**Week 1:-**

1. **Given an array of nonnegative integers, design a linear algorithm and implement it using a program to find whether given key element is present in the array or not. Also, find total number of comparisons for each input case.**

**Code:-**

#include <iostream>

#include <vector>

using namespace std;

void linearSearch(vector<int> &arr, int n,int k){

int iter\_cnt = 0;

for (auto i : arr) {

iter\_cnt++;

if(i == k){

cout << "Present " << iter\_cnt << endl;

return;

}}

cout << "Not present " << iter\_cnt << endl;

}

int main(){

int t, n, k;

cout << "Enter number of test cases : ";

cin >> t;

while (t) {

cout << "Enter size of Array : ";

cin >> n;

vector<int> arr(n);

cout << "Enter elements : ";

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

cin >> arr[i];

cout << "Enter target element : ";

cin >> k;

linearSearch(arr, n, k);

t--;

}

return 0;

}

**Output:-**

**A black screen with white text

Description automatically generated**

1. **Given an already sorted array of positive integers, design an algorithm and implement it using a program to find whether given key element is present in the array or not. Also, find total number of comparisons for each input case.**

**Code:-**

#include <iostream>

#include <vector>

using namespace std;

void binarySearch(vector<int> &arr, int n, int k){

int iter\_cnt = 0, s = 0, e = n - 1;

while (s <= e){

iter\_cnt++;

int mid = s + (e - s) / 2;

if (arr[mid] == k){

cout << "Present " << iter\_cnt << endl;

return;

}

else if (arr[mid] > k)

e = mid - 1;

else

s = mid + 1;

}

cout << "Not present " << iter\_cnt << endl;

}

int main(){

int t, n, k;

cout << "Enter number of test cases : ";

cin >> t;

while (t){

cout << "Enter size of Array : ";

cin >> n;

vector<int> arr(n);

cout << "Enter elements : ";

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

cin >> arr[i];

cout << "Enter target element : ";

cin >> k;

binarySearch(arr, n, k);

t--;

}

return 0;

}

**Output:-**

**A screen shot of a computer

Description automatically generated**

1. **Given an already sorted array of positive integers, design an algorithm and implement it using a program to find whether a given key element is present in the sorted array or not. For an array arr[n], search at the indexes arr[0], arr[2], arr[4],. ,arr[2k] and so on. Once the interval (arr[2k] < key < arr[ 2k+1] ) is found, perform a linear search operation from the index 2k to find the element key.**

**Code:-**

#include <iostream>

#include <vector>

using namespace std;

void linearSearch(vector<int> &arr, int s, int e, int k, int &iter\_cnt)

{

for (int i = s; i <= e; i++)

{

iter\_cnt++;

if (arr[i] == k)

{

cout << "Present " << iter\_cnt << endl;

return;

}

else if (arr[i] > k)

{

break;

}

}

cout << "Not present " << iter\_cnt << endl;

}

void jumpSearch(vector<int> &arr, int n, int k)

{

int iter\_cnt = 0;

if (arr[0] == k)

{

iter\_cnt++;

cout << "Present " << iter\_cnt << endl;

return;

}

if (n >= 1 && arr[1] == k)

{

iter\_cnt++;

cout << "Present " << iter\_cnt << endl;

return;

}

int i = 2;

for (i; i < n; i \*= 2)

{

iter\_cnt++;

if (arr[i] == k)

{

cout << "Present " << iter\_cnt << endl;

return;

}

else if (arr[i] > k)

{

linearSearch(arr, i / 2, i - 1, k, iter\_cnt);

return;

}

}

linearSearch(arr, i / 2, n - 1, k, iter\_cnt);

}

int main()

{

int t, n, k;

cout << "Enter number of test cases : ";

cin >> t;

while (t)

{

cout << "Enter size of Array : ";

cin >> n;

vector<int> arr(n);

cout << "Enter elements : ";

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

{

cin >> arr[i];

}

cout << "Enter target element : ";

cin >> k;

jumpSearch(arr, n, k);

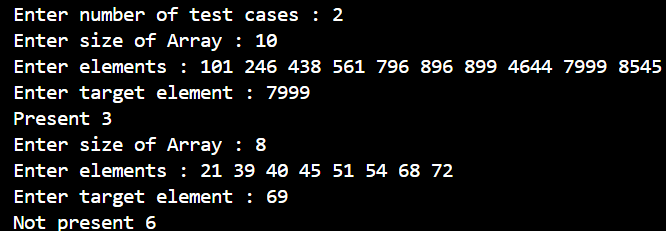
t--;

}

return 0;

