**1. Relative Sorting**

Given two arrays **A1[]** and **A2[]** of size **N** and **M** respectively. The task is to sort A1 in such a way that the relative order among the elements will be same as those in A2. For the elements not present in A2, append them at last in sorted order. It is also given that the number of elements in A2[] are smaller than or equal to number of elements in A1[] and A2[] has all distinct elements.  
**Note:** Expected time complexity is **O(N log(N))**.

**Input:**  
First line of input contains number of testcases. For each testcase, first line of input contains length of arrays N and M and next two line contains N and M elements respectively.

**Output:**  
Print the relatively sorted array.

**Constraints:**  
1 ≤ T ≤ 100  
1 ≤ N,M  ≤ 106  
1 ≤ A1[], A2[] <= 106

**Example:**  
**Input:**  
2  
11 4  
2 1 2 5 7 1 9 3 6 8 8  
2 1 8 3  
8 4  
2 6 7 5 2 6 8 4   
2 6 4 5  
**Output:**  
2 2 1 1 8 8 3 5 6 7 9  
2 2 6 6 4 5 7 8

CODE:

using namespace std;

int main()

{

int t;

cin>>t;

while(t--)

{

int n,m;

cin>>n>>m;

int arr1[n];

int arr2[m];

map<int,int> mp;

map<int,int> no;

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

{

cin>>arr1[i];

mp[arr1[i]]++;

}

vector<int> v;

vector<int> ans;

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

{

cin>>arr2[i];

no[arr2[i]]++;

for(int j=0;j<mp[arr2[i]];j++)

v.push\_back(arr2[i]);

}

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

{

if(no[arr1[i]]==0)

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

}

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

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

{

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

}

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

cout<<v[i]<<" ";

cout<<endl;

}

}

**2. Sorting Elements of an Array by Frequency**

Given an array**A[]**of 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.

**Input:**  
The first line of input contains an integer T denoting the number of test cases. The description of T test cases follows. The first line of each test case contains a single integer N denoting the size of array. The second line contains N space-separated integers A1, A2, ..., AN denoting the elements of the array.

**Output:**  
For each testcase, in a new line, print each sorted array in a seperate line. For each array its numbers should be seperated by space.

**Constraints:**  
1 ≤ T ≤ 70  
30 ≤ N ≤ 130  
1 ≤ Ai ≤ 60

**Example:**  
**Input:**  
2  
5  
5 5 4 6 4  
5  
9 9 9 2 5

**Output:**  
4 4 5 5 6  
9 9 9 2 5

**Explanation:**  
**Testcase1:** The highest frequency here is 2. Both 5 and 4 have that frequency. Now since the frequencies are same then smaller element comes first. So 4 4 comes first then comes 5 5. Finally comes 6.  
The output is **4 4 5 5 6.**

**Testcase2:** 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.**

CODE:

using namespace std;

bool compare(pair<int,int> p1,pair<int,int> p2)

{

if(p1.second==p2.second)

return p1.first<p2.first;

return p1.second>p2.second;

}

int main()

{

int t;

cin>>t;

while(t--)

{

int n;

cin>>n;

int arr[n];

map<int,int> mp;

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

{

cin>>arr[i];

mp[arr[i]]++;

}

vector<pair<int,int>> v;

map<int,int> ::iterator it=mp.begin();

while(it!=mp.end())

{

v.push\_back(make\_pair(it->first,it->second));

it++;

}

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

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

for(int j=0;j<v[i].second;j++)

cout<<v[i].first<<" ";

cout<<endl;

}

return 0;

}

**3. 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 <= 104  
-1000 <= A[i] <= 1000, for each valid i

CODE:

#include <bits/stdc++.h>

using namespace std;

int maxLen(int A[], int n);

int main()

{

int T;

cin >> T;

while (T--)

{

int N;

cin >> N;

int A[N];

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

cin >> A[i];

cout << maxLen(A, N) << endl;

}

}

// } Driver Code Ends

/\*You are required to complete this function\*/

int maxLen(int arr[], int n)

{

int sum=0;

int maxi=0;

unordered\_map<int,int>mp;

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

{

sum+=arr[i];

if(sum==0)

{

maxi=max(maxi,i+1);

}

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

{

int len = (i-mp[sum]);

maxi=max(maxi,len);

}

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

mp[sum]=i;

}

return maxi;

}

**4. Common elements**

Given three arrays sorted in increasing order. Find the elements that are common in all three arrays.  
**Note:**can you take care of the duplicates without using any additional Data Structure?

