1. **Find the element that appears once in sorted array**

Given a sorted array arr[] of size N. Find the element that appears only once in the array. All other elements appear exactly twice.

**Example 1:**

**Input:**

N = 11

arr[] = {1, 1, 2, 2, 3, 3, 4, 50, 50, 65, 65}

**Output:** 4

**Explanation:** 4 is the only element that

appears exactly once.

**Your Task:**  
You don't need to read input or print anything. Complete the function**findOnce()**which takes sorted array and its size as its input parameter and returns the element that appears only once.

**Expected Time Complexity:** O(log N)  
**Expected Auxiliary Space:** O(1)

**Constraints:**  
-105 <= N <= 105

Code:

// Driver code

#include<bits/stdc++.h>

using namespace std;

// } Driver Code Ends

class Solution

{

public:

int findOnce(int arr[], int n)

{

int l=0;

int h=n-1;

while(l<=h)

{

int mid=(l+h)/2;

if(l==h)

return arr[mid];

else if(mid&1)

{

if(arr[mid]==arr[mid-1])

l=mid+1;

else

h=mid-1;

}

else

{

if(arr[mid]==arr[mid+1])

l=mid+2;

else

h=mid;

}

}

}

};

// { Driver Code Starts.

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];

}

Solution ob;

int res = ob.findOnce(A,n);

cout << res << endl;

}

return 0;

} // } Driver Code Ends

2. **Search in a Rotated Array**

Given a sorted and rotated array **A**of N distinct elements which is rotated at some point, and given an element **K**. The task is to find the index of the given element K in the array A.

**Input:**  
The first line of the input contains an integer **T**, denoting the total number of test cases. Then T test cases follow. Each test case consists of three lines. First line of each test case contains an integer **N** denoting the size of the given array. Second line of each test case contains N space separated integers denoting the elements of the array A. Third line of each test case contains an integer Kdenoting the element to be searched in the array.

**Output:**  
Corresponding to each test case, output the index of the element found in the array.  If element is not present, then output -1.

**Constraints:**  
1 ≤ T ≤ 100  
1 ≤ N ≤ 107  
0 ≤ Ai ≤ 108  
1 ≤ K ≤ 108

**Example:**  
**Input:**  
3  
9  
5 6 7 8 9 10 1 2 3  
10  
3  
3 1 2  
1  
4  
3 5 1 2  
6

**Output:**  
5  
1  
-1

**Explanation:**  
**Testcase 1:**10 is found at index 5.

**Code:**

using namespace std;

int check(int arr[],int key,int n){

int l=0;

int h=n-1;

while(l<=h)

{

int mid=(l+h)/2;

if(arr[mid]==key)

return mid;

else if(arr[l]<arr[mid])

{

if(arr[mid]>=key && key>=arr[l])

h=mid-1;

else

l=mid+1;

}

else

{

if(arr[mid]<=key && key<=arr[h])

l=mid+1;

else

h=mid-1;

}

}

return -1;

}

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 key;

cin>>key;

cout<<check(arr,key,n)<<endl;

}

return 0;

}

3. **Binary Search**

Given a sorted array of integers and a key to be searched in the array. Print the position of the key in the array, if present. If it's not present in the array, report it.

**Input:**  
The first line contains an integer **'T'** denoting the number of test cases. Then **'T'** test cases follow. Each test case consists of 3 lines. First line of each test case contains an integer **N**denoting the size of the array. Second line of each test case consists of**'N'**space separated integers denoting the elements of the array**A[]**. The third line contains a key **'k'**.

**Output:**  
Prints the position of the key if its present in the array else print **-1** if the key is not present in the array.

**User Task:**  
You don't need to read input or print anything. Your task is to complete the function **bin\_search()** which takes the array **A[]**, key **k**and the leftmost and the rightmost index in the array (**left**and **right**respectively) and returns the position where the key **k**occurs in the array. If it's not present, return -1.

**Expected Time Complexity:**O(LogN)  
**Expected Auxiliary Space:**O(LogN) if solving recursively and O(1) otherwise.

**Constraints:**  
1 <= T <= 100  
1 <= N <= 104  
1 <= arr[i] <= 104

**Example:**  
**Input:**  
2  
5  
1 2 3 4 5   
4  
5  
11 22 33 44 55  
445

**Output:**  
3  
-1

**Explanation:**  
**Test Case 1:**  
4 is present at the index 3 (0-based) in the array.

**Test Case 2:**  
Since 445 is not present in the given array, we return -1.

Code:

#include<bits/stdc++.h>

using namespace std;

int bin\_search(int A[],int left,int right,int k);

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];

int key;

cin>>key;

int found = bin\_search(a,0,N-1,key);

cout<<found<<endl;

}

}

// } Driver Code Ends

/\*You need to complete this function \*/

int bin\_search(int arr[], int start, int end, int k)

{

while(start<=end)

{

int mid=(start+end)/2;

if(arr[mid]==k)

return mid;

else if(arr[mid]>k)

return bin\_search(arr,start,mid-1,k);

else

return bin\_search(arr,mid+1,end,k);

}

return -1;

}

4. **Sum of Middle Elements of two sorted arrays**

Given 2 sorted arrays **A** and **B** of size **N** each. Print sum of middle elements of the array obtained after merging the given arrays.