}

**Output:-**

****

**Week 2:-**

1. **Given a sorted array of positive integers containing few duplicate elements, design an algorithm and implement it using a program to find whether the given key element is present in the array or not. If present, then also find the number of copies of given key.**

**Code:-**

#include <iostream>

#include <vector>

using namespace std;

int firstOccurence(vector<int> &arr, int n, int k)

{

int s = 0, e = n - 1;

int ans = -1;

while (s <= e)

{

int mid = s + (e - s) / 2;

if (arr[mid] == k)

{

ans = mid;

e = mid - 1;

}

else if (arr[mid] > k)

{

e = mid - 1;

}

else

{

s = mid + 1;

}

}

return ans;

}

int lastOccurence(vector<int> &arr, int n, int k)

{

int s = 0, e = n - 1;

int ans = -1;

while (s <= e)

{

int mid = s + (e - s) / 2;

if (arr[mid] == k)

{

ans = mid;

s = mid + 1;

}

else if (arr[mid] > k)

{

e = mid - 1;

}

else

{

s = mid + 1;

}

}

return ans;

}

void countOccurence(vector<int> &arr, int n, int k)

{

int first\_occur = firstOccurence(arr, n, k);

int last\_occur = lastOccurence(arr, n, k);

int occ = last\_occur - first\_occur + 1;

if (first\_occur >= 0)

{

cout << k << " - " << occ << endl;

}

else

{

cout << "Key not present" << endl;

}

}

int main(){

int t, n, k;

cout << "Enter number of test cases : ";

cin >> t;

while (t){

cout << "Enter size of Array : ";

cin >> n;

vector<int> arr(n);

cout << "Enter elements : ";

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

cin >> arr[i];

}

cout << "Enter target element : ";

cin >> k;

countOccurence(arr, n, k);

t--;

}

return 0;

}

**Output:-**

A screenshot of a computer screen

Description automatically generated

1. **Given a sorted array of positive integers, design an algorithm and implement it using a program to find three indices i, j, k such that arr[i] + arr[j] = arr[k].**

**Code:-**

#include <iostream>

#include <vector>

using namespace std;

bool findTriplets(vector<int> &arr, int n)

{

int k = n - 1;

while (k > 1)

{

int i = 0, j = k - 1;

while (i < j)

{

int sum = arr[i] + arr[j];

if (sum == arr[k])

{

cout << i << "," << j << "," << k << endl;

return true;

}

if (sum > arr[k])

{

j--;

}

else

{

i++;

}

}

k--;

}

return false;

}

int main()

{

int t, n, k;

cout << "Enter number of test cases : ";

cin >> t;

while (t)

{

cout << "Enter size of Array : ";

cin >> n;

vector<int> arr(n);

cout << "Enter elements : ";

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

{

cin >> arr[i];

}

if (!findTriplets(arr, n))

{

cout << "No sequence found\n";

}

t--;

}

return 0;

}

**Output:-**

A black screen with numbers

Description automatically generated

1. **Given an array of nonnegative integers, design an algorithm and a program to count the number of pairs of integers such that their difference is equal to a given key, K.**

**Code:-**

#include <iostream>

#include <vector>

#include <unordered\_map>

using namespace std;

int countPairs(vector<int> &arr, int n, int k){

int ans = 0;

for (int i = 0; i < n - 1; i++) {

for (int j = i + 1; j < n; j++) {

if (abs(arr[i] - arr[j]) == k) {

ans++;

}

}}

return ans;

}

int main(){

int t, n, k;

cout << "Enter number of test cases : ";

cin >> t;

while (t) {

cout << "Enter size of Array : ";

cin >> n;

vector<int> arr(n);

cout << "Enter elements : ";

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

cin >> arr[i];

}

cout << "Enter difference : ";

cin >> k;

cout << countPairs(arr, n, k) << endl;

t--;

}

return 0;

}

**Output:-**

A screen shot of a computer

Description automatically generated

**Week 3:-**

1. **Given an unsorted array of integers, design an algorithm and a program to sort the array using insertion sort. Your program should be able to find number of comparisons and shifts ( shifts - total number of times the array elements are shifted from their place) required for sorting the array.**

**Code:-**

#include <iostream>

#include <vector>

using namespace std;

void insertionSort(vector<int> &arr, int n)

{

int shift = 0, comparison = 0;

for (int i = 1; i < n; i++)

{

int key = arr[i];

int j = i;

while (j > 0 && key < arr[j - 1])

{

shift++;

comparison++;

arr[j] = arr[j - 1];

j--;

}

arr[j] = key;

shift++;

}

cout << "Shifts : " << shift << endl

<< "comparision : " << comparison << endl;

}

int main()