**Example 1:**

**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

**Explanation:** 20 and 80 are the only

common elements in A, B and C.

**Your Task:**  
You don't need to read input or print anything. Your task is to complete the function **commonElements()**which take the 3 arrays A[], B[], C[] and their respective sizes n1, n2 and n3 as inputs and returns an array containing the common element present in all the 3 arrays in sorted order.   
If there are no such elements return an empty array. In this case the output will be printed as -1.

**Expected Time Complexity:**O(n1 + n2 + n3)  
**Expected Auxiliary Space:** O(n1 + n2 + n3)

**Constraints:**  
1 <= n1, n2, n3 <= 10^5  
The array elements can be both positive or negative integers.

CODE:

using namespace std;

int main()

{

int t;

cin>>t;

while(t--)

{

int n,m,p;

cin>>n>>m>>p;

map<int,int> mp;

int flag=0;

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

{

int x;

cin>>x;

mp[x]++;

}

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

{

int x;

cin>>x;

if(mp[x]>0)

mp[x]=-1;

}

set<int> s;

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

{

int x;

cin>>x;

if(mp[x]==-1)

{

flag=1;

s.insert(x);

}

}

if(flag)

{

set<int>::iterator it=s.begin();

while(it!=s.end())

{

cout<<\*it<<" ";

it++;

}

}

if(!flag)

cout<<"-1";

cout<<endl;

}

return 0;

}

**5. Find all four sum numbers**

Given an array A of size N, find all combination of four elements in the array whose sum is equal to a given value K. For example, if the given array is {10, 2, 3, 4, 5, 9, 7, 8} and K = 23, one of the quadruple is “3 5 7 8” (3 + 5 + 7 + 8 = 23).

The output should contain only unique quadrples  For example, if input array is {1, 1, 1, 1, 1, 1} and K = 4, then output should be only one quadrple {1, 1, 1, 1}.

**Note:** Print -1 if no such quadruple is found. 

**Input:**  
The first line of input contains an integer T denoting the no of test cases. Then T test cases follow. Each test case contains two lines. The first line of input contains two integers N and K. Then in the next line are N space separated values of the array.  
  
**Output:**  
For each test case in a new line print all the quadruples present in the array separated by space which sums up to value of K. Each quadruple is unique which are separated by a delimeter "$" and are in increasing order.  
  
**Constraints:**  
1 <= T <= 100  
1 <= N <= 1000  
-1000 <= K <= 1000  
-100 <= A[] <= 100  
  
**Example:  
Input:**  
2  
5 3  
0 0 2 1 1   
7 23  
10 2 3 4 5 7 8

**Output:**  
0 0 1 2 $  
2 3 8 10 $2 4 7 10 $3 5 7 8 $

CODE:

using namespace std;

void func(vector<int> v, int n, int k) {

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

int i = 0, j = 0, sum = 0;

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

for (j = i + 1; j < n; j++) {

mp[v[i] + v[j]].push\_back(make\_pair(i, j));

}

}

set<vector<int>> s;

vector<int> temp;

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

for (j = i + 1; j < n; j++) {

sum = v[i] + v[j];

if (mp.find(k - sum) != mp.end()) {

vector<pair<int, int>> vec = mp[k - sum];

int len = vec.size();

for (int k = 0; k < len; k++) {

pair<int, int> p = vec[k];

if (p.first != i && p.first != j && p.second != i && p.second != j) {

temp.push\_back(v[i]);

temp.push\_back(v[j]);

temp.push\_back(v[p.first]);

temp.push\_back(v[p.second]);

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

s.insert(temp);

temp.clear();

}

}

}

}

}

if (s.size() == 0) {

cout << "-1";

}

else {

set<vector<int>>::iterator it;

for (it = s.begin(); it != s.end(); it++) {

temp = \*it;

cout << temp[0] << " " << temp[1] << " "<< temp[2] << " " << temp[3] << " $";

}

}

cout << endl;

}

int main() {

int t = 0;

cin >> t;

int n = 0, k = 0;

while (t-- > 0) {

cin >> n >> k;

vector<int> v(n);

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

cin >> v[i];

}

func(v, n, k);

}

return 0;

}

**6. Swapping pairs make sum equal**

Given two arrays of integers **A[]** and **B[]** of size **N** and **M**, the task is to check if a pair of values (one value from each array) exists such that swapping the elements of the pair will make the sum of two arrays equal.

