* **Experiment-2: Arrays**

**Code-1:**

#include <bits/stdc++.h>

using namespace std;

// } Driver Code Ends

// User function template for C++

class Solution {

public:

// Function returns the second

// largest elements

int print2largest(vector<int> &arr) {

int max = arr[0];

int n = arr.size();

int smax = -1;

for(int i = 1; i<n; i++)

{

if(arr[i]>max){

smax = max;

max = arr[i];

}

else if(arr[i]<max && arr[i]>smax){

smax = arr[i];

}

}

return smax;

}

};

//{ Driver Code Starts.

int main() {

int t;

cin >> t;

cin.ignore();

while (t--) {

vector<int> arr;

string input;

getline(cin, input);

stringstream ss(input);

int number;

while (ss >> number) {

arr.push\_back(number);

}

Solution ob;

int ans = ob.print2largest(arr);

cout << ans << endl;

}

return 0;

}

**Code-2:**

#include <bits/stdc++.h>

using namespace std;

// } Driver Code Ends

// User function template for C++

class Solution {

public:

// Function returns the second

// largest elements

int print2largest(vector<int> &arr) {

int max = arr[0];

int n = arr.size();

int smax = -1;

for(int i = 1; i<n; i++)

{

if(arr[i]>max){

smax = max;

max = arr[i];

}

else if(arr[i]<max && arr[i]>smax){

smax = arr[i];

}

}

return smax;

}

};

//{ Driver Code Starts.

int main() {

int t;

cin >> t;

cin.ignore();

while (t--) {

vector<int> arr;

string input;

getline(cin, input);

stringstream ss(input);

int number;

while (ss >> number) {

arr.push\_back(number);

}

Solution ob;

int ans = ob.print2largest(arr);

cout << ans << endl;

}

return 0;

}

**Code-3:**

class Solution {

public:

    bool check(vector<int>& nums) {

        int n = nums.size();

        for(int i=1;i<n;i++){

            if(nums[i]>=nums[i-1]){

            }

            else{

                return false;

            }

        }

        return true;

    }

};

**Code-4:**

lass Solution {

public:

    int removeDuplicates(vector<int>& nums) {

       int n = nums.size();

       int i = 0;

       for(int j = 1; j<n;j++){

        if(nums[j]!=nums[i]){

            nums[i+1]=nums[j];

            i++;

        }

       }

       return (i+1);

    }

};

**Code-5:**

class Solution {

public:

    void rotate(vector<int>& nums, int k) {

       int n = nums.size();

        int temp = nums[0];

        for(int i=1;i<n;i++){

            nums[i-1]=nums[i];

        }

        nums[n-1]=temp;

    }

};

**Code-6:**

class Solution {

public:

    void rotate(vector<int>& nums, int k) {

       int n = nums.size();

        k = k % n;  // Handle cases where k >= n

        reverse(nums.begin(), nums.end());

        reverse(nums.begin(), nums.begin() + k);

        reverse(nums.begin() + k, nums.end());

    }

};

**Code-7:**

class Solution {

public:

    void moveZeroes(vector<int>& nums) {

        int j = -1;

        int n = nums.size();

        for(int i=0;i<n;i++){

            if(nums[i]==0){

                j=i;

                break;

            }

        }

        for(int i=j+1;i<n;i++){

            if(nums[i]!=0){

            swap(nums[i],nums[j]);

            j++;

            }

        }

    }

};

**Code-8:**

class Solution {

public:

    void moveZeroes(vector<int>& nums) {

        int j = -1;

        int n = nums.size();

        for(int i=0;i<n;i++){

            if(nums[i]==0){

                j=i;

                break;

            }

        }

        for(int i=j+1;i<n;i++){

            if(nums[i]!=0){

            swap(nums[i],nums[j]);

            j++;

            }

        }

    }

};

**Code-9:**

//{ Driver Code Starts

#include <bits/stdc++.h>

using namespace std;

// } Driver Code Ends

class Solution{

public:

//arr1,arr2 : the arrays

// n, m: size of arrays

//Function to return a list containing the union of the two arrays.

vector<int> findUnion(int arr1[], int arr2[], int n, int m)

{

int i=0;

int j=0;

vector<int> unionArr;

while(i<n && j<m){

if(arr1[i]<=arr2[j]){

if(unionArr.size()==0 || unionArr.push\_back!=arr1[i]){

unionArr.push\_back(arr1[i]);

}

i++;

}

else{

if(arr1[i]<=arr2[j]){

if(unionArr.size()==0 || unionArr.push\_back!=arr2[j]){

unionArr.push\_back(arr2[j]);