**Input:**  
The first line contains T denoting the number of testcases. Then follows description of testcases.  
Each case begins with a single positive integer **N** denoting the size of array. The second line contains the N space separated positive integers denoting the elements of array A. The third line contains N space separated positive integers denoting the elements of array B.

**Output:**  
For each testcase, print the sum of middle elements of two sorted arrays. The number of the elements in the merged array are even so there will be 2 numbers in the center n/2 and n/2 +1. You have to print their sum.

**Constraints:**  
1 <= T <= 50  
1 <= N <= 103  
1 <= A[i] <= 106  
1 <= B[i] <= 106

**Example:**  
**Input :**  
1  
5  
1 2 4 6 10  
4 5 6 9 12

**Output :**  
11

**Explanation:**  
**Testcase 1:** After merging two arrays, sum of middle elements is 11 (5 + 6).

Code:

using namespace std;

int ans(int \*a,int \*b,int low,int high,int n)

{

int x = (low+high)/2;

if(a[x-1]<=b[n-x] and b[n-x-1]<=a[x])

return max(a[x-1],b[n-x-1])+min(b[n-x],a[x]);

else if(a[x]>b[n-x])

return ans(a,b,low,x-1,n);

else

return ans(a,b,x+1,high,n);

}

int main()

{

int t;

cin>>t;

while(t--)

{

int n;

cin>>n;

int arr1[n];

int arr2[n];

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

cin>>arr1[i];

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

cin>>arr2[i];

cout<<ans(arr1,arr2,0,n-1,n)<<endl;

}

return 0;

}

5. **Quick Sort**

Given an array of integers. Complete the partition() function used for the implementation of Quick Sort.

**Example 1:**

**Input**: N = 5, arr[] = { 4, 1, 3, 9, 7}

**Output**: 1 3 4 7 9

**Example 2:**

**Input**: N = 10,

arr[] = {10, 9, 8, 7, 6, 5, 4, 3, 2, 1}

**Output**: 1 2 3 4 5 6 7 8 9 10

**Your Task:**You don't need to read input or print anything. Your task is to complete the function **partition()** which takes the array arr[] and the range of indices to be considered [low, high] and partitions the array in the given range considering the last element as the pivot such that, all the elements less than(or equal to) the pivot lie before the elements greater than it.   
  
**Expected Time Complexity:**O(N).  
**Expected Auxiliary Space:**O(1).  
  
**Constraints:**  
1 <= N <= 103  
1 <= arr[i] <= 104

Code:

#include <stdio.h>

#include <bits/stdc++.h>

using namespace std;

int partition (int arr[], int low, int high);

/\* The main function that implements QuickSort

arr[] --> Array to be sorted, low --> Starting index, high --> Ending index \*/

void quickSort(int arr[], int low, int high)

{

if (low < high)

{

/\* pi is partitioning index, arr[p] is now at right place \*/

int pi = partition(arr, low, high);

// Separately sort elements before partition and after partition

quickSort(arr, low, pi - 1);

quickSort(arr, pi + 1, high);

}

}

/\* Function to print an array \*/

void printArray(int arr[], int size)

{

int i;

for (i=0; i < size; i++)

printf("%d ", arr[i]);

printf("\n");

}

// Driver program to test above functions

int main()

{

int arr[1000],n,T,i;

scanf("%d",&T);

while(T--){

scanf("%d",&n);

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

scanf("%d",&arr[i]);

quickSort(arr, 0, n-1);

printArray(arr, n);

}

return 0;

}// } Driver Code Ends

/\* The main function that implements QuickSort

arr[] --> Array to be sorted, low --> Starting index, high --> Ending index

void quickSort(int arr[], int low, int high) {

if (low < high) {

// pi is partitioning index, arr[p] is now at right place

int pi = partition(arr, low, high);

// Separately sort elements before / partition and after partition

quickSort(arr, low, pi - 1);

quickSort(arr, pi + 1, high);

}

}\*/

/\* This function takes last element as pivot, places the pivot element

at its correct position in sorted array, and places all smaller (smaller

than pivot) to left of pivot and all greater elements to right of pivot \*/

int partition (int arr[], int low, int high)

{

int pivot=arr[high];

int i=low-1;

for(int j=low;j<high;j++)

{

if(arr[j]<pivot)

{

i++;

swap(arr[i],arr[j]);

}

}

swap(arr[i+1],arr[high]);

return i+1;

// Your code here

}

6. **Merge Sort**

The task is to complete **merge()** function which is used to implement Merge Sort.

