

## WEEK1

**Q1) 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. (Time Complexity =  $O(n)$ , where  $n$  is the size of input).**

```
#include <bits/stdc++.h>
using namespace std;
int main()
{
    int t;
    cin>>t;
    while(t--)
    {
        int n, key;
        cin>>n;
        int arr[n];
        for(int i = 0; i < n; i++)
        {
            cin>>arr[i];
        }
        cin>>key;
        int j;
        for(j = 0; j < n; j++)
        {
            if(arr[j] == key)
            {
                cout<<"Present "<<j+1<<"\n";
                break;
            }
        }
        if(j == n)
        {
            cout<<"Not Present "<<j<<"\n";
        }
    }
}
```

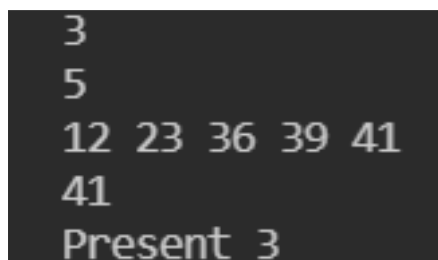
## OUTPUT

```
3
8
34 35 65 31 25 89 64 30
89
Present 6
```

**Q2) 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. (Time Complexity =  $O(n \log n)$ , where  $n$  is the size of input).**

```
#include <bits/stdc++.h>
using namespace std;
int main()
{
    int t;
    cin>>t;
    while(t--)
    {
        int n, key;
        cin>>n;
        int arr[n];
        for(int i = 0; i < n; i++)
        {
            cin>>arr[i];
        }
        cin>>key;
        int l = 0, h = n - 1;
        int i = 0, flag = 0;
        while(l <= h)
        {
            int mid = (l + h) / 2;
            i++;
            if(arr[mid] == key)
            {
                flag = 1;
                cout<<"Present "<<i<<endl;
                break;
            }
            else if(arr[mid] > key) h = mid - 1;
            else l = mid + 1;
        }
        if(!flag) cout<<"Present "<<n/2<<endl;
    }
    return 0;
}
```

### **OUTPUT**

A screenshot of a terminal window showing the output of the program. The input sequence is 3, 5, 12 23 36 39 41, 41. The output is "Present 3".

```
3
5
12 23 36 39 41
41
Present 3
```

**Q3) 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. (Complexity  $< O(n)$ , where  $n$  is the number of elements need to be scanned for searching):**

### **Jump Search**

#### **Input format:**

The first line contains number of test cases,  $T$ .

For each test case, there will be three input lines.

First line contains  $n$  (the size of array).

Second line contains  $n$  space-separated integers describing array.

Third line contains the key element that need to be searched in the array.

#### **Output format:**

The output will have  $T$  number of lines.

For each test case, output will be "Present" if the key element is found in the array, otherwise "Not Present".

Also, for each test case output the number of comparisons required to search the key.

```
#include <bits/stdc++.h>
using namespace std;
void jump(int arr[], int n, int key)
{
    int start = 0, comp = 0, flag = 0;
    int end = sqrt(n);
    while(arr[end] <= key && end < n)
    {
        comp++;
        start = end;
        end += sqrt(n);
        if(end > n-1) end = n;
    }
    for(int i=start; i<end; i++)
    {
        if(arr[i] == key)
        {
            flag = true;
            break;
        }
    }
    if(flag) cout<<"Present "<<comp<<endl;
    else cout<<"Not present"<<endl;
}
int main()
{
    int n;
    cin>>n;
    while(n-->0)
    {
        int size;
        cin>>size;
        int arr[size];
        for(int i = 0; i < size; i++) cin>>arr[i];
        int key;
        cin>>key;
```

```
jump(arr, size, key);  
}  
return 0;  
}
```

### **OUTPUT**

```
8  
21 39 40 45 51 54 68 72  
69  
Not Present 4
```

## Week 2

**Q1) 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. (Time Complexity =  $O(\log n)$ )**

**Input format:**

**The first line contains number of test cases, T.**

**For each test case, there will be three input lines.**

**First line contains n (the size of array).**

**Second line contains space-separated integers describing array.**

**Third line contains the key element that need to be searched in the array.**

**Output format:**

**The output will have T number of lines.**

**For each test case T, output will be the key element and its number of copies in the array if the key element is**

**present in the array otherwise print “Key not present”.**

```
#include<iostream>
using namespace std;
int main()
{
    int n,x;
    int count=0;
    cout<<"enter the size of array:"<<endl;
    cin>>n;
    int A[n];
    cout<<"\n Enter the element of array:"<<endl;
    for(int i=0;i<n;i++)
    {
        cin>>A[i];
    }
    cout<<"\n Enter the element you want to search"<<endl;
    cin>>x;
    for(int i=0;i<n;i++)
    {
        if(A[i]==x)
        {
            count++;
        }
    }
    cout<<"The frequency of key element is :"<<count;
    return 0;
}
```

## OUTPUT

```
enter the size of array:
10

Enter the element of array:
235 235 278 278 763 764 790 853 981 981

Enter the element you want to search
981
The frequency of key element is :2
```

**Q2) 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  $\text{arr}[i] + \text{arr}[j] = \text{arr}[k]$ .**

**Input format:**

**The first line contains number of test cases, T.**

**For each test case, there will be two input lines.**

**First line contains n (the size of array).**

**Second line contains space-separated integers describing array.**

**Output:**

**The output will have T number of lines.**

**For each test case T, print the value of i, j and k, if found else print "No sequence found".**

```
#include <bits/stdc++.h>
```

```
using namespace std;
```

```
int main() {
```

```
int t;
```

```
cin>>t;
```

```
while(t-->0) {
```

```
int n;
```

```
cin>>n;
```

```
int arr[n];
```