**Example 1:**

**Input:** N = 6, M = 4

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

B[] = (3, 6, 3, 3)

**Output:** 1

**Explanation**: Sum of elements in A[] = 11

Sum of elements in B[] = 15, To get same

sum from both arrays, we can swap following

values: 1 from A[] and 3 from B[]

**Example 2:**

**Input:** N = 4, M = 4

A[] = {5, 7, 4, 6}

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

**Output:** 1

**Explanation**: We can swap 6 from array

A[] and 2 from array B[]

**Your Task:**  
This is a function problem. You don't need to take any input, as it is already accomplished by the driver code. You just need to complete the function **findSwapValues**() that takes array**A,**array**B,**integer**N,**and integer**M**as parameters and returns **1** if there exists any such pair otherwise returns **-1**.

**Expected Time Complexity:** O(MlogM+NlogN).  
**Expected Auxiliary Space:** O(1).

**Constraints:**  
1 ≤ N, M ≤ 106

CODE:

#include <bits/stdc++.h>

using namespace std;

// } Driver Code Ends

class Solution{

public:

int findSwapValues(int a[], int n, int b[], int m)

{

int suma=0;

int sumb=0;

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

suma+=a[i];

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

sumb+=b[i];

// cout<<suma<<"-"<<sumb<<endl;

if(suma==sumb)

return -1;

sort(a,a+n);

sort(b,b+m);

if(suma>sumb)

{

int diff=(suma-sumb);

if(diff%2)

return -1;

diff=diff/2;

int i=0,j=0;

while(i<n && j<m)

{

int curr=a[i]-b[j];

if(curr==diff)

{

return 1;

}

else if(curr<diff)

{

i++;

}

else

{

j++;

}

}

}

else

{

int diff=sumb-suma;

if(diff%2)

return -1;

diff=diff/2;

int i=0,j=0;

// cout<<"diff="<<diff<<endl;

while(i<m && j<n)

{

int curr=b[i]-a[j];

// cout<<"curr="<<curr<<endl;

if(curr==diff)

{

return 1;

}

else if(curr<diff)

{

i++;

}

else

{

j++;

}

}

}

return -1;

// Your code goes here

}

};

// { Driver Code Starts.

int main()

{

int t;

cin >> t;

while (t--)

{

int n,m;

cin>>n>>m;

int a[n];

int b[m];

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

cin>>a[i];

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

cin>>b[i];

Solution ob;

cout << ob.findSwapValues(a, n, b, m);

cout << "\n";

}

return 0;

}

**7. 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

CODE:

#include <bits/stdc++.h>

using namespace std;

vector <int> countDistinct(int[], int, int);

int main()

{

int t;

cin >> t;

while (t--)

{

int n, k;

cin >> n >> k;

int a[n];

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

cin >> a[i];

vector <int> result = countDistinct(a, n, k);

for (int i : result)

cout << i << " ";

cout << endl;

}

return 0;

}// } Driver Code Ends

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

{

map<int,int> mp;

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

{

mp[arr[i]]++;

}

int cnt=0;

map<int,int> ::iterator it=mp.begin();

while(it!=mp.end())

{

it++;

cnt++;

}

vector<int> v;

v.push\_back(cnt);

int j=k;

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

{

mp[arr[i]]--;

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

cnt--;

if(mp[arr[j]]==0)

cnt++;

mp[arr[j]]++;

v.push\_back(cnt);

j++;

}

return v;

//code here.

}

**8. Array Pair Sum Divisibility Problem**

Given an array of integers and a number k, write a function that returns true if given array can be divided into pairs such that sum of every pair is divisible by k.

**Input:**  
The first line of input contains an integer T denoting the number of test cases. The T test cases follow. Each test case contains an integer n denoting the size of the array. The next line contains the n space separated integers forming the array. The last line contains the value of k.

**Output:**  
Print "True" (without quotes) if given array can be divided into pairs such that sum of every pair is divisible by k or else "False" (without quotes).

**Constraints:**  
1<=T<=100  
2<=n<=10^5  
1<=a[i]<=10^5  
1<=k<=10^5  
  
**Example:  
Input:**  
2  
4  
9 7 5 3  
6  
4  
91 74 66 48  
10

**Output:**  
True  
False

CODE:

// in pairs such that every pair is divisible by k.