**Example 1:**

**Input:**

5

4 1 3 9 7

**Output:**

1 3 4 7 9

**Example 2:**

**Input:**

10

10 9 8 7 6 5 4 3 2 1

**Output:**

1 2 3 4 5 6 7 8 9 10

**Your Task:**  
You don't need to take the input or print anything. Your task is to complete the function **merge()** which takes the array arr[], the starting position of the first array (l),  the ending position of the first array (m) and the ending position of the second array (r) as its inputs and modifies the array arr[] such that it is sorted from position l to position r. The range [l,m] and [m+1,r] are already sorted.

**Expected Auxiliary Space:**O(n)  
**Expected Time Complexity:**O(n)  (for the merge() function)

**Constraints:**  
1 <= N <= 105  
1 <= arr[i] <= 103

Code:

// C program for implementation of Merge Sort

#include <stdio.h>

#include <bits/stdc++.h>

using namespace std;

void merge(int arr[], int l, int m, int r);

/\* l is for left index and r is right index of the

sub-array of arr to be sorted \*/

void mergeSort(int arr[], int l, int r)

{

if (l < r)

{

// Same as (l+r)/2, but avoids overflow for

// large l and h

int m = l+(r-l)/2;

// Sort first and second halves

mergeSort(arr, l, m);

mergeSort(arr, m+1, r);

merge(arr, l, m, r);

}

}

/\* Function to print an array \*/

void printArray(int arr[], int size)

{

int i;

for (i=0; i < size; i++)

printf("%d ", arr[i]);

printf("\n");

}

// Driver program to test above functions

int main()

{

int n,T,i;

scanf("%d",&T);

while(T--){

scanf("%d",&n);

int arr[n+1];

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

scanf("%d",&arr[i]);

mergeSort(arr, 0, n-1);

printArray(arr, n);

}

return 0;

}// } Driver Code Ends

/\* The task is to complete merge() which is used

in below mergeSort() \*/

/\* l is for left index and r is right index of the

sub-array of arr to be sorted

void mergeSort(int arr[], int l, int r) {

if (l < r) {

int m = l+(r-l)/2;

mergeSort(arr, l, m);

mergeSort(arr, m+1, r);

merge(arr, l, m, r);

}

} \*/

// Merges two subarrays of arr[]. First subarray is arr[l..m]

// Second subarray is arr[m+1..r]

void merge(int arr[], int l, int m, int r)

{

int n1=m-l+1;

int n2=r-m;

int left[n1];

int right[n2];

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

left[i]=arr[l+i];

for(int i=0;i<n2;i++)

right[i]=arr[i+m+1];

int i=0;

int j=0;

int k=l;

while(i<n1 && j<n2)

{

if(left[i]<right[j])

{

arr[k]=left[i];

i++;

k++;

}

else

{

arr[k]=right[j];

k++;

j++;

}

}

while(i<n1)

{

arr[k]=left[i];

k++;

i++;

}

while(j<n2)

{

arr[k]=right[j];

j++;

k++;

}

}

7. **K-th element of two sorted Arrays**

Given two sorted arrays **A** and **B** of size **M** and **N** respectively and an element k. The task is to find the element that would be at the k’th position of the final sorted array.

**Input:**  
First line consists of test cases T. First line of every test case consists of 3 integers N, M and K, denoting M number of elements in A, N number of elements in B and kth position element. Second and Third line of every test case consists of elements of A and B respectively.

**Output:**  
Print the element at the Kth position.

**Constraints:**  
1 <= T <= 200  
1 <= N, M <= 106  
1 <= Ai, Bi <= 106  
1 <= K <= N+M

**Example:  
Input:**  
1  
5 4 5  
2 3 6 7 9  
1 4 8 10

**Output:**  
6

**Explanation:  
Testcase 1:** Element at 5th position after merging both arrays will be 6.

Code:

using namespace std;

int kth(int \*arr1, int \*arr2, int \*end1, int \*end2, int k)

{

if (arr1 == end1)

return arr2[k];

if (arr2 == end2)

return arr1[k];

int mid1 = (end1 - arr1) / 2;

int mid2 = (end2 - arr2) / 2;

if (mid1 + mid2 < k)

{

if (arr1[mid1] > arr2[mid2])

return kth(arr1, arr2 + mid2 + 1, end1, end2,

k - mid2 - 1);

else

return kth(arr1 + mid1 + 1, arr2, end1, end2,

k - mid1 - 1);

}

else

{

if (arr1[mid1] > arr2[mid2])

return kth(arr1, arr2, arr1 + mid1, end2, k);

else

return kth(arr1, arr2, end1, arr2 + mid2, k);

}

}

int main()

{

int t;

cin>>t;

while(t--)

{

int n,m,k;

cin>>n>>m>>k;

int arr1[n],arr2[m];

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

cin>>arr1[i];

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

cin>>arr2[i];

cout<<kth(arr1,arr2,arr1+n,arr2+m,k-1)<<endl;

}

return 0;

}