1.Linear Sea­­rch:

ALGORITHM:- Search linearly in the given array for required element & return TRUE or False based on whether the element is present or not.

linearsearch(int arr[],int n,int x,int comparisons){

for(i=0 to n){

comparisons++;

if(arr[i]==x)return i;

}

return -1;

}

#include <iostream>

#include <stdio.h>

**using** **namespace** std;

**int** main()

{

**int** a[100],n;

scanf("%d",&n);

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

scanf("%d",&a[i]);

**int** key,comp=0,flag=0;

scanf("%d",&key);

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

comp++;

**if**(a[i]==key){

printf("key found\n");

printf("index is : %d\n",i);

flag=1;

**break**;

}

}

**if**(flag==0){

printf("key not found\n");

}

printf("Total comparisons : %d\n",comp);

**return** 0;

}

**OUTPUT:**

5

1

2

3

4

5

3

**key found**

**index is : 2**

**Total comparisons : 1**

**Program ended with exit code: 0**

2.Binary Search:

ALGORITHM: Searc for the element by repeatedly dividing the array into two parts .

binarysearch(int a[],int l,int r,int key){

int cmp=0;

while(l<=r){

mid=(l+r)/2;

cmp++;

if(a[mid]==key)

print(“Present”,cmp);

else if(a[mid]<key)

l=mid+1;

else r=mid-1;

}

print(“Not Present”,cmp);

}

#include <stdio.h>

**int** main()

{

**int** a[100],n;

scanf("%d",&n);

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

scanf("%d",&a[i]);

**int** key,comp=0,flag=0;

scanf("%d",&key);

**int** start=0,end=n-1,mid;

mid=(start+end)/2;

**while**(start<=end){

comp++;

**if**(a[mid]==key){

printf("key found\n");

printf("index is : %d\n",mid);

flag=1;

**break**;

}

**else** **if**(a[mid]>key)

end=mid-1;

**else**

start=mid+1;

mid=(start+end)/2;

}

**if**(flag==0){

printf("key not found\n");

}

printf("Total comparisons : %d\n",comp);

}

**OUTPUT:**

5

1

2

3

4

5

3

**key found**

**index is : 2**

**Total comparisons : 1**

**Program ended with exit code: 0**

3.Jump Search:

Algorithm: Using this method, we jump with a step of root-n rather than 1. It helps to reduce the time complexity, in certain interval we do a linear search[O(root-n)].

JumpSearch(int a[], int n, int key){

Jump = sqrt(n);

Int cmp;

Prev = 0, strt = 0;

While(true){

cmp++;

if(step<n && a[strt]==key){return strt;}

else if(strt<n && a[strt]<key){strt+=Jump;}

else {

Prev=strt-jmp;

While(prev<n && prev<=strt){

If(a[prev]==key)return prev;

else prev++;

cmp++;

}

Return -1;

}

}

Return -1;}

|  |
| --- |
| #include <iostream> |
|  |

|  |
| --- |
| #include <algorithm> |
|  |

|  |
| --- |
| #include <vector> |
|  |

|  |
| --- |
| #include <cmath> |
|  |

|  |
| --- |
| using namespace std; |
|  |

|  |
| --- |
| int cmp = 0; |
|  |

|  |
| --- |
| int jumpsearch(int a[], int n, int key) { |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| int jmp = sqrt(n), strt = 0; |
|  |

|  |
| --- |
| while (true) { |
|  |

|  |
| --- |
| cmp++; |
|  |

|  |
| --- |
| if (strt < n && a[strt] == key) {return strt;} |
|  |

|  |
| --- |
| else if (strt < n && a[strt] < key) { |
|  |

|  |
| --- |
| strt += jmp; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| else { |
|  |

|  |
| --- |
| int prev = strt - jmp; |
|  |

|  |
| --- |
| while (prev < n && prev <= strt) { |
|  |

|  |
| --- |
| if (a[prev] == key)return prev; |
|  |

|  |
| --- |
| else prev++; |
|  |

|  |
| --- |
| cmp++; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| return -1; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| return -1; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| void solve() { |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| int n; cin >> n; |
|  |

|  |
| --- |
| int a[n]; |
|  |

|  |
| --- |
| for (int &i : a)cin >> i; |
|  |

|  |
| --- |
| int key; cin >> key; |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| sort(a, a + n); |
|  |

|  |
| --- |
| cmp = 0; |
|  |

|  |
| --- |
| int ans = jumpsearch(a, n, key); |
|  |

|  |
| --- |
| if (ans == -1)cout << "Not Present"<<cmp << endl; |
|  |

|  |
| --- |
| else { |
|  |

|  |
| --- |
| cout << "Present" <<cmp << endl; |
|  |

|  |
| --- |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| int main() { |
|  |

|  |
| --- |
| int t; cin >> t; |
|  |

|  |
| --- |
| while (t--) { |
|  |

|  |
| --- |
| solve(); |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| return 0; |
|  |