#include <bits/stdc++.h>

using namespace std;

// Returns true if arr[0..n-1] can be divided into pairs

// with sum divisible by k.

bool canPairs(int arr[], int n, int k)

{

// An odd length array cannot be divided into pairs

if (n & 1)

return false;

// Create a frequency array to count occurrences

// of all remainders when divided by k.

unordered\_map<int, int> freq;

// Count occurrences of all remainders

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

freq[arr[i] % k]++;

// Traverse input array and use freq[] to decide

// if given array can be divided in pairs

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

{

// Remainder of current element

int rem = arr[i] % k;

// If remainder with current element divides

// k into two halves.

if (2\*rem == k)

{

// Then there must be even occurrences of

// such remainder

if (freq[rem] % 2 != 0)

return false;

}

// If remainder is 0, then there must be two

// elements with 0 remainder

else if (rem == 0)

{

if (freq[rem] & 1)

return false;

}

// Else number of occurrences of remainder

// must be equal to number of occurrences of

// k - remainder

else if (freq[rem] != freq[k - rem])

return false;

}

return true;

}

/\* Driver program to test above function \*/

int main()

{

int t;

cin>>t;

while(t--)

{

int n;

cin>>n;

int arr[n];

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

cin>>arr[i];

int k;

cin>>k;

canPairs(arr, n, k)? cout << "True": cout << "False";

cout<<endl;;

}

return 0;

}

**9. Longest consecutive subsequence**

Given an array **arr**[] of positive integers. Find the length of the longest sub-sequence such that elements in the subsequence are consecutive integers, the**consecutive numbers can be in any order.**

**Input:**  
The first line of input contains T, number of test cases. First line of line each test case contains a single integer N.  
Next line contains N integer array.

**Output:**  
Print the output of each test case in a seprate line.

**Constraints:**  
1 <= T <= 100  
1 <= N <= 105  
0 <= a[i] <= 105

**Example:**  
**Input:**  
2  
7  
2 6 1 9 4 5 3  
7  
1 9 3 10 4 20 2

**Output:**  
6  
4

CODE:

using namespace std;

int main()

{

int t;

cin>>t;

while(t--)

{

int n;

cin>>n;

int arr[n];

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

{

cin>>arr[i];

}

sort(arr,arr+n);

int len=0;

int cnt=0;

for(int i=0;i<n-1;i++)

{

if((arr[i+1]-arr[i])==1)

{

cnt++;

len=max(cnt,len);

}

else if(arr[i+1]-arr[i]==0)

{

continue;

}

else

{

cnt=0;

}

}

cout<<len+1<<endl;

}

return 0;

}

**10. Array Subset of another array**

Given two arrays: arr1[0..m-1] of size m and arr2[0..n-1] of size n. Task is to check whether arr2[] is a subset of arr1[] or not. Both the arrays can be both unsorted or sorted. It may be assumed that elements in both array are distinct.

**Input:**  
The first line of input contains an integer T denoting the number of test cases. Then T test cases follow. Each test case contains an two integers m and n denoting the size of arr1 and arr2 respectively. The following two lines contains the space separated elements of arr1 and arr2 respectively.  
  
**Output:**  
Print "Yes"(without quotes) if arr2 is subset of arr1.  
Print "No"(without quotes) if arr2 is not subset of arr1.  
  
**Constraints:**  
1 <= T <= 100  
1 <= m,n <= 105  
1 <= arr1[i], arr2[j] <= 105  
  
**Example:  
Input:**  
3  
6 4  
11 1 13 21 3 7  
11 3 7 1  
6 3  
1 2 3 4 5 6  
1 2 4  
5 3  
10 5 2 23 19  
19 5 3  
  
**Output:**  
Yes  
Yes  
No

CODE:

using namespace std;

#include <bits/stdc++.h>

int main() {

int t;

cin>>t;

while(t--)

{

int n,n1;

cin>>n>>n1;

int arr[n],arr1[n1];

unordered\_map<int,int>mp;

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

{

cin>>arr[i];

mp[arr[i]]++;

}

for(int i=0;i<n1;i++)

cin>>arr1[i];

if(n1>n)

cout<<"No"<<endl;

else

{

int flag=0;

for(int i=0;i<n1;i++)

{

if(mp.find(arr1[i])==mp.end())

{flag=1;

break;

}

}

if(flag==1)

cout<<"No"<<endl;

else

cout<<"Yes"<<endl;

}

}

}

**11. Find all pairs with a given sum**

Given two unsorted arrays **A** of size **N** and **B** of size **M** of distinct elements, the task is to find all pairs from both arrays whose sum is equal to **X**.

