**Introduction to Algorithms**

**Module 04: Assignment 01 (Theory)**

Answer Sheet

|  |
| --- |
| **Q1. Write a C++ program that takes N integer numbers and sorts them in**  **non-increasing order using Merge Sort.** |
| Ans:  #include <bits/stdc++.h>  using namespace std;  void merge(int numArr[], int l, int r, int mid)  {      int leftPartSize = mid - l + 1;      int L[leftPartSize + 1];      int rightPartSize = r - mid;      int R[rightPartSize + 1];      for (int i = l, j = 0; i <= mid; i++, j++)      {          L[j] = numArr[i];      }      for (int i = mid + 1, j = 0; i <= r; i++, j++)      {          R[j] = numArr[i];      }      L[leftPartSize] = INT\_MIN;      R[rightPartSize] = INT\_MIN;      int LP\_pointer = 0, RP\_pointer = 0;      for (int i = l; i <= r; i++)      {          if (L[LP\_pointer] >= R[RP\_pointer])          {              numArr[i] = L[LP\_pointer];              LP\_pointer++;          }          else          {              numArr[i] = R[RP\_pointer];              RP\_pointer++;          }      }  };  void merge\_sort(int arr[], int l, int r)  {      if (l == r)      {          return;      }      int mid = (l + r) / 2;      merge\_sort(arr, l, mid);      merge\_sort(arr, mid + 1, r);      merge(arr, l, r, mid);  };  void m\_Sort(int arr[], int l, int r)  {      merge\_sort(arr, l, r);  }  int main()  {      int n;      cin >> n;      int num\_arr[n];      for (int i = 0; i < n; i++)      {          cin >> num\_arr[i];      }      m\_Sort(num\_arr, 0, n - 1);      for (int i = 0; i < n; i++)      {          cout << num\_arr[i] << " ";      }      // \_\_\_\_\_\_      return 0;  } |

|  |
| --- |
| **Q2. 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.** |
| Ans:  #include <bits/stdc++.h>  using namespace std;  void findIndex(int arr[], int val, int l, int r, int mid)  {      if (l > r) {          cout << "Not Found";          return;      }      if (arr[mid] == val) {          cout << mid;          return;      }      else if (arr[mid] < val){          findIndex(arr, val, mid + 1, r, (mid + 1 + r) / 2);      }      else {          findIndex(arr, val, l, mid - 1, (l + mid - 1) / 2);      }  };  int main()  {      int n;      cin >> n;      int arrNum[n];      for (int i = 0; i < n; i++)      {          cin >> arrNum[i];      }      int k;      cin >> k;      int l = 0, r = n - 1;      int mid = (l + r) / 2;      findIndex(arrNum, k, l, r, mid);      return 0;  } |

|  |
| --- |
| Q3. 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. |
| Ans:  #include <bits/stdc++.h>  using namespace std;  void find\_duplicate(int arr[], int val, int l, int r, int mid)  {      if (l > r)      {          return;      }      if (arr[mid] == val)      {          if (arr[mid + 1] == arr[mid] || arr[mid - 1] == arr[mid])          {              cout << "YES";          }          else          {              cout << "NO";          }          return;      }      else if (arr[mid] < val)      {          find\_duplicate(arr, val, mid + 1, r, (mid + 1 + r) / 2);      }      else      {          find\_duplicate(arr, val, l, mid - 1, (l + mid - 1) / 2);      }  };  int main()  {      int n;      cin >> n;      int arrN[n];      for (int i = 0; i < n; i++)      {          cin >> arrN[i];      }      int k;      cin >> k;      int l = 0, r = n - 1;      int mid = (l + r) / 2;      find\_duplicate(arrN, k, l, r, mid);      // \_\_\_\_\_\_\_\_\_\_      return 0;  } |

|  |
| --- |
| Q4**. Calculate the time complexity -** |
| 1. Answer: O (n log n); 2. Answer: O (); 3. Answer: O (log n \* ); 4. Answer: O (); (around) |

|  |
| --- |
| Q5. 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. |
| Ans:  #include <bits/stdc++.h>  using namespace std;  int main()  {      int n;      cin >> n;      int arrFirst[n + 1];      for (int i = 0; i < n; i++)      {          cin >> arrFirst[i];      }      int m;      cin >> m;      int arrSecond[m + 1];      for (int i = 0; i < m; i++)      {          cin >> arrSecond[i];      }      int nS = n + m;      int newArray[nS];      arrFirst[n] = INT\_MIN;      arrSecond[m] = INT\_MIN;      int p1 = 0, p2 = 0;      for (int i = 0; i <= n + m; i++)      {          if (arrSecond[p2] <= arrFirst[p1])          {              newArray[i] = arrFirst[p1];              p1++;          }          else          {              newArray[i] = arrSecond[p2];              p2++;          }      }      for (int i = 0; i < nS; i++)      {          cout << newArray[i] << " ";      }      // \_\_\_\_\_\_      return 0;  } |