}

**OUTPUT:**

3

5

12 23 36 39 41

41

Present3

8

21 39 40 45 51 54 68 72

69

Not Present4

10

101 246 438 561 796 896 899 4644 7999 8545

7999

Present3

4. Number of Occurences of an element in sorted array

Algorithm: since array is sorted, a binary search can be used to find the element and its left and right size can be checked for duplicates.

Occurences(int a[],int n,int x){

Pos = binarySearch(a, 0, n-1, x);

J= pos+1;

Count++;

While(a[j]==x){ j++; count++;}

J=pos-1;

While(a[j] == x) {j--; count++;}

Print(x,”occurs ”,count,” Times”);

}

#include<iostream>

using namespace std;

int main()

{

int t;

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

cin>>t;

while(t--)

{

int n;

cout<<"Enter the size of the array: ";

cin>>n;

cout<<"Enter "<<n<<" Elements in the array"<<"\n";

int arr[n];

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

{

cin>>arr[i];

}

int key;

cout<<"Enter the key to search: ";

cin>>key;

int j=0,count=0, = 0, pos=0,r = n;

while(l<=r)

{

int mid = l + (r-l)/2;

if(arr[mid]>key)

r = mid-1;

else if(arr[mid]<key)

l = mid+1;

else

{

pos=mid;

break;

}

}

If(arr[pos]==key){count++;

J=pos+1;

while(j<n && arr[j] == arr[pos])

{j++;count++;}

J=pos-1;

while(j>=0 && arr[j] == arr[pos])

{j--;count++;}

cout<<key<<" occurs "<<count<<" times"<<"\n";

}

else cout<<”NOT FOUND”<<endl;

}

return 0;

}

**OUTPUT:**

Enter the number of test cases: 2

Enter the size of the array: 10

Enter 10 Elements in the array

235 235 278 278 763 764 790 853 981 981

Enter the key to search: 981

981 occurs 2 times

Enter the size of the array: 15

Enter 15 Elements in the array

1 2 2 3 3 5 5 5 25 75 75 75 97 97 97

Enter the key to search: 75

75 occurs 3 times

5. Two Numbers having sum equal to third number.

Algorithm: Since the array is sorted, using three for loops can be efficient method.

Equal\_sum(int arr[],int n){

For(i=0 to n-2; i++){

For(j=i+1 to n-1; j++){

For(k=j+1 to n;k++{

If(arr[i]+arr[j]==arr[k]){

print(i+1, j+1, k+1);return;

}

}

}

}

Print(“No sequence Found”);

}

#include<iostream>

using namespace std;

void equal\_sum(int arr[],int n)

{

for(int i=0;i<n-2;i++)

{

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

{

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

{

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

{

cout<<i+1<<j+1<<k+1<<"\n";

return;

}

}

}

}

cout<<"no sequence present\n";

}

int main()

{

int t;

cin>>t;

while(t--)

{

int n;

cin>>n;

int arr[n];

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

{

cin>>arr[i];

}

Equal\_sum(arr,n);

}

}

6 .Pair having difference equal to key :

Algorithm : This can be done by using linear search on each element a[i] to find(a[i]+key).

Equal\_difference(int arr[],int n){

Int Count=0;

For(i=0 to n-2; i++){

For(j=i+1 to n-1; j++){

For(k=j+1 to n;k++{

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

Count++;

}

}

}

}

Return Count;

}

#include<iostream>

#include<cmath>

using namespace std;

void equal\_difference(int arr[],int n,int key)

{

int count=0;

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

{

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

{

if( abs(arr[i]-arr[j])==key)

{

count++;

}

}

}

if(count==0)

{

cout<<"no pair found\n";

}

else

{

cout<<count<<"pairs that are equal to key\n";

}

}

int main()

{

int t;

cin>>t;

while(t--)

{

int n;

cin>>n;

int arr[n];

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

{

cin>>arr[i];

}

int key;

cin>>key;

pair\_count(arr,n,key);

}

}

**OUTPUT:**

Input:

2

5

1 51 84 21 31

20

Output: 2

Input:

10

24 71 16 92 12 28 48 14 20 22

4

Output: 4

8.Insertion Sort :

Algorithm: Insertion sort works by inserting new element

or element of an array into an already sorted subarray. [O(n^2)].

