(int i=1;i<n;i++){
            leftSum[i] = nums[i-1] + leftSum[i-1];
        }

        // rightSum
        for(int i=n-2;i>=0;i--){
            rightSum[i] = nums[i+1] + rightSum[i+1];
        }

        // Compare rightSum and leftSum to find pivot
        for(int i=0;i<n;i++){
            if(leftSum[i]==rightSum[i]){
                return i;
            }
        }

        return -1;
    }

    // Find Pivot Index
    int pivotIndex(vector<int>& nums) {
        int ans1 = prefixSum(nums);
        return ans1;
    }
};


/*
TIME COMPLEXITY: O(N), Where N is length of nums
SPACE COMPLEXITY: O(N), Where N is length of rightSum and leftSum
*/

/*
Example 1:
Input: nums = [1,7,3,6,5,6]
Output: 3
Explanation:
The pivot index is 3.
Left sum = nums[0] + nums[1] + nums[2] = 1 + 7 + 3 = 11
Right sum = nums[4] + nums[5] = 5 + 6 = 11

Example 2:
Input: nums = [1,2,3]
Output: -1
Explanation:
There is no index that satisfies the conditions in the problem statement.
*/
```

14. Missing Number (Leetcode-268)

Input { nums | 1 | 7 | 3 | 2 | 5 | 6 | 0 | (0 1 2 3 4 5 6)

Given Range [0,7] N = 7

output { missing number = 4

Approach: 1 step:01 Sum of Nums = 24

step:02 Sum of Range = $n*(\frac{n+1}{2})$ = 28

Approach : 1

Step:01 Sum of Nums = 24

Step:02 Sum of Range = $n*(\frac{n+1}{2})$ = 28

Step=03 missing NO = step2 - step1 = 28 - 24 = 4

O/P

```cpp
/*
Approach 01:
Step 01: Sum of nums
Step 02: Sum of range [0,N]
Step 03: Find missingNum = Step2 - Step1

Time Complexity: O(N), where N is length of nums
Space Complexity: O(1), no extra space used
*/
int totalSum(vector<int>& nums){
    int n=nums.size();

    // Step 01: Sum of nums
    int sum = 0;
    for(int i=0;i<n;i++){
        sum+=nums[i];
    }

    // Step 02: Sum of range [0,N]
    int totalSum=(n*(n+1))/2;

    // Step 03: Find missingNum = Step2 - Step1
    int missingNum=totalSum-sum;

    return missingNum;
}
```

Approach:02 XOR

Input { nums [1 7 3 2 5 6 0] (0 1 2 3 4 5 6)

Given Range [0, 7] N=7

Output { missing number = 4

```
Approach 02:
Step 01: XOR all nums[i] values
Step 02: XOR all range items
Step 03: return remaining range items
```

A ^ 0 = A
A ^ A = 0 Rules

ANS = 0

STEP:01 ANS ^ nums[i] ⇒ 0 ^ 1 ^ 7 ^ 3 ^ 2 ^ 5 ^ 6 ^ 0

Step:02 ANS ^ i ⇒ 0 ^ 1 ^ 2 ^ 3 ^ 4 ^ 5 ^ 6 ^ 7

⇒ 0 ^ 4

⇒ 4 → O/P

```cpp
/*
Approach 02:
```

=> 4 → o/p

```cpp
/*
Approach 02:
Step 01: XOR all nums[i] values
Step 02: XOR all range items
Step 03: return remaining range items

Time Complexity: O(N), where N is length of nums
Space Complexity: O(1), no extra space used
*/
int xorApproach(vector<int>& nums){
    int n=nums.size();
    int ans = 0;

    // Step 01: XOR all nums[i] values
    for(int i=0; i<n; i++){
        ans = ans ^ nums[i];
    }

    // Step 02: XOR all range items
    for(int i=0; i<=n; i++){
        ans = ans ^ i;
    }

    // Step 03: return remaining range items
    return ans;
}
```

Approach :03 SORTING

Input { nums | 1 | 7 | 3 | 2 | 5 | 6 | 0 | (0 1 2 3 4 5 6)

Given Range [0, 7] N=7

output { missing number = 4

{ Approach 03:
Step 01: Sorting nums
Step 02: Compare nums[i] with indeces
Step 03: Return not matched index or nums size }

STEP:01 SORT

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

STEP:02 i=0 nums[i] == i

0 0 == 0

STEP:02

i = 0	num[i] == i
0	0 == 0, i++
1	1 == 1, i++
2	2 == 2, i++
3	3 == 3, i++
4	5 != 4

Run jaa → out

Return 4 index

Output = 4

EX: 2

nums | 0 | 1 |

N = 2

Range ∈ [0,2]

missing No = 2

STEP:01

0	1
0	1

STEP:02

i = 0	num[i] == i
0	0 == 0, i++
1	1 == 1, i++
2	

Outside of loop

Return nums.size() = 2

TO ANS KYA HOGA?

```
/*
Approach 03:
Step 01: Sorting nums
Step 02: Compare nums[i] with indeces
Step 03: Return not matched index or nums size in the case of example [0,1] missing num=2

Time Complexity: O(NlogN), where N is length of nums
Space Complexity: O(1), no extra space used
*/
int sortingApproach(vector<int>& nums){
    int n=nums.size();

    // Step 01: Sorting nums
    sort(nums.begin(), nums.end());

    // Step 02: Compare nums[i] with indeces
    for(int i=0;i<n;i++){
        if(nums[i]==i){
            continue;
        }
        else{
            // Step 03: Return not matched index
            return i;
        }
    }
```

```
            else{
                // Step 03: Return not matched index
                return i;
            }
        }

        // Step 03: Return numsSize N when all index matched with nums[i]
        return n;
    }
```