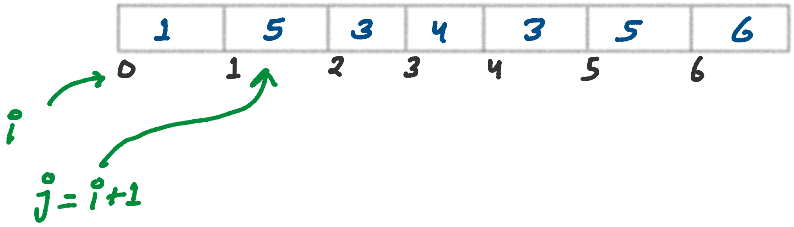
20 September 2023 15:07

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;
```

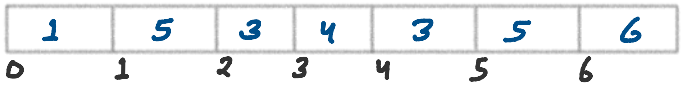
⇒ O(N*N)
⇒ O(N²)

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

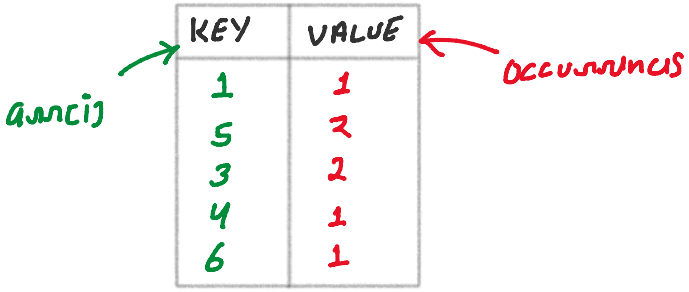
=> O(N^2)

T.C => O(N^2)

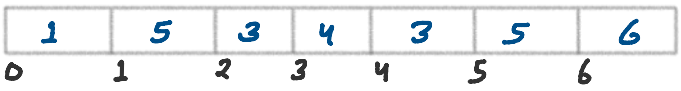
Optimal approach with hashing technique

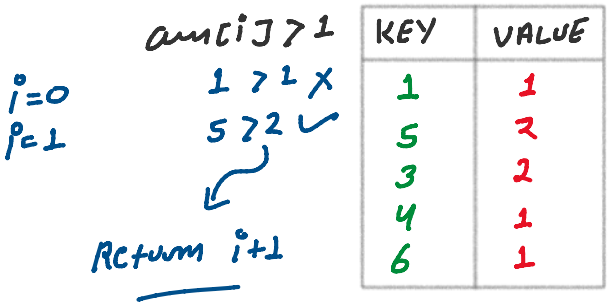


Step 01: *Traverse array and store it's element as hashing*



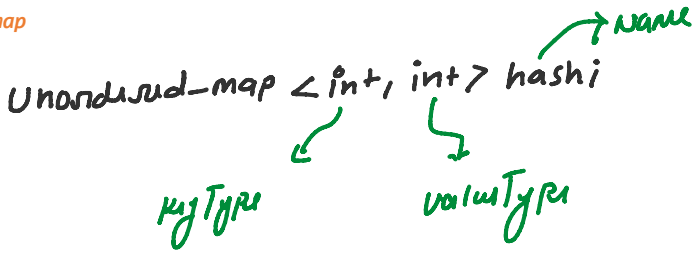
Step 02: *Traverse array to check each element if it has occurrence in future*





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)

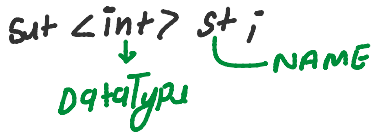
6. Common Element in 3 Sorted Array (GFG)

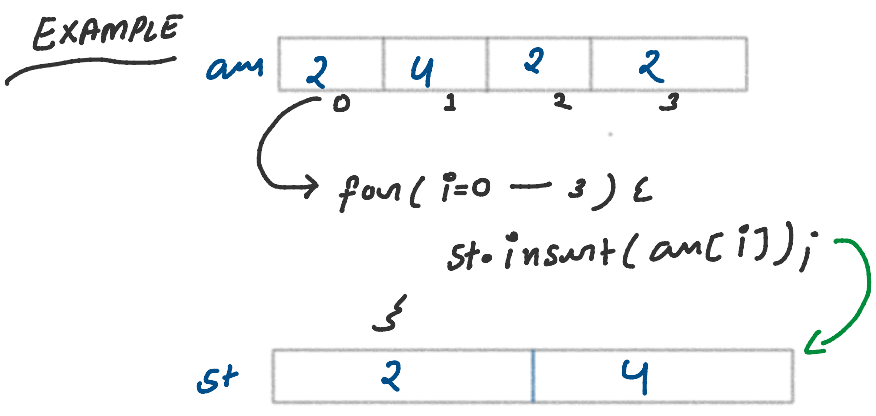
Example 01:	Example 02:
Input:	Input:
n1 = 6; A = {1, 5, 10, 20, 40, 80}	n1 = 3; A = {3, 3, 3}
n2 = 5; B = {6, 7, 20, 80, 100}	n2 = 3; B = {3, 3, 3}
n3 = 8; C = {3, 4, 15, 20, 30, 70, 80, 120}	n3 = 3; C = {3, 3, 3}
Output: 20 80	Output: 3

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

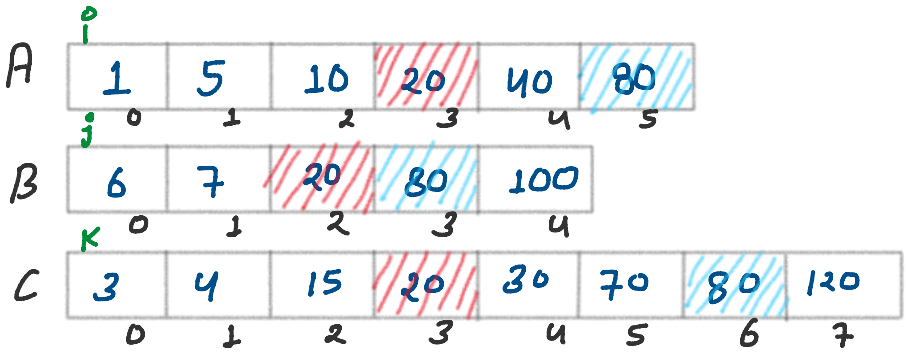




Read more from this resource: https://www.geeksforgeeks.org/set-in-cpp-stl/

RY 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])
{
        i++;
}
else if (B[j]<C[k])
{
        j++;
}
else
{
        k++;
}
```