InsertionSort(int a[], int n){

int comp=0, int Shift=0;

For(i=1 to n; i++){

Key=a[i];

J=i-1;

While(j>=0 && key[a[j]]{

Shift++; comp++;

A[j+1] = a[j];

j--;

}

Shift++;

A[j+1] = key;

}

Print(Shift);

Print(Comp);

}

#include<iostream>

using namespace std;

void insertion(int arr[],int n)

{

int cmp=0;

int shift=0;

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

{

int temp=arr[i];

int j=i-1;

while(j>=0 &&arr[j]>temp)

{

cmp++;

shift++;

arr[j+1]=arr[j];

j--;

}

shift++;

arr[j+1]=temp;}

cout<<shift<<endl;

cout<<cmp<<endl;

}

int main()

{

int t;

cin>>t;

while(t--)

{

int n;

cin>>n;

int arr[n];

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

{

cin>>arr[i];

}

insertion(arr,n);

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

{

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

}

}

}

**OUTPUT:**

3

8

-23 65 -31 76 46 89 45 32

Shift= 20

Comparions=13

-31 -23 32 45 46 65 76 89

10

54 65 34 76 78 97 46 32 51 21

Shift= 37

Comparions=28

21 32 34 46 51 54 65 76 78 97

15

63 42 223 645 652 31 324 22 533 -12 54 65 86 46 325

Shift= 68

Comparions=54

-12 22 31 42 46 54 63 65 86 223 324 325 533 645 652

9.Selection Sort:

Algorithm: In this algorithm, minimum element is found in the successive decreasing subarray and placed at the first position of the array(O(logn)).

SelectionSort(int a[], int n){

Int c=0,s=0;

For(p = 1 to n; i++){

Min = a[i-1];

loc = i-1;

For(j=I to n; j++){

If(a[j] < min) min = a[j]; loc= j;

c++;

}

s++;

A[loc] = A[i-1];

A[i-1] = min;

}

Print(s);

Print(c);

}

#include<iostream>

using namespace std;

void selection(int arr[], int n)

{

int mdx,cmp=0,swaps=0;

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

{

mdx=i;

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

{

cmp++;

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

{

mdx=j;

}

}

swaps++;

if(mdx!=i)

{

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

}

}

cout<<swaps<<" \n";

cout<<cmp<<"\n";

}

int main()

{

int t;

cin>>t;

while(t--)

{

int n;

cin>>n;

int arr[n];

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

{

cin>>arr[i];

}

selection(arr,n);

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

{

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

}

}

}

**OUTPUT:**

3

8

-13 65 -21 76 45 89 45 12

Swaps=7

comparison=28

-21 -13 12 45 45 65 76 89

10

54 65 34 76 78 97 46 32 51 21

Swaps=9

comparison=45

21 32 34 46 51 54 65 76 78 97

15

63 42 223 645 652 31 324 22 553 12 54 65 86 46 325

Swaps=14

comparison=105

12 22 31 42 46 54 63 65 86 223 324 325 553 645 652

10.Checking for duplicates in unsorted array.

ALGORITHM: Sort the array using any sorting technique and then check adjacent element of the array for redundancy.

mergesort(arr,0,n-1)

for(i=0 to n){

if(i+1<n && i-1>=0){

if(a[i]==a[i-1] || a[i]==a[i+1]){

print("YES");exit(0);

}

}

}

print("NO");

#include<iostream>

using namespace std;

void merge(int arr[],int l, int m, int r)

{

int n1=m-l+1;

int n2=r-m;

int L[n1];

int R[n2];

for(int i=0;i<n1;i++)

{

L[i]=arr[l+i];

}

for(int j=0;j<n2;j++)

{

R[j]=arr[m+1+j];

}

int i=0,j=0,k=l;

while(i<n1&&j<n2)

