**Answer Script**

|  |
| --- |
| Question No. 01 |
| Write a C++ program that takes N integer numbers and sorts them in non-increasing order using Merge Sort. You can’t use any built-in function for sorting. |
| Answer No. 01 |
| #include<bits/stdc++.h>  using namespace std;  void merge(vector<int> &v, int l, int r, int mid){      int left\_size = mid-l+1;      int L[left\_size + 1];      int right\_size = r-mid;      int R[right\_size + 1];      for (int i = l, j=0; i <= mid; i++, j++)      {          L[j] = v[i];      }      for (int i = mid+1, j=0; i <= r; i++, j++)      {          R[j] = v[i];      }      L[left\_size] = INT\_MIN;      R[right\_size] = INT\_MIN;      int lp = 0, rp = 0;      for (int i = l; i <= r; i++)      {          if(L[lp] >= R[rp]){              v[i] = L[lp];              lp++;          } else {              v[i] = R[rp];              rp++;          }      }  }  void mergeSort(vector<int> &v, int l, int r){      if(l == r) return;      int mid = (l+r)/2;      mergeSort(v, l, mid); // divide ---> left segment      mergeSort(v, mid+1, r); // divide ----> right segment      merge(v, l, r, mid); // conquer ----> merge left and right  }  int main()  {      int n; cin>>n;      vector<int> v;      for (int i = 0; i < n; i++)      {          int val; cin>>val;          v.push\_back(val);      }      mergeSort(v, 0, n-1);      for(int val:v){          cout<<val<<" ";      }        return 0;  } |

|  |
| --- |
| Question No. 02 |
| Write a C++ program that takes N integer numbers that are sorted and distinct. The next line will contain an integer k. You need to tell whether K exists in that array or not. If it exists, print its index otherwise print “Not Found”. You must solve this in O(logn) complexity. |
| Answer No. 02 |

#include<bits/stdc++.h>

using namespace std;

int binarySearch(vector<int> v, int l, int r, int sz, int k){

    while (l<=r)

    {

        int mid = (l+r)/2;

        if(v[mid] == k){

            return mid;

        } else if (v[mid] < k){

            l = mid + 1;

        } else {

            r = mid - 1;

        }

    }

    return -1;

}

int main()

{

    int n; cin>>n;

    vector<int> v;

    for (int i = 0; i < n; i++)

    {

        int val; cin>>val;

        v.push\_back(val);

    }

    int k; cin>>k;

    int result = binarySearch(v, 0, n-1, n, k);

    if(result == -1) cout<<"Not Found"<<endl;

    else cout<<result<<endl;

    return 0;

}

|  |
| --- |
| Question No. 03 |
| You are given an array of N positive integers. The next line will contain an integer K. You need to tell whether there exists more than one occurrence of K in that array or not. If there exists more than one occurrence of K print YES, Otherwise print NO. See the sample input-output for more clarification. The given array will be sorted in increasing order. And it is guaranteed that at least one occurrence of K will exist. You must solve this in O(logn) complexity. |
| Answer No. 03 |

#include<bits/stdc++.h>

using namespace std;

bool binarySearch(vector<int> v, int l, int r, int sz, int k){

    while (l<=r)

    {

        int mid = (l+r)/2;

        if(v[mid] == k){

            if(v[mid-1] == k) return true;

            else if(v[mid+1] == k) return true;

            else return false;

        } else if (v[mid] < k){

            l = mid + 1;

        } else {

            r = mid - 1;

        }

    }

    return false;

}

int main()

{

    int n; cin>>n;

    vector<int> v;

    for (int i = 0; i < n; i++)

    {

        int val; cin>>val;

        v.push\_back(val);

    }

    int k; cin>>k;

    bool result = binarySearch(v, 0, n-1, n, k);

    if(result) cout<<"YES"<<endl;

    else cout<<"NO"<<endl;

    return 0;

}

|  |
| --- |
| Question No. 04 |
| Calculate the time complexity of the following code snippets. |
| Answer No. 04 |
| 1. complexity **– O(n\*logn) -> outer loop = O(logn) and inner loop = O(n/5) or O(n)** 2. complexity **– O(√ n) -> square root of n** 3. complexity **– O(logn\*√ n) -> outer loop = O(logn) and inner loop = O(√ n)** 4. complexity **– O(√ n)** |

|  |
| --- |
| Question No. 05 |
| You are given two sorted arrays arr1 and arr2 in descending order. Your task is to merge these two arrays into a new array result using the merge sort technique, but Instead of merging the arrays in ascending order, you need to merge them in descending order to create the result array. |
| Answer No. 05 |
| #include<bits/stdc++.h>  using namespace std;  vector<int> merge(vector<int> v1, int m, vector<int> v2, int n){      vector<int> v;      v1.push\_back(INT\_MIN);      v2.push\_back(INT\_MIN);      int lp = 0, rp = 0;      for (int i = 0; i < (n+m) ; i++)      {          if(v1[lp] >= v2[rp]){              v.push\_back(v1[lp]);              lp++;          } else {              v.push\_back(v2[rp]);              rp++;          }      }        return v;  }  int main()  {      int m; cin>>m;      vector<int> v1;      for (int i = 0; i < m; i++)      {          int val; cin>>val;          v1.push\_back(val);      }      int n; cin>>n;      vector<int> v2;      for (int i = 0; i < n; i++)      {          int val; cin>>val;          v2.push\_back(val);      }      vector<int> v = merge(v1, m, v2, n);      for(int val:v){          cout<<val<<" ";      }        return 0;  } |