**############################################################**

**1- Check if two arrays are equal or not**

Given two arrays **A** and **B** of equal size **N**, the task is to find if given arrays are equal or not. Two arrays are said to be equal if both of them contain same set of elements, arrangements (or permutation) of elements may be different though.  
**Note :** If there are repetitions, then counts of repeated elements must also be same for two array to be equal.

**Example 1:**

**Input:**

N = 5

A[] = {1,2,5,4,0}

B[] = {2,4,5,0,1}

**Output:** 1

**Explanation:** Both the array can be

rearranged to {0,1,2,4,5}

**Example 2:**

**Input:**

N = 3

A[] = {1,2,5}

B[] = {2,4,15}

**Output:** 0

**Explanation:** A[] and B[] have only

one common value.

**Your Task:**  
Complete **check()**function which takes both the given array and their size as function arguments and returns **true** if the arrays are equal else **returns false**.The 0 and 1 printing is done by the driver code.

**Expected Time Complexity**: O(N)  
**Expected Auxilliary Space** : O(N)

**Constraints:**  
1<=N<=107  
1<=A[],B[]<=1018

bool check(vector<ll> a, vector<ll> b, int n) {

unordered\_map<ll,ll> mp;

for(auto i : a)

mp[i]++;

for(auto i : b)

mp[i]--;

for(auto i: mp)

if(i.second != 0)

return 0;

return 1;

}

**############################################################**

**2- Max distance between same elements**

Given an array with repeated elements, the task is to find the maximum distance between two occurrences of an element.

**Example 1:**

**Input**

n= 6

arr = {1, 1, 2, 2, 2, 1}

**Output**

5

**Explanation**

arr[] = {1, 1, 2, 2, 2, 1}

Max Distance: 5

Distance for 1 is: 5-0 = 5

Distance for 2 is : 4-2 = 2

Max Distance 5

**Example 2:**

**Input**

n = 12

arr = {3, 2, 1, 2, 1, 4, 5, 8, 6, 7, 4, 2}

**Output**

10

**Explanation**

arr[] = {3, 2, 1, 2, 1, 4, 5, 8, 6, 7, 4, 2}

Max Distance 10

maximum distance for 2 is 11-1 = 10

maximum distance for 1 is 4-2 = 2

maximum distance for 4 is 10-5 = 5

**Your Task:**  
Complete **maxDistance()**function which takes both the given array and their size as function arguments and returns the maximum distance between 2 same elemenrs.

**Expected Time Complexity**: O(N)  
**Expected Auxilliary Space** : O(N)

**Constraints:**

1<=N<=106

int maxDistance(int arr[], int n)

{

unordered\_map<int, vector<int>> mp;

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

mp[arr[i]].push\_back(i);

int ans = 0;

for(auto i : mp)

{ vector<int> temp(i.second);

ans = max(ans, temp[temp.size()-1] - temp[0]);

}

return ans;

}

**############################################################**

**3- Two Sum**

Given an array of positive integers and an integer. Determine whether or not there exist two elements in A whose sum is exactly equal to that integer.

**Example 1:**

**Input:**

N = 6, X = 16

A[] = {1,4,45,6,10,8}

**Output:** Yes

**Explanation:** 10 and 6 are numbers

making a pair whose sum is equal to 16.

**Example 2:**

**Input:**

N = 5, X = 10

A[] = {1,2,4,3,6}

**Output:** Yes

**Your Task :**  
You don't need to read input or print anything. Your task is to complete the boolean function **keypair()** which takes the array A[], the size of the array (N) and another number (X) as inputs and returns true if there exists a pair in A[] that sums up to X and returns false otherwise. (THe driver code will print "Yes" if the returned values is true, otherwise "No")

**Expected Time Complexity:** O(N).  
**Expected Auxiliary Space:** O(N).

**Constraints:**  
1 ≤ N ≤ 105  
1 ≤ A[i] ≤ 105  
1 ≤ X ≤ 2\*105

bool keypair(vector<int> a, int n, int x)

{

unordered\_map<int, int> mp;

int sum = 0;

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

if(mp.find(x - a[i]) != mp.end())

return true;

else

mp[a[i]] = x - a[i];

return false;

}

**############################################################**

**4- Largest subarray with 0 sum**

Given an array having both positive and negative integers. The task is to compute the length of the largest subarray with sum 0.

**Example 1:**

**Input:**

N = 8

A[] = {15,-2,2,-8,1,7,10,23}

**Output:** 5

**Explanation:** The largest subarray with

sum 0 will be -2 2 -8 1 7.