{

if(L[i]<=R[j])

arr[k++]=L[i++];

else

arr[k++]=R[j++];

}

while(i<n1)

arr[k++]=L[i++];

while(j<n2)

arr[k++]=R[j++];

}

void mergeSort(int arr[],int l,int r)

{

if(l<r)

{

int m=l+(r-l)/2;

mergeSort(arr,l,m);

mergeSort(arr,m+1,r);

merge(arr,l,m,r);

}

}

int main()

{

int t;

cin>>t;

while(t--)

{

int n,flag=0;

cin>>n;

int arr[n];

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

{

cin>>arr[i];

}

mergeSort(arr,0,n-1);

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

if(i+1<n && i-1>=0){

if(a[i]==a[i-1] || a[i]==a[i+1]){

cout<<”YES”<<endl;return;

}

}

}

Cout<<”NO”<<endl;

}

}

**OUTPUT:**

3

5

28 52 83 14 75

NO

10

75 65 1 65 2 6 86 2 75 8

Yes

15

75 35 86 57 98 23 73 1 64 8 11 90 61 19 20

NO

11 Inversions:

ALGORITHM: - Merge function of mergesort can be used to find the number of inversions.

merge(int arr[],int temp[], left, mid, right) {

i = left, j = mid, k = 0;

inversions = 0;

while (i <= mid - 1 && j <= right) {

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

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

else {

temp[k++] = arr[j++];

inversion = inversions + (mid - i);

}

}

while (i <= mid - 1) {

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

}

while (i <= right) {

temp[k++] = arr[j++];

}

return inversions;

}

mergesort(arr[], temp[], begin, end) {

inversion = 0;

if (end > begin) {

mid = (begin + end) / 2;

inversions += mergesort(arr, temp, begin, mid);

inversions += mergesort(arr, temp, mid + 1, end);

inversions += merge(arr, temp, begin, mid + 1, end);

}

return inversions;

}

#include<iostream>

using namespace std;

int merge(int arr[],int l, int m, int r)

{

int inversions=0;

int n1=m-l+1;

int n2=r-m;

int L[n1];

int R[n2];

for(int i=0;i<n1;i++)

{

L[i]=arr[l+i];

}

for(int j=0;j<n2;j++)

{

R[j]=arr[m+1+j];

}

int i=0,j=0,k=l;

while(i<n1&&j<n2)

{

if(L[i]<=R[j])

arr[k++]=L[i++];

else

{

arr[k++]=R[j++];

inversions = inversions + (n1 - i);

}

}

while(i<n1)

arr[k++]=L[i++];

while(j<n2)

arr[k++]=R[j++];

return inversions;

}

int mergeSort(int arr[],int l,int r)

{

int inversions=0;

if(l<r)

{

int m=l+(r-l)/2;

inversions +=mergeSort(arr,l,m);

inversions +=mergeSort(arr,m+1,r);

inversions += merge(arr,l,m,r);

}

return inversions;

}

int main()

{

int t;

cin>>t;

while(t--)

{

::inv=0;

::cmp=0;

int n,flag=0;

cin>>n;

int arr[n];

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

{

cin>>arr[i];

}

int inv=mergeSort(arr,0,n-1);

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

{

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

}

cout<<"\nInversion="<<inv<<endl;

cout<<"comparion="<<cmp<<endl;

}

}

**OUTPUT:**

**3**

**8**

**23 65 21 76 46 89 45 32**

**21 23 32 45 46 65 76 89**

**Inversion=13**

**comparion=16**

**10**

**54 65 34 76 78 97 46 32 51 21**

**21 32 34 46 51 54 65 76 78 97**

**Inversion=28**

**comparion=22**

**15**

**63 42 223 645 652 31 324 22 553 12 54 65 86 46 325**

**12 22 31 42 46 54 63 65 86 223 324 325 553 645 652**

**Inversion=54**

**comparion=43**

12. Randomized Quick Sort

ALGORITHM:Instead of using first element for pivot the first element is randomly swapped with another element & rest is normal quicksort.

partition(arr[],low,high){

pivot=arr[high];

i=low-1;

for(j=low to high){

if(arr[j]<=pivot){

i++;

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

}

}

swap(arr[i+1],arr[high]);

return i+1;

}

partition\_r(arr[],low,high){

srand(time(NULL));

random=low+rand()%(high-low);

swap(arr[random],arr[high]);

return partition(arr,low,high);