**Input:**  
The first line of input contains an integer T denoting the no of test cases. Then T test cases follow. Each test case contains 3 lines . The first line contains 3 space separated integers N, M, X. Then in the next two lines are space separated values of the array A and B respectively.

**Output:**  
For each test case in a new line print the sorted space separated values of all the pairs u,v where u belongs to array A and v belongs to array B, such that each pair is separated from the other by a ',' without quotes also add a space after the ',' . If no such pair exist print **-1**.

**Constraints:**  
1 <= T <= 100  
1 <= N, M, X <= 106  
-106 <= A, B <= 106

**Example:  
Input:**  
2  
5 5 9  
1 2 4 5 7  
5 6 3 4 8  
2 2 3  
0 2  
1 3  
**Output:**  
1 8, 4 5, 5 4  
0 3, 2 1

CODE:

using namespace std;

int main()

{

int t;

cin>>t;

while(t--)

{

int n,m,k;

cin>>n>>m>>k;

int arr1[n];

int arr2[m];

map<int,int>mp;

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

{

cin>>arr1[i];

mp[arr1[i]]++;

}

int flag=0;

vector<pair<int,int>>v;

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

{

cin>>arr2[i];

if(mp.find(k-arr2[i])!=mp.end())

{

//cout<<k-arr2[i]<<" "<<arr2[i];

v.push\_back(make\_pair(k-arr2[i],arr2[i]));

}

}

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

if(v.size()==0)

{

cout<<-1<<endl;

continue;

}

if(v.size()>0)

cout<<v[0].first<<" "<<v[0].second;

if(v.size()>1)

for(int i=1;i<v.size();i++)

{

cout<<", "<<v[i].first<<" "<<v[i].second;

}

cout<<endl;

}

return 0;

}

**12. Find first repeated character**

Given a string **S**. The task is to find the first repeated character in it. We need to find the character that occurs more than once and whose index of second occurrence is smallest. S contains only lowercase letters.  
**Input:**  
The first line of input contains an integer **T** denoting the no of test cases. Then **T** test cases follow. Each test case contains a string **S**.

**Output:**  
For each test case in a new line print the first repeating character in the string. If no such character exist print -1.

**Constraints:**  
1 <= T <= 100  
1 <= |S| <=104

**Example:**  
**Input:**  
2  
geeksforgeeks  
hellogeeks

**Output:**  
e  
l

CODE:

using namespace std;

int main()

{

int t;

cin>>t;

while(t--)

{

string str;

cin>>str;

map<int,int> mp;

int flag=0;

for(int i=0;i<str.length();i++)

{

if(mp[str[i]]>0)

{

flag=1;

cout<<str[i]<<endl;

break;

}

mp[str[i]]++;

}

if(!flag)

cout<<-1<<endl;

}

return 0;

}

**13. Zero Sum Subarrays**

You are given an array of integers. You need to print the total count of sub-arrays having their sum equal to 0

**Input:**  
First line of the input contains an integer T denoting the number of test cases. Each test case consists of two lines. First line of each test case contains an Integer N denoting the size of the array, and the second line contains N space separated integers.

**Output:**  
For each test case, in a new line, print the total number of subarrays whose sum is equal to 0.

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

**Constraints:**  
1 <= T <= 100  
1 <= N <= 107  
-1010 <= Ai <= 1010

**Example:**  
**Input:**  
2  
6  
0 0 5 5 0 0  
10  
6  -1 -3 4 -2 2 4 6 -12 -7  
**Output:**  
6  
4

CODE:

using namespace std;

#define ll long long

int main()

{

ll t;

cin>>t;

while(t--)

{

ll n;

cin>>n;

ll arr[n];

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

cin>>arr[i];

unordered\_map<ll,ll> mp;

// vector<pair<ll,ll>> v;

ll sum=0;

ll cnt=0;

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

{

sum+=arr[i];

if(sum==0)

{

cnt++;

}

if(mp[sum])

{

cnt+=mp[sum];

}

mp[sum]++;

}

cout<<cnt<<endl;

}

return 0;

}

**14. Minimum indexed character**

Given a string **str** and another string **patt**. Find the character in **patt** that is present at the minimum index in **str**. If no character of **patt** is present in **str** then print ‘No character present’.