**Your Task:**  
You just have to complete the function **maxLen()**whichtakes two arguments an array**A** and**n,** where n is the size of the array A and returns the length of the largest subarray with 0 sum.

**Expected Time Complexity:**O(N\*Log(N)).  
**Expected Auxiliary Space:**O(N).

**Constraints:**  
1 <= N <= 105  
-1000 <= A[i] <= 1000, for each valid i

int maxLen(int arr[], int n)

{

unordered\_map<int, int> mp;

int ans = 0, sum = 0;

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

{ sum += arr[i];

if(sum == 0)

ans = i + 1;

if(mp.find(sum) != mp.end())

ans = max(ans, i - mp[sum]);

if(mp.find(sum) == mp.end())

mp[sum] = i;

}

return ans;

}

**############################################################**

**5- Print Non-Repeated Elements**

*Hashing is very useful to keep track of the frequency of the elements in a list.*

You are given an array of integers. You need to print the non-repeated elements as they appear in the array.

**Example 1:**

**Input:**

n = 10

arr[] = {1,1,2,2,3,3,4,5,6,7}

**Output:** 4 5 6 7

**Explanation:** 4, 5, 6 and 7 are the only

elements which is having only 1

frequency and hence, Non-repeating.

**Example 2:**

**Input:**

n = 5

arr[] = {10,20,40,30,10}

**Output:** 20 40 30

**Explanation:** 20, 40, 30 are the only

elements which is having only 1

frequency and hence, Non-repeating.

**Your Task:**  
You don't need to read input or print anything. You only need to complete the function **printNonRepeated()**that takes **arr and n as parameters**and return the array which has the distinct elements in same order as they appear in input array. The newline is **appended**automatically by the **driver code**.

**Expected Time Complexity:**O(n).  
**Expected Auxiliary Space:**O(n).

**Constraints:**  
1 <= n <= 103  
0 <= arri<= 107

vector<int> printNonRepeated(int arr[],int n)

{

unordered\_map<int, int> mp;

vector<int> ans;

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

mp[arr[i]]++;

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

if(mp[arr[i]] == 1)

ans.push\_back(arr[i]);

return ans;

}

**############################################################**

**6- Count distinct elements in every window**

Given an array of integers and a number K. Find the count of distinct elements in every window of size K in the array.

**Example 1:**

**Input:**

N = 7, K = 4

A[] = {1,2,1,3,4,2,3}

**Output:** 3 4 4 3

**Explanation:** Window 1 of size k = 4 is

1 2 1 3. Number of distinct elements in

this window are 3.

Window 2 of size k = 4 is 2 1 3 4. Number

of distinct elements in this window are 4.

Window 3 of size k = 4 is 1 3 4 2. Number

of distinct elements in this window are 4.

Window 4 of size k = 4 is 3 4 2 3. Number

of distinct elements in this window are 3.

**Example 2:**

**Input:**

N = 3, K = 2

A[] = {4,1,1}

**Output:** 2 1

**Your Task:**  
Your task is to complete the function **countDistinct()** which takes the array A[], the size of the array(N) and the window size(K) as inputs and returns an array containing the count of distinct elements in every contiguous window of size K in the array A[].

**Expected Time Complexity:** O(N).  
**Expected Auxiliary Space:** O(N).

**Constraints:**  
1 <= N <= K <= 105  
1 <= A[i] <= 105, for each valid i

vector <int> countDistinct (int arr[], int n, int k)

{

vector<int> ans;

unordered\_map<int, int> mp;

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

mp[arr[i]]++;

ans.push\_back(mp.size());

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

{ if(--mp[arr[i-k]] == 0)

mp.erase(arr[i-k]);

mp[arr[i]]++;

ans.push\_back(mp.size());

}

return ans;

}

**############################################################**

**7- Sorting Elements of an Array by Frequency**

Given an arrayof integers, **sort**the array according to **frequency**of elements. That is elements that have higher frequency come first. If frequencies of two elements are same, then smaller number comes first.

**Example 1:**

**Input:**

N = 5

A[] = {5,5,4,6,4}

**Output:** 4 4 5 5 6

**Explanation:** The highest frequency here is

2. Both 5 and 4 have that frequency. Now

since the frequencies are same then

smallerelement comes first. So 4 4 comes

firstthen comes 5 5. Finally comes 6.

