Question: 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.

Marks: 20

Sample Input	Sample Output
7 1 2 9 4 0 2 5	9 5 4 2 2 1 0
6 5 3 -1 3 3 8	8 5 3 3 3 -1

Ans :

```
#include <bits/stdc++.h>
using namespace std;
void merge(int arr[], int left, int right, int mid)
{
   int leftSize = mid - left + 1;
   int rightSize = right - mid;
   int l[leftSize];
   int r[rightSize];
   for (int i = 0; i < leftSize; i++)
   {
       l[i] = arr[left + i];
   }
```

```
    for (int i = 0; i < rightSize; i++)
    {
        r[i] = arr[mid + 1 + i];
    }
    int lp = 0, rp = 0, k = left;
    while (lp < leftSize && rp < rightSize)
    {
        if (l[lp] >= r[rp])
        {
            arr[k] = l[lp];
            lp++;
            k++;
        }
        else
        {
            arr[k] = r[rp];
            rp++;
            k++;
        }
    }
    while (lp < leftSize)
    {
        arr[k] = l[lp];
        lp++;
        k++;
    }
    while (rp < rightSize)
    {
```

```
        arr[k] = r[rp];
        rp++;
        k++;
    }
}
void mergeSort(int arr[], int left, int right)
{
    if (left == right)
    {
        return;
    }
    int mid = (left + right) / 2;
    mergeSort(arr, left, mid);
    mergeSort(arr, mid + 1, right);
    merge(arr, left, right, mid);
}
int main()
{
    int size;
    cin >> size;
    int arr[size];
    for (int i = 0; i < size; i++)
    {
        cin >> arr[i];
    }
    int left = 0;
    int right = size - 1;
    mergeSort(arr, left, right);
```

```
    for (int i = 0; i < size; i++)
    {
        cout << arr[i] << " ";
    }
    return 0;
}
```

Question: 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.

Marks: 20

Sample Input	Sample Output
8 -4 0 2 6 9 10 29 54 29	6
10 0 1 2 3 4 5 6 7 8 9 -3	Not Found

Ans :

```
#include <bits/stdc++.h>
using namespace std;
void binarySearch(int arr[], int size)
{
    int left = 0;
    int right = size - 1;
    int mid;
    int key;
    cin >> key;
    int flag = 0;
    while (left != right)
    {
        mid = (left + right) / 2;
        if (arr[mid] == key)
        {
            flag = 1;
            break;
        }
        else if (key < arr[mid])
        {
            right = mid - 1;
        }
        else if (key > arr[mid])
        {
            left = mid + 1;
        }
    }
```

```
    if (flag == 1)
    {
        cout << mid << endl;
    }
    else
    {
        cout << "Not Found" << endl;
    }
}
int main()
{
    int size;
    cin >> size;
    int arr[size];
    for (int i = 0; i < size; i++)
    {
        cin >> arr[i];
    }
    binarySearch(arr, size);
    return 0;
}
```

Question: 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.**

Marks: 20

Sample Input	Sample Output
7 1 3 4 6 6 9 17 6	YES
10 0 1 2 3 4 5 6 7 8 9 3	NO

Ans :

```
#include <bits/stdc++.h>
using namespace std;
int findFrequency(int arr[], int size, int k)
{
   int l = 0;
   int r = size - 1;
```

```
    int mid;
    while (l != r)
    {
        mid = (l + r) / 2;
        if (arr[mid] == k)
        {
            return 1;
        }
        else if (k < arr[mid])
        {
            r = mid - 1;
        }
        else if (k > arr[mid])
        {
            l = mid + 1;
        }
    }
    return 0;
}
int main()
{
    int size;
    cin >> size;
    int arr[size];
    for (int i = 0; i < size; i++)
    {
        cin >> arr[i];
    }
```

```
    int count = 0;
    int k;
    cin >> k;
    count = count + findFrequency(arr, size, k);
    count = count + findFrequency(arr, size, k);
    if (count > 1)
    {
        cout << "YES" << endl;
    }
    else
    {
        cout << "NO" << endl;
    }
    return 0;
}
```

Calculate the time complexity of the following code snippets.

Marks: 20

(a)

```
int count = 0;
for (int i = n; i > 0; i /= 2)      ---------- O(logN)
{
   for (int j = 0; j < n; j+=5)   ---------- O(N) * O(logN) = O(NlogN)
  {
      count += 1;
   }
}
```

So the time complexity of the above code snippet is = O(NlogN).

(b)

```
for(int i =1; i*i<n; i++)
{
    cout << “hello”;
}
```

i	i*i
1	1*1=1
2	2*2=2
3	3*3=9
4	4*4=16
-	
-	
-	
k	k*k=k^2

Suppose for the k value it crosses n value, so k^2>N
Or k^2=N
=> k = √N
So time complexity of the above code snippet is = O(√N)

(c)

```
for(int i =1; i<n; i=i*2)    ---------- O(logN)
{
   for(int j=1; j*j<n; j+=2) ---------- O(√N) * O(logN) = O(√NlogN)
  {
     cout << “hello”;
   }
}
```

j	j*j	j=j+2

1	1*1=1	1+2=3
3	3*3=9	3+2=5
5	5*5=25	5+2=7
7	7*7=49	7+2=9
9	9*9=81	9+2=11
-		
-		
-		
k	k*k=k^2	

Suppose for the k value it crosses n value, so k^2>N

Or k^2=N

=> k = √N

And i loop's time complexity is O(logN)

So time complexity of the above code snippet is = O(√N) * O(logN) = O(√NlogN)

(d)

```
int m = 1;
for(int i=0; m<=n; i++)
{
   m+=i;
}
```

i	m =m+i
0	0+1=1
1	1+1=2
2	2+2=4
3	4+3=7
4	7+4=11

At k'th iteration -> 1+2+3+4+----------+k=p

When the p>N condition becomes false and loop breaks.
We know that, 1+2+3+4+----------+k = (k(k+1))/2
So, (k(k+1))/2 > N
Or (k(k+1))/2 = N
=> (k^2 + k)/2 = N
=> k^2 + k = 2N
k^2 = N (higher order gets priority and coefficients are not considered)
So, k = √N
So time complexity of the above code snippet is = O(√N)

Question: 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.

You cannot use stl sort function here

Marks: 20

4 8 6 4 2 4 7 5 3 1	8 7 6 5 4 3 2 1

Ans :

```
#include <bits/stdc++.h>
using namespace std;
void merge(int l[], int r[], int size1, int size2)
{
    int n = size1 + size2;
    int arr[n];
    int lp = 0, rp = 0, k = 0;
    while (lp < size1 && rp < size2)
    {
        if (l[lp] >= r[rp])
        {
            arr[k] = l[lp];
            lp++;
            k++;
        }
        else
        {
            arr[k] = r[rp];
            rp++;
            k++;
        }
    }

    while (lp < size1)
    {
        arr[k] = l[lp];
        lp++;
        k++;
    }
    while (rp < size2)
    {
        arr[k] = r[rp];
        rp++;
        k++;
    }
    for (int i = 0; i < n; i++)
    {
```

```
        cout << arr[i] << " ";
    }
}

int main()
{
    int size1;
    cin >> size1;
    int l[size1];
    for (int i = 0; i < size1; i++)
    {
        cin >> l[i];
    }
    int size2;
    cin >> size2;
    int r[size2];
    for (int i = 0; i < size2; i++)
    {
        cin >> r[i];
    }
    merge(l, r, size1, size2);
    return 0;
}
```