Terminating Condition

→ (i<n1 && j<n2 && k<n3)

```
// 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])
```

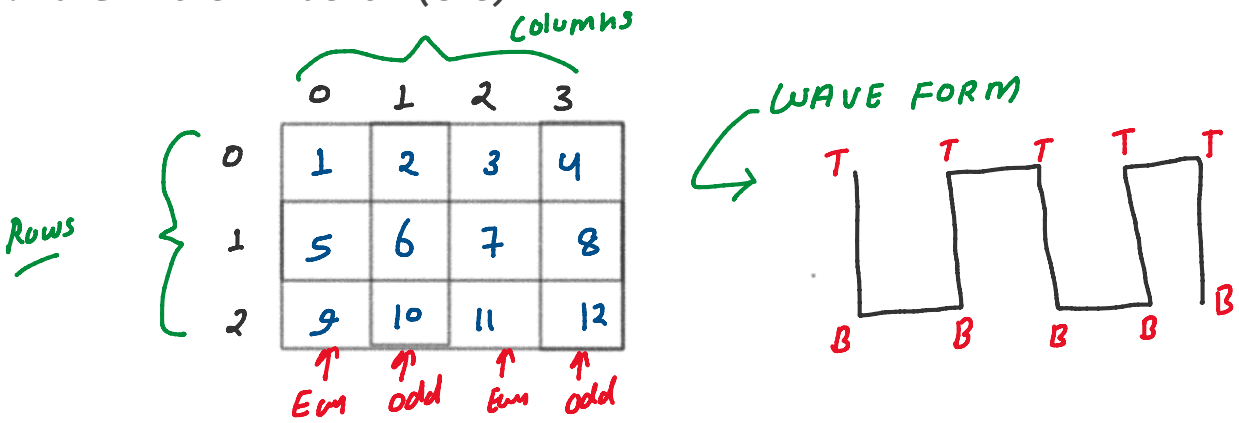
```
                else if(A[i]<B[j])
                {
                    i++;
                }
                else if(B[j]<C[k])
                {
                    j++;
                }
                else
                {
                    k++;
                }
            }

            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***

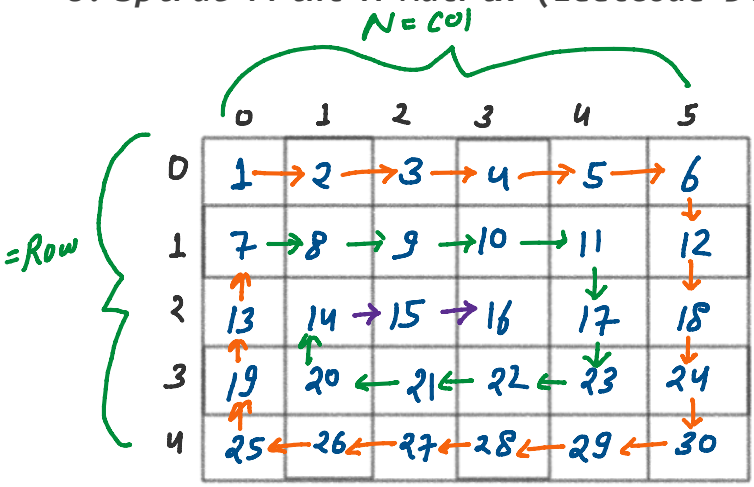
```
// Print wave matrix
void printWaveMatrix(vector<vector<int>> matrix){
    int row = matrix.size();
    int col = matrix[0].size();

    // 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
*/
```