The output is **4 4 5 5 6.**

**Example 2:**

**Input:**

N = 5

A[] = {9,9,9,2,5}

**Output:** 9 9 9 2 5

**Explanation:** The highest frequency here is

3. The element 9 has the highest frequency

So 9 9 9 comes first. Now both 2 and 5

have same frequency. So we print smaller

element first.

The output is **9 9 9 2 5.**

**Your Task:**

You only need to complete the **function sortByFreq**that takes **arr, and n**as parameters and **returns**the sorted array.

**Expected Time Complexity:**O(NLogN).  
**Expected Auxiliary Space:**O(N).

**Constraints:**  
1 ≤ N ≤ 105  
1 ≤ Ai ≤ 105

static bool cmp(pair<int, int> a, pair<int, int> b)

{ if(a.second == b.second)

return a.first<b.first;

return a.second>b.second;

}

vector<int> sortByFreq(int arr[],int n)

{

unordered\_map<int, int> mp;

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

mp[arr[i]]++;

vector<pair<int, int>> v;

for(auto i:mp)

v.push\_back(i);

sort(v.begin(), v.end(), cmp);

vector<int> ans;

for(auto i:v)

while(i.second-- > 0)

ans.push\_back(i.first);

return ans;

}

**############################################################**

**8- Top K Frequent Elements in Array - |**

Given a non-empty array of integers, find the top k elements which have the highest frequency in the array. If two numbers have the same frequency then the larger number should be given preference.

**Example 1:**

**Input:**

nums = {1,1,1,2,2,3},

k = 2

**Output:** {1, 2}

**Example 2:**

**Input:**

nums = {1,1,2,2,3,3,3,4},

k = 2

**Output:** {3, 2}

**Explanation:** Elements 1 and 2 have the

same frequency ie. 2. Therefore, in this

case, the answer includes the element 2

as 2 > 1.

**User Task:**  
The task is to complete the function **topK()** that takes the array and integer k as input and returns a list of top k frequent elements.

**Expected Time Complexity** : O(NlogN)  
**Expected Auxilliary Space** : O(N)

**Constraints:**  
1 <= N <= 105  
1<=A[i]<=105

static bool cmp(pair<int, int> a, pair<int, int> b)

{

if ( a.second == b.second)

return a.first > b.first;

return a.second > b.second;

}

vector<int> topK(vector<int>& nums, int k)

{

unordered\_map<int,int> mp;

for(auto i : nums)

mp[i]++;

vector<pair<int,int>> v;

for(auto i : mp)

v.push\_back(i);

sort(v.begin(), v.end(), cmp);

vector<int> ans;

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

{

ans.push\_back(v[i].first);

}

return ans;

}

**############################################################**

**9- Zero Sum Subarrays**

You are given an array arr[] of size n. Find the total count of sub-arrays having their sum equal to 0.

**Example 1:**

**Input:**

n = 6

arr[] = {0,0,5,5,0,0}

**Output:** 6

**Explanation:** The 6 subarrays are

[0], [0], [0], [0], [0,0], and [0,0].

**Example 2:**

**Input:**

n = 10

arr[] = {6,-1,-3,4,-2,2,4,6,-12,-7}

**Output:** 4

**Explanation:** The 4 subarrays are [-1 -3 4]

[-2 2], [2 4 6 -12], and [-1 -3 4 -2 2]

**Your Task:**  
You don't need to read input or print anything. Complete thefunction **findSubarray()** that takes the array arr and its size n as input parametersand returns the total number of sub-arrays with 0 sum. 

**Expected Time Complexity** : O(n)  
**Expected Auxilliary Space** : O(n)

**Constraints:**  
1<= n <= 107  
-1010 <= arri <= 1010

ll findSubarray(vector<ll> arr, int n ) {

int sum = 0, ans = 0;

unordered\_map<ll, int> mp;

mp[0] = 1;

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

{ sum += arr[i];

if(mp.find(sum) != mp.end())

{ ans += mp[sum];

mp[sum]++;

}

else

mp[sum] = 1;

}

return ans;

}

**############################################################**

**10- Sort an array according to the other**

Given two integer arrays **A1[ ]**and **A2[ ]**of size **N** and **M** respectively. Sort the first array **A1[ ]**such that all the relative positions of the elements in the first array are the same as the elements in the second array **A2[ ]**.  
See example for better understanding.  
**Note**: If elements are repeated in the second array, consider their first occurance only.

