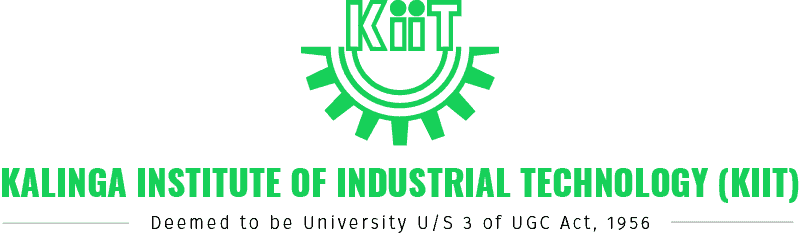
****

**NAME : SOHAM SAMANTA**

**ROLL NUMBER : 20051107**

**SECTION : CSE-14**

**DAA LAB ASSIGNMENT-4**

**4.1** Write a program to search an element x in an array of n integers using binary search algorithm that uses divide and conquer technique. Find out the best case, worst case and average case time complexities for different values of n and plot a graph of the time taken versus n. The n integers can be generated randomly and x can be choosen randomly, or any element of the array or middle or last element of the array depending on type of time complexity analysis.

**Solution:**

//SOHAM SAMANTA CODES

#include<bits/stdc++.h>

using namespace std;

#define ll long long int

#define mod 1000000007

#define PI 3.1415926535897932384626433832

#define ss ios\_base::sync\_with\_stdio(false);cin.tie(NULL);

int bs(int arr[],int k , int n){

int st=0;

int end=n-1;

while(st<=end){

int mid=st+(end-st)/2;

if(k<arr[mid]){

end=mid-1;

} else if(k>arr[mid]){

st=mid+1;

} else{

return mid;

}

}

return -1;

}

int32\_t main(){

ss;

int n,key;

cin>>n>>key;

int arr[n];

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

cin>>arr[i];

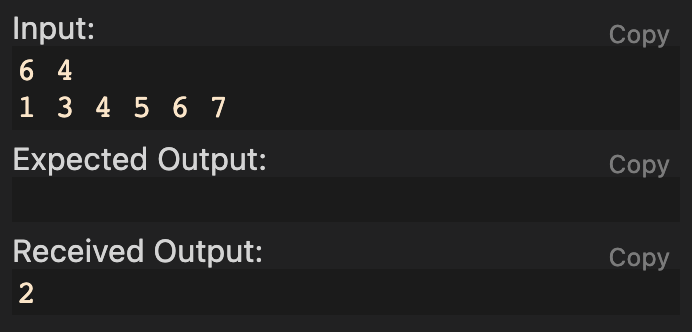
}

cout<<bs(arr,key,n);

return 0;

}

**Output:**

****

**4.2** Write a program to sort a list of n elements using the merge sort method and determine the time required to sort the elements. Repeat the experiment for different values of n and different nature of data (random data, sorted data, reversely sorted data) in the list. n is the user input and n integers can be generated randomly. Finally plot a graph of the time taken versus n

**Solution:**

//SOHAM SAMANTA CODES

#include<bits/stdc++.h>

using namespace std;

#define ll long long int

#define mod 1000000007

#define PI 3.1415926535897932384626433832

#define ss ios\_base::sync\_with\_stdio(false);cin.tie(NULL);

void merge(int array[], int const left, int const mid, int const right){

auto const subArrayOne = mid - left + 1;

auto const subArrayTwo = right - mid;

auto \*leftArray = new int[subArrayOne], \*rightArray = new int[subArrayTwo];

for (auto i = 0; i < subArrayOne; i++)

leftArray[i] = array[left + i];

for (auto j = 0; j < subArrayTwo; j++)

rightArray[j] = array[mid + 1 + j];

auto indexOfSubArrayOne = 0,

indexOfSubArrayTwo = 0;

int indexOfMergedArray = left;

while (indexOfSubArrayOne < subArrayOne

&& indexOfSubArrayTwo < subArrayTwo) {

if (leftArray[indexOfSubArrayOne]

<= rightArray[indexOfSubArrayTwo]) {

array[indexOfMergedArray]

= leftArray[indexOfSubArrayOne];

indexOfSubArrayOne++;

}

else {

array[indexOfMergedArray]

= rightArray[indexOfSubArrayTwo];

indexOfSubArrayTwo++;

}

indexOfMergedArray++;

}

while (indexOfSubArrayOne < subArrayOne) {

array[indexOfMergedArray]

= leftArray[indexOfSubArrayOne];

indexOfSubArrayOne++;

indexOfMergedArray++;

}

while (indexOfSubArrayTwo < subArrayTwo) {

array[indexOfMergedArray]

= rightArray[indexOfSubArrayTwo];

indexOfSubArrayTwo++;

indexOfMergedArray++;

}

delete[] leftArray;

delete[] rightArray;

}

void mergeSort(int array[], int const begin, int const end){

if (begin >= end)

return;

auto mid = begin + (end - begin) / 2;

mergeSort(array, begin, mid);

mergeSort(array, mid + 1, end);

merge(array, begin, mid, end);

}

void printArray(int A[], int size){

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

cout << A[i] << " ";

}

int32\_t main(){

ss;

int arr[] = { 12, 11, 13, 5, 6, 7 };

auto arr\_size = sizeof(arr) / sizeof(arr[0]);

cout << "Given array is \n";

printArray(arr, arr\_size);

mergeSort(arr, 0, arr\_size - 1);

cout << "\nSorted array is \n";

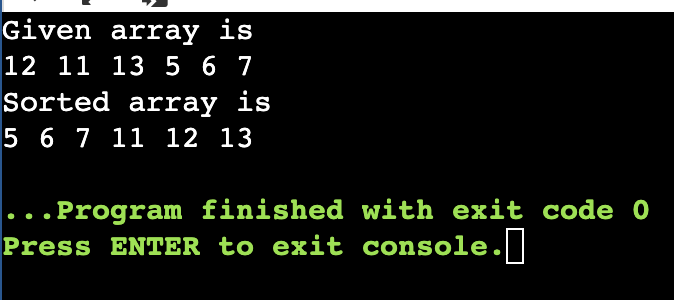
printArray(arr, arr\_size);

return 0;

return 0;

}

**Output:**

****