**Input:**  
The first line of input contains an integer T denoting the number of test cases. Then the description of T test cases follow. Each test case contains two strings **str** and **patt** respectively.

**Output:**  
Output the character in patt that is present at the minimum index in str. Print "**$**" (without quotes) if no character of patt is present in str.

**Constraints:**  
1 <= T <= 105  
1 <= |str|, |patt| <= 105

**Example:  
Input:**  
2  
geeksforgeeks  
set  
adcffaet  
onkl

**Output:**  
e  
$

CODE:

int main()

{

int t;

cin>>t;

while(t--)

{

string str,test;

cin>>str>>test;

map<char,int> mp;

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

{

mp[test[i]]++;

}

int flag=0;

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

{

if(mp[str[i]]>0)

{

flag=1;

cout<<str[i]<<endl;

break;

}

}

if(!flag)

cout<<"$"<<endl;

}

return 0;

}

**15. Check if two arrays are equal or not**

Given two arrays **A** and **B** of equal size, 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.

**Input:**  
The first line of input contains an integer**T** denoting the no of test cases. Then **T** test cases follow.  Each test case contains 3 lines of input. The first line contains an integer **N**denoting the size of the array. The second line contains element of array **A[]**. The third line contains elements of the array **B[]**.

**Output:**  
For each test case, print **1** if the arrays are equal else print **0**.

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

**Constraints:**  
1 <= T <= 100  
1 <= N <= 107  
1 <= A[],B[] <= 1018

**Example:  
Input:**  
2  
5  
1 2 5 4 0  
2 4 5 0 1  
3  
1 2 5  
2 4 15  
**Output:**  
1  
0

**Explanation:**  
**Testcase1:**  
Input : A[] = {1, 2, 5, 4, 0}; B[] = {2, 4, 5, 0, 1}; Since all the array elements are same. So,  
Output : 1  
**Testcase2:**  
Input : A[] = {1, 2, 5}; B[] = {2, 4, 15}; Since all the array elements are not same. So,  
Output : 0

CODE:

using namespace std;

#define ll long long

int main()

{

ll t;

cin>>t;

while(t--)

{

ll n;

cin>>n;

map<ll,ll> mp;

map<ll,ll> mp1;

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

{

ll x;

cin>>x;

mp[x]++;

}

ll arr[n];

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

{

cin>>arr[i];

mp1[arr[i]]++;

}

ll flag=0;

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

{

if(mp[arr[i]]!=mp1[arr[i]])

{

flag=1;

break;

}

}

if(flag)

cout<<0<<endl;

else

cout<<1<<endl;

}

return 0;

}

**16. Uncommon characters**

Given two strings A and B. Find the characters that are not common in the two strings.

**Example 1:**

**Input:**

A = geeksforgeeks

B = geeksquiz

**Output:** fioqruz

**Explanation:**

The characters 'f', 'i', 'o', 'q', 'r', 'u','z'

are either present in A or B, but not in both.

**Example 2:**

**Input:**

A = characters

B = alphabets

**Output:** bclpr

**Explanation:** The characters 'b','c','l','p','r'

are either present in A or B, but not in both.

**Your Task:**  
You dont need to read input or print anything. Complete the function **UncommonChars()** which takes strings A and B as input parameters and returns a string that contains all the uncommon characters in sorted order. If no such character exists return "-1".

**Expected Time Complexity:**O(M+N) where M and N are the sizes of A and B respectively.  
**Expected Auxiliary Space:** O(M+N)

**Constraints:**  
1<= M,N <= 104  
String may contain duplicate characters

CODE:

using namespace std;

int main()

{

int t;

cin>>t;

while(t--)

{

string a,b;

cin>>a>>b;

map<char,int> mp1;

map<char,int> mp;

for(int i=0;i<a.length();i++)

{

mp1[a[i]]=1;

}

string str="";

for(int i=0;i<b.length();i++)

{

mp[b[i]]=1;

}

for(int i=0;i<26;i++)

{

char c=char(i+97);

if((mp[c]==1 && mp1[c]==0) || (mp1[c]==1 && mp[c]==0) )

str+=c;

}

if(str=="")

cout<<-1<<endl;

else

cout<<str<<endl;

}

return 0;

}

**17. Smallest window in a string containing all the characters of another string**

Given a string **S** and text **T**. Output the smallest window in the string **S** having all characters of the text **T**. Both the string S and text T contains lowercase english alphabets.