}

quicksort(arr[],low,high){

if(low<high){

pi=partition\_r(arr,low,high);

quicksort(arr,low,pi-1);

quicksort(arr,pi+1,high);

}

}

#include <cstdlib>

#include <time.h>

#include <iostream>

using namespace std;

 int swaps=0;

int comp=0;

int partition(int arr[], int low, int high)

{

    int pivot = arr[high];

    int i = (low - 1);

    for (int j = low; j <= high - 1; j++)

    {

comp++;

        if (arr[j] <= pivot) {

             i++;

            swap(arr[i], arr[j]);

swaps++;

        }

    }

swaps++;

    swap(arr[i + 1], arr[high]);

    return (i + 1);

}

int partition\_r(int arr[], int low, int high)

{

    srand(time(NULL));

    int random = low + rand() % (high - low);

    swap(arr[random], arr[high]);

    return partition(arr, low, high);

}

void quickSort(int arr[], int low, int high)

{

    if (low < high) {

        int pi = partition\_r(arr, low, high);

        quickSort(arr, low, pi - 1);

        quickSort(arr, pi + 1, high);

    }

}

int main()

{

int t;

cin>>t;

while(t--)

{

int n;

cin>>n;

int arr[n];

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

{

cin>>arr[i];

}

quickSort(arr,0,n-1);

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

{

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

}

cout<<"\nSwaps="<<swaps<<endl;

cout<<"comparion="<<comp<<endl;

}

}

int main()

{

    int arr[] = { 10, 7, 8, 9, 1, 5 };

    int n = sizeof(arr) / sizeof(arr[0]);

    quickSort(arr, 0, n - 1);

    printf("Sorted array: \n");

    printArray(arr, n);

    return 0;

}

**OUPUT:**

3

8

23 65 21 76 46 89 45 32

21 23 32 45 46 65 76 89

Swaps=10

Comparions=14

10

54 65 34 76 78 97 46 32 51 21

21 32 34 46 51 54 65 76 78 97

Swaps=21

Comparions=29

15

63 42 223 645 652 31 324 22 553 12 54 65 86 46 325

12 22 31 42 46 54 63 65 86 223 324 325 553 645 652

Swaps=39

Comparions=45

13.Kth Smallest:-

ALGORITHM:-

The idea is to sort the array using any sorting algorithm then find kth smallest element from beginning. O(nlogn)

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

mergesort(a,0,n);

Print(“Kth Smallest element is”,arr[k-1]);

}

#include<iostream>

using namespace std;

void merge(int arr[], int l, int m, int r)

{

int n1 = m - l + 1,n2=r-m;

int L[n1],R[n2];

for (int i = 0; i < n1; i++){

L[i] = arr[l + i]; }

for (int j = 0; j < n2; j++){

R[j] = arr[m + 1 + j]; }

int i = 0, j = 0, k = l;

while (i < n1 && j < n2){

if (L[i] <= R[j])

arr[k++] = L[i++];

else

arr[k++] = R[j++];

}

while (i < n1) {arr[k++] = L[i++];}

while (j < n2) {arr[k++] = R[j++];}

}

void mergeSort(int arr[], int l, int r)

{

if (l < r)

{

int m = l + (r - l) / 2;

mergeSort(arr, l, m);

mergeSort(arr, m + 1, r);

merge(arr, l, m, r);

}

}

int main()

{

int t;

cin >> t;

while (t--)

{

int n, flag = 0;

cin >> n;

int arr[n];

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

{

cin >> arr[i];

}

int k; cin >> k;

mergeSort(arr, 0, n - 1);

cout << " Kth smallest element is " << arr[k - 1] << endl;

}

return 0;

}

**OUPUT:**

Input:  
1

10  
123 656 54 765 344 514 765 34 765 234  
3

Output: 12314. Element with maximum occurrences

ALGORITHM: -

Using counting sort to get the frequency of each alphabet and finding maximum among them.

countingsort(string s) {

n = s.length();

char arr[n];

x = 0;

for (auto i : s){arr[x++] = i;}

count[26];

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

count[arr[i] - 'a']++;

}

char l = 'a';

num = 0;

for (i = 0 to 26) {

if (num < count[i]) {

num = count[i]; l = l + i;

}

}

if (num <= 1)print("NO DUPLICATES");

else print(l, "-", num);