{

int t, n, k;

cout << "Enter number of test cases : ";

cin >> t;

while (t)

{

cout << "Enter size of Array : ";

cin >> n;

vector<int> arr(n);

cout << "Enter elements : ";

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

{

cin >> arr[i];

}

insertionSort(arr, n);

for (auto i : arr)

{

cout << i << " ";

}

cout << endl;

t--;

}

return 0;

}

**Output:-**

A black screen with white numbers

Description automatically generated

1. **Given an unsorted array of integers, design an algorithm and implement a program to sort this array using selection sort. Your program should also find number of comparisons and number of swaps required.**

**Code:-**

#include <iostream>

#include <vector>

using namespace std;

void selectionSort(vector<int> &arr, int n)

{

int swaps = 0, comparison = 0;

for (int i = 0; i < n - 1; i++)

{

int idx = i;

for (int j = i + 1; j < n; j++)

{

comparison++;

if (arr[j] < arr[idx])

idx = j;

}

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

swaps++;

}

cout << "Swaps : " << swaps << endl

<< "comparision : " << comparison << endl;

}

int main()

{

int t, n, k;

cout << "Enter number of test cases : ";

cin >> t;

while (t)

{

cout << "Enter size of Array : ";

cin >> n;

vector<int> arr(n);

cout << "Enter elements : ";

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

{

cin >> arr[i];

}

selectionSort(arr, n);

for (auto i : arr)

{

cout << i << " ";

}

cout << endl;

t--;

}

return 0;

}

**Output:-**

A black screen with white numbers

Description automatically generated

1. **Given an unsorted array of positive integers, design an algorithm and implement it using a program to find whether there are any duplicate elements in the array or not.**

**Code:-**

#include <iostream>

#include <vector>

using namespace std;

void merge(vector<int> &arr, int s, int e, int mid)

{

vector<int> temp;

int idx1 = s, idx2 = mid + 1;

while (idx1 <= mid && idx2 <= e)

{

if (arr[idx1] <= arr[idx2])

{

temp.push\_back(arr[idx1++]);

}

else

{

temp.push\_back(arr[idx2++]);

}

}

while (idx1 <= mid)

{

temp.push\_back(arr[idx1++]);

}

while (idx2 <= e)

{

temp.push\_back(arr[idx2++]);

}

for (int i = 0; i < e - s + 1; i++)

{

arr[s + i] = temp[i];

}

}

void mergeSort(vector<int> &arr, int s, int e)

{

if (s >= e)

{

return;

}

int mid = s + (e - s) / 2;

mergeSort(arr, mid + 1, e);

mergeSort(arr, s, mid);

merge(arr, s, e, mid);

}

int main()

{

int t, n, k;

cout << "Enter number of test cases : ";

cin >> t;

while (t)

{

cout << "Enter size of Array : ";

cin >> n;

vector<int> arr(n);

cout << "Enter elements : ";

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

{

cin >> arr[i];

}

mergeSort(arr, 0, n - 1);

int j = 1, flag = 0;

for (int i = 0; i < n - 1; i++)

{

if (arr[i] == arr[j])

{

flag = 1;

break;

}

j++;

}

if (flag)

cout << "YES" << endl;

else

cout << "NO" << endl;

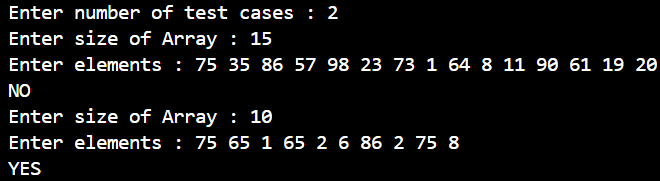
t--;

}

return 0;

}

**Output:-**



**Week 4:-**

1. **Given an unsorted array of integers, design an algorithm and implement it using a program to sort an array of elements by dividing the array into two subarrays and combining these subarrays after sorting each one of them. Your program should also find number of comparisons and inversions during sorting the array.**

**Code:-**

#include <iostream>

#include <vector>

using namespace std;

void merge(vector<int> &arr, int s, int mid, int e, int &comp)

{

vector<int> temp;

int i = s, j = mid + 1;

while (i <= mid && j <= e)

{

comp++;

if (arr[i] < arr[j])

{

temp.push\_back(arr[i++]);

}

else

{

temp.push\_back(arr[j++]);

}

}

while (i <= mid)

{

temp.push\_back(arr[i++]);

}

while (j <= e)

{

temp.push\_back(arr[j++]);

}

for (int i = 0; i < temp.size(); i++)

{

arr[s + i] = temp[i];