**Input:**  
First line of the input contains an integer T, denoting the number of test cases. Then T test case follows. Each test contains 2 lines having a string **S** and next line contains text **T**.

**Output:**  
Output the smallest window of the string containing all the characters of the text. If such window doesn`t exist or this task can not be done then print -1.

**Constraints:**  
1 <= T <= 100  
1 <= |N|, |X| <= 1000

**Example:  
Input:**  
2  
timetopractice  
toc  
zoomlazapzo  
oza

**Output:**  
toprac  
apzo

CODE:

using namespace std;

bool check(vector<int> v1,vector<int> v2)

{

for(int i=0;i<26;i++)

{

if(v1[i]<v2[i])

return false;

}

return true;

}

int main()

{

int t;

cin>>t;

while(t--)

{

string a,b;

cin>>a>>b;

vector<int> v1(26);

vector<int> v2(26);

for(int i=0;i<b.length();i++)

v2[b[i]-'a']++;

int len=INT\_MAX;

string ans="";

int start=0;

for(int i=0;i<a.length();i++)

{

v1[a[i]-'a']++;

while(check(v1,v2))

{

if(i-start+1<len)

{

len=i-start+1;

ans=a.substr(start,len);

}

v1[a[start]-'a']--;

start++;

}

}

if(ans=="")

cout<<-1;

else

cout<<ans;

cout<<endl;

}

return 0;

}

**18. First element to occur k times**

Given an array of **N** integers. The task is to find the first element that occurs **K** number of times. If no element occurs **K** times the print **-1**.

**Input:**  
The first line of input contains an integer T denoting the number of test cases. Then T test cases follow. The first line of each test case contains an integer N denoting the size of an array and the number K. The second line of each test case contains N space separated integers denoting elements of the array A[ ].

**Output:**  
For each test case in a new line print the required answer.

**Constraints:**  
1 <= T <= 100  
1 <= N, K <= 105  
1<= A <= 106

**Example:**  
**Input :**  
1  
7 2  
1 7 4 3 4 8 7  
**Output :**  
7

**CODE:**

using namespace std;

int main()

{

int t;

cin>>t;

while(t--)

{

int n,k;

cin>>n>>k;

map<int,int> mp;

int flag=0;

vector<int> v;

int arr[n];

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

{

int x;

cin>>x;

arr[i]=x;

mp[x]++;

}

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

{

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

{

flag=1;

cout<<arr[i]<<endl;

break;

}

}

if(!flag)

cout<<-1<<endl;

}

return 0;

}

**19. Check if frequencies can be equal**

Given a string **s** which contains only lower alphabetic characters, check if it is possible to remove at most one character from this string in such a way that frequency of each distinct character becomes same in the string.

**Example 1:**

**Input:**

s = xyyz

**Output:** 1

**Explanation:** Removing one 'y' will make

frequency of each letter 1.

**Example 2:**

**Input:**

s = xxxxyyzz

**Output:** 0

**Explanation:** Frequency can not be made same

same by removing just 1 character.

**Your Task:**  
You dont need to read input or print anything. Complete the function**sameFreq()**which takes a string as input parameter and returns a boolean value denoting if same frequency is possible or not.  
**Note:**The driver code print 1 if the value returned is true, otherwise 0.

**Expected Time Complexity:** O(N) where N is length of str  
**Expected Auxiliary Space:** O(1)

**Constraints:**  
1 <= str.length() <= 104

CODE:

using namespace std;

int main()

{

int t;

cin>>t;

while(t--)

{

string str;

cin>>str;

map<int,int> mp;

int maxn=INT\_MIN;

for(int i=0;i<str.length();i++)

{

mp[str[i]]++;

maxn=max(maxn,mp[str[i]]);

}

map<int,int> ::iterator it=mp.begin();

int cnt0=0;

int cnt1=0;

int cnt=0;

while(it!=mp.end())

{

if(maxn-it->second==0)

cnt0++;

if(maxn-it->second==1)

cnt1++;

it++;

cnt++;

}

if((cnt0==cnt-1 && cnt1==1) || (cnt0==1 && cnt1==cnt-1) || (cnt0==cnt))

cout<<1<<endl;

else

cout<<0<<endl;

}

return 0;

}