**Example 1:**

**Input:**

N = 11

M = 4

A1[] = {2, 1, 2, 5, 7, 1, 9, 3, 6, 8, 8}

A2[] = {2, 1, 8, 3}

**Output:**

2 2 1 1 8 8 3 5 6 7 9

**Explanation:** Array elements of A1[] are

sorted according to A2[]. So 2 comes first

then 1 comes, then comes 8, then finally 3

comes, now we append remaining elements in

sorted order.

**Example 2:**

**Input:**

N = 11

M = 4

A1[] = {2, 1, 2, 5, 7, 1, 9, 3, 6, 8, 8}

A2[] = {99, 22, 444, 56}

**Output:**

1 1 2 2 3 5 6 7 8 8 9

**Explanation:** No A1[] elements are in A2[]

so we cannot sort A1[] according to A2[].

Hence we sort the elements in non-decreasing

order.

**Your Task:**  
You don't need to read input or print anything. Your task is to complete the function **sortA1ByA2()** which takes the array **A1[ ]**, array **A2[ ]** and their respective size **N** and **M** as input parameters and returns the sorted array **A1[ ]**such that the relative positions of the elements in **A1[ ]** are same as the elements in **A2[ ]**. For the elements not present in **A2[ ]** but in **A1[ ]**, it appends them at the last in **increasing**order.

**Expected Time Complexity:** O(N \* Log(N)).  
**Expected Auxiliary Space:** O(N).

**Constraints:**  
1 ≤ N, M ≤ 106  
1 ≤ A1[i], A2[i] ≤ 106

static bool cmp(pair<int, int> a, pair<int, int> b)

{

return a.first < b.first;

}

vector<int> sortA1ByA2(vector<int> a1, int n, vector<int> a2, int m)

{

unordered\_map<int, int> mp;

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

mp[a1[i]]++;

vector<int> ans;

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

{

if(mp.find(a2[i]) != mp.end())

{

while(mp[a2[i]]-- > 0)

ans.push\_back(a2[i]);

if(mp[a2[i]] == 0)

mp.erase(a2[i]);

}

}

vector<pair<int, int>> v;

for(auto i: mp)

v.push\_back(i);

sort(v.begin(), v.end(), cmp);

for(auto i : v)

{

while(i.second-- > 0)

ans.push\_back(i.first);

}

return ans;

}

**############################################################**

**11- Find All Four Sum Numbers**

Given an array of integers and another number. Find all the **unique**quadruple from the given array that sums up to the given number.

**Example 1:**

**Input:**

N = 5, K = 3

A[] = {0,0,2,1,1}

**Output:** 0 0 1 2 $

**Explanation:** Sum of 0, 0, 1, 2 is equal

to K.

**Example 2:**

**Input:**

N = 7, K = 23

A[] = {10,2,3,4,5,7,8}

**Output:** 2 3 8 10 $2 4 7 10 $3 5 7 8 $

**Explanation:** Sum of 2, 3, 8, 10 = 23,

sum of 2, 4, 7, 10 = 23 and sum of 3,

5, 7, 8 = 23.

**Your Task:**  
You don't need to read input or print anything. Your task is to complete the function **fourSum()** which takes the array arr[] and the integer k as its input and returns an array containing all the quadruples in a lexicographical manner. Also note that all the quadruples should be internally sorted, ie for any quadruple [q1, q2, q3, q4] the following should follow: q1 <= q2 <= q3 <= q4.  (In the output each quadruple is separate by $. The printing is done by the driver's code)

**Expected Time Complexity:** O(N3).  
**Expected Auxiliary Space:** O(N2).

**Constraints:**  
1 <= N <= 100  
-1000 <= K <= 1000  
-100 <= A[] <= 100

vector<vector<int> > fourSum(vector<int> &arr, int k)

{

set<vector<int>> num;

sort(arr.begin(), arr.end());

int sum = 0;

int n = arr.size();

for(int i = 0; i < n-3; i++)

{

for(int j = i+1; j < n-2; j++)

{

int l = j+1;

int r = n-1;

while(l < r)

{

sum = arr[i] + arr[j] + arr[l] + arr[r];

if(sum == k)

{

vector<int> t;

t.push\_back(arr[i]);

t.push\_back(arr[j]);

t.push\_back(arr[l]);

t.push\_back(arr[r]);

num.insert(t);

l = l + 1;

}

else if(sum > k)

r = r - 1;

else

l = l + 1;

}

}

}

vector<vector<int>> ans;

for(auto i:num)

ans.push\_back(i);

return ans;

}