}

}

void mergerSort(vector<int> &arr, int s, int e, int &comp)

{

if (s >= e)

return;

int mid = s + (e - s) / 2;

mergerSort(arr, s, mid,comp);

mergerSort(arr, mid + 1, e,comp);

merge(arr, s, mid, e, comp);

}

int main()

{

int t, n, k;

cout << "Enter number of test cases : ";

cin >> t;

while (t)

{

cout << "Enter size of Array : ";

cin >> n;

vector<int> arr(n);

cout << "Enter elements : ";

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

{

cin >> arr[i];

}

int comp = 0 ;

mergerSort(arr, 0, n - 1, comp);

for (auto i : arr)

{

cout << i << " ";

}

cout << endl;

cout << "Comparision : " << comp << endl;

t--;

}

}

**Output:-**

A screenshot of a computer screen

Description automatically generated

1. **Given an unsorted array of integers, design an algorithm and implement it using a program to sort an array of elements by partitioning the array into two subarrays based on a pivot element such that one of the sub array holds values smaller than the pivot element while another sub array holds values greater than the pivot element.**

**Code:-**

#include <iostream>

#include <cstdlib>

#include <vector>

using namespace std;

int partition(vector<int> &arr, int s, int e, int &comp, int &swaps)

{

int n = e - s + 1;

int pivot = s + rand() % n;

int cnt = 0;

for (int k = s; k <= e; k++)

{

if (k != pivot && arr[pivot] >= arr[k])

{

comp++;

cnt++;

}

}

int pIdx = s + cnt;

swaps++;

swap(arr[pivot], arr[pIdx]);

int i = s, j = e;

while (i < pIdx && j > pIdx)

{

while (arr[i] < arr[pIdx])

{

comp++;

i++;

}

while (arr[j] > arr[pIdx])

{

comp++;

j--;

}

if (i < pIdx && j > pIdx)

{

comp++;

swaps++;

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

}

}

return pIdx;

}

void quickSort(vector<int> &arr, int s, int e, int &comp, int &swaps)

{

if (s >= e)

return;

int p = partition(arr, s, e, comp, swaps);

quickSort(arr, s, p - 1, comp, swaps);

quickSort(arr, p + 1, e, comp, swaps);

}

int main()

{

int t, n;

cout << "Enter number of test cases : ";

cin >> t;

while (t)

{

cout << "Enter size of Array : ";

cin >> n;

vector<int> arr(n);

cout << "Enter elements : ";

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

{

cin >> arr[i];

}

int comp = 0, swaps = 0;

quickSort(arr, 0, n - 1, comp, swaps);

for (auto i : arr)

{

cout << i << " ";

}

cout << endl;

cout << "Comparision : " << comp << endl;

cout << "Swaps : " << swaps << endl;

t--;

}

}

**Output:-**

A black screen with white numbers

Description automatically generated

1. **Given an unsorted array of integers, design an algorithm and implement it using a program to find Kth smallest or largest element in the array.**

**Code:-**

#include <iostream>

#include <vector>

using namespace std;

int partition(vector<int> &arr, int s, int e)

{

int cnt = 0;

for (int k = s + 1; k <= e; k++)

{

if (arr[s] > arr[k])

{

cnt++;

}

}

int pIdx = s + cnt;

swap(arr[s], arr[pIdx]);

int i = s, j = e;

while (i < pIdx && j > pIdx)

{

while (arr[i] < arr[pIdx])

{

i++;

}

while (arr[j] > arr[pIdx])

{

j--;

}

if (i < pIdx && j > pIdx)

{

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

}

}

return pIdx;

}

int kthLargest(vector<int> &arr, int s, int e, int k, int n)

{

if (k >= 0 && k < n)

{

int p = partition(arr, s, e);

if (p == k)

return arr[p];

else if (p > k)

return kthLargest(arr, s, p - 1, k, n);

else

return kthLargest(arr, p + 1, e, k, n);

}

else

return -1;

}

int main()

{

int t, n, k;

cout << "Enter number of test cases : ";

cin >> t;

while (t)

{

cout << "Enter size of Array : ";

cin >> n;

vector<int> arr(n);

cout << "Enter K : ";

cin >> k;

cout << "Enter elements : ";

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

{

cin >> arr[i];

}

int kthlargest = kthLargest(arr, 0, n - 1, n - k, n);

cout << "Kth Largest : " << kthlargest << endl;

int kthsmallest = kthLargest(arr, 0, n - 1, k - 1, n);

cout << "Kth Smallest : " << kthsmallest << endl;

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

{

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

}

cout << endl;

t--;

}

}

**Output:-**

**A screenshot of a computer screen

Description automatically generated**