}

#include<iostream>

using namespace std;

void countingsort(char s[], int n) {

char arr[n] = {0};

int x = 0;

for (int i = 0; i < n; i++) {arr[x++] = s[i];}

int count[26] = {0};

for (int i = 0; i < n; i++) {count[arr[i] - 'a']++;}

char l = 'a';

int num = 0;

for (int i = 0; i < 26; i++) {

if (num < count[i]) {

num = count[i];

l = l + i;

}

}

if (num <= 1)cout << "NO Duplicates Found " << endl;

else cout << l << "->" << num << endl;

}

int main()

{

int t; cin >> t;

while (t--) {

int n; cin >> n;

char a[n];

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

cin >> a[i];

}

countingsort(a, n);

}

return 0;

}

**OUTPUT:**

3

10

a e d w a d q a f p

a-> 3

15

r k p g v y u m q a d j c z e

No Duplicates Present

20

g t l l t c w a w g l c w d s a a v c l

l-> 4

15.Sum equal to target

ALGORITHM: - First sort the array, then use two pointers at start & end.Keep chcking if sum of those elements is equal to target or less or greater. change the pointers accordingly.

func(int a[],int n,int target){

mergesort(a, 0, n - 1);

i = 0, j = n - 1;

whlie(i < j) {

currsum = a[i] + a[j];

if (currsum < target)

i++;

else if (currsum > target)

j--;

else {print(a[i], a[j]); exit(0); }

}

Print(“No Such element exist”);

}

#include<iostream>

using namespace std;

void merge(int arr[], int l, int m, int r)

{

int n1 = m - l + 1;

int n2 = r - m;

int L[n1];

int R[n2];

for (int i = 0; i < n1; i++)

{

L[i] = arr[l + i];

}

for (int j = 0; j < n2; j++)

{

R[j] = arr[m + 1 + j];

}

int i = 0, j = 0, k = l;

while (i < n1 && j < n2)

{

if (L[i] <= R[j])

arr[k++] = L[i++];

else

arr[k++] = R[j++];

}

while (i < n1)

arr[k++] = L[i++];

while (j < n2)

arr[k++] = R[j++];

}

void mergeSort(int arr[], int l, int r)

{

if (l < r)

{

int m = l + (r - l) / 2;

mergeSort(arr, l, m);

mergeSort(arr, m + 1, r);

merge(arr, l, m, r);

}

}

int main()

{

int t;

cin >> t;

while (t--)

{

int n, key, flag = 0;

cin >> n;

int a[n];

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

{

cin >> a[i];

}

cin >> key;

mergeSort(a, 0, n - 1);

int i = 0, j = n - 1, currsum = 0;

while (i < j) {

currsum = a[i] + a[j];

if (currsum < key)

i++;

else if (currsum > key)

j--;

else {

cout << a[i] << " " << a[j] << endl;

flag = 1;

break;

}

}

if (flag == 0)

cout << "No such element exist" << endl;

}

return 0;

}

**OUPUT:**

2

10

64 28 97 40 12 72 84 24 38 10

50

10 40

15

56 10 72 91 29 3 41 45 61 20 11 39 9 12 94

302

No such element exist

16.Intersection of two sorted arrays.

ALGORITHM:- Similar to merging of two sorted arrays we will only print those elements which are common to both arrays.

print\_intersection(int arr1[],n,int arr2[],m){

i=j=0;

while(i<m && j<m){

if(arr1[i]<arr2[j]){

i++;

}

else if(arr2[j]<arr1[i]){

j++;

}

else {

print(arr[j]);

i++;

j++;

}

}

}

#include <iostream>

using namespace std;

void print\_intersection(int a[], int n, int b[], int m) {

int i = 0;

int j = 0;

while (i < n && j < m) {

if (a[i] < b[j])i++;

else if (a[i] > b[j])j++;

else {cout << a[i] << " ";

i++;j++;}

}

}

int main() {

int t; cin >> t;

while (t--) {

int n; cin >> n;

int a[n];

for (int i = 0; i < n; i++)cin >> a[i];

int m; cin >> m;

int b[m];

for (int i = 0; i < n; i++)cin >> b[i];

print\_intersection(a, n, b, m);

}

}

**OUPUT:**

**Input:**

1

5

1 2 3 4 5

6

3 4 5 6 7 7

**Output: 3 4 5**