}

j++;

}

}

while(j<m){

if(arr1[i]<=arr2[j]){

if(unionArr.size()==0 || unionArr.push\_back!=arr2[j]){

unionArr.push\_back(arr2[j]);

}

j++;

}

}

while(i<n){

if(arr1[i]<=arr2[j]){

if(unionArr.size()==0 || unionArr.push\_back!=arr1[i]){

unionArr.push\_back(arr1[i]);

}

i++;

}

}

}

}

}

};

//{ Driver Code Starts.

int main() {

int T;

cin >> T;

while(T--){

int N, M;

cin >>N >> M;

int arr1[N];

int arr2[M];

for(int i = 0;i<N;i++){

cin >> arr1[i];

}

for(int i = 0;i<M;i++){

cin >> arr2[i];

}

Solution ob;

vector<int> ans = ob.findUnion(arr1,arr2, N, M);

for(int i: ans)cout<<i<<' ';

cout << endl;

}

return 0;

}

// } Driver Code Ends

**Code-10:**

class Solution {

public:

    int missingNumber(vector<int>& nums) {

       int xor1=0;

       int xor2=0;

       int n = nums.size();

       for(int i=0;i<n;i++){

        xor2 = xor2^nums[i];

        xor1 = xor1^(i+1);

       }

        xor1 = xor1^n;

        return xor1^xor2;

    }

};

**Code-11:**

class Solution {

public:

    int findMaxConsecutiveOnes(vector<int>& nums) {

        int max1 = 0;

        int cnt = 0;

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

            if(nums[i]==1){

                cnt++;

                max1=max(max1,cnt);

            }

            else{

                cnt = 0;

            }

        }

        return max1;

    }

};

**Code-12:**

class Solution {

public:

    int singleNumber(vector<int>& nums) {

        int xorr = 0;

        int n=nums.size();

        for(int i=0;i<n;i++){

            xorr = xorr^nums[i];

        }

        return xorr;

    }

};

**Code-13:**

#include <bits/stdc++.h>

using namespace std;

int getLongestSubarray(vector<int>& a, long long k) {

int n = a.size(); *// size of the array.*

int left = 0, right = 0; *// 2 pointers*

long long sum = a[0];

int maxLen = 0;

while (right < n) {

*// if sum > k, reduce the subarray from left*

*// until sum becomes less or equal to k:*

while (left <= right && sum > k) {

sum -= a[left];

left++;

}

*// if sum = k, update the maxLen i.e. answer:*

if (sum == k) {

maxLen = max(maxLen, right - left + 1);

}

*// Move forward thw right pointer:*

right++;

if (right < n) sum += a[right];

}

return maxLen;

}

int main()

{

vector<int> a = {2, 3, 5, 1, 9};

long long k = 10;

int len = getLongestSubarray(a, k);

cout << "The length of the longest subarray is: " << len << "\n";

return 0;

}

**Code-14:**

#include <bits/stdc++.h>

using namespace std;

int getLongestSubarray(vector<int>& a, int k) {

int n = a.size(); // size of the array.

map<int, int> preSumMap;

int sum = 0;

int maxLen = 0;

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

//calculate the prefix sum till index i:

sum += a[i];

// if the sum = k, update the maxLen:

if (sum == k) {

maxLen = max(maxLen, i + 1);

}

// calculate the sum of remaining part i.e. x-k:

int rem = sum - k;

//Calculate the length and update maxLen:

if (preSumMap.find(rem) != preSumMap.end()) {

int len = i - preSumMap[rem];

maxLen = max(maxLen, len);

}

//Finally, update the map checking the conditions:

if (preSumMap.find(sum) == preSumMap.end()) {

preSumMap[sum] = i;

}

}

return maxLen;

}

int main()

{

vector<int> a = { -1, 1, 1};

int k = 1;

int len = getLongestSubarray(a, k);

cout << "The length of the longest subarray is: " << len << "\n";

return 0;

}

**Code-15:** *2 sum problem*

class Solution {

public:

    vector<int> twoSum(vector<int>& nums, int target) {

        unordered\_map<int, int> mpp;

        int n=nums.size();

    for (int i = 0; i < n; i++) {

        int arr=nums[i];

        int moreNeeded = target - arr;

        if (mpp.find(moreNeeded) != mpp.end()) {

            return {mpp[moreNeeded], i};

        }

        mpp[arr] = i;

    }

    return { -1, -1};

}

};

Code-16*: Sorting of array 0’s,1’s,2’s*

**s**