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

```
{
```

```
cin>>arr[i];
```

```
}
```

```
vector<int> v;
```

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

```
{
```

```
int k = n - 1;
```

```
int j = i + 1;
```

```
while(j < n && k > j)
```

```
{
```

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

```
{
```

```
v.push_back(i+1);
```

```
v.push_back(j+1);
```

```
v.push_back(k+1);
```

```
break;
```

```
}
```

```
else if(arr[i] + arr[j] > arr[k])
```

```
{
```

```
j++;
```

```
k = n - 1;
```

```
}
```

```
else k--; }
```

```
}
```

```
if(v.empty())
```

```
cout<<"No sequence found"<<endl;
```

```
else
```

```
{
```

```
for(auto it: v)
```

```
{
```

```
cout<<it<<" ";
```

```
}
```

```
cout<<endl;
```

```
}
```

```
}
```

}

## OUTPUT

```
3
5
1 5 84 209 341
No sequence found
10
24 28 48 71 86 89 92 120 194 201
2 7 8
```

**Q3) 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.**

**Input format:**

**The first line contains number of test cases, T.**

**For each test case, there will be three input lines.**

**First line contains n (the size of array).**

**Second line contains space-separated integers describing array. Third line contains the key element.**

**Output format:**

**The output will have T number of lines.**

**For each test case T, output will be the total count i.e., number of times such pair exists.**

```
#include <bits/stdc++.h>
```

```
using namespace std;
```

```
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;
```

```
sort(arr, arr + n);
```

```
int c = 0, l, h;
```

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

```
{
```

```
l = i;
```

```
h = n - 1;
```

```
while(l < h)
```

```
{
```

```
if(arr[h] - arr[l] == key)
```

```
{
```

```
c++;
```

```
h--;
```

```
l++;
```

```
}
```

```
else if(arr[h] - arr[l] > key) h--;
```

```
else l++;
```

```
}
```

```
}
```

```
cout<<c<<<endl;
```

```
}
```

```
}
```

**OUTPUT**

```
2
```

```
5
```

```
1 51 84 21 31
```

```
20
```

```
2
```



### Week 3

**Q1) 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.**

**Input Format:**

**The first line contains number of test cases, T.**

**For each test case, there will be two input lines.**

**First line contains n (the size of array).**

**Second line contains space-separated integers describing array.**

**Output Format:**

**The output will have T number of lines.**

**For each test case T, there will be three output lines.**

**First line will give the sorted array.**

**Second line will give total number of comparisons.**

**Third line will give total number of shift operations required.**

```
#include<bits/stdc++.h>
using namespace std;
void insertionSort(int a[], int n)
{
    int ctr=0,flag=0;
    for(int i=1;i<n;i++)
    {
        ctr++;
        int j;
        j=i-1;
        int x=a[i];
        while (j>-1 && a[j]>x)
        {
            ctr++;
            a[j+1]=a[j];
            flag++;
            j--;
        }
        a[j+1]=x;
    }
    for(int i=0;i<n;i++)
    {
        cout<<a[i]<<" ";
    }
    cout<<endl;
    cout<<" Shifts = "<<ctr<<endl;
    cout<<"Comparison = "<<flag<<endl;
}
int main()
{
    int n;
    cout<<"enter the no. of test cases"<<endl;
    cin>>n;
    while(n-->0)
    {
        int c;
        cout<<"enter the no. of elements in an array"<<endl;
        cin>>c;
        int a[c];
```

```
for(int i=0;i<c;i++)  
{  
cin>>a[i];  
}  
insertionSort(a,c);  
}  
}
```

## OUTPUT

```
enter the no. of test cases  
3  
enter the no. of elements in an array  
8  
-23 65 -31 76 46 89 45 32  
-31 -23 32 45 46 65 76 89  
Shifts = 20  
Comparison = 13
```

**Q2) 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.**

**Input Format:**

**The first line contains number of test cases, T.**

**For each test case, there will be two input lines.**

**First line contains n (the size of array).**

**Second line contains space-separated integers describing array.**

**Output Format:**

**The output will have T number of lines.**

**For each test case T, there will be three output lines.**

**First line will give the sorted array.**

**Second line will give total number of comparisons.**

**Third line will give total number of swaps required.**

```
#include<bits/stdc++.h>
using namespace std;
void selectionSort(int a[], int n)
{
    int ctr=0,flag=0,k;
    for (int i=0;i<n;i++)
    {
        ctr++;
        for(int j=i+1;j<n;j++)
        {
            flag++;
            if(a[i]>a[j])
            {
                int temp=a[i];
                a[i]=a[j];
                a[j]=temp;
            }
        }
    }
    for(int i=0;i<n;i++)
    {
        cout<<a[i]<<" ";
    }
    cout<<endl;
    cout<<"Swaps = "<<ctr<<endl;
    cout<<"Comparison = "<<flag<<endl;
}
int main()
{
    int n;
    cout<<"enter the no. of test cases"<<endl;
    cin>>n;
    while(n-->0)
    {
        int c;
        cout<<"enter the no. of elements in an array"<<endl;
        cin>>c;
        int a[c];
        for(int i=0;i<c;i++)
        {
            cin>>a[i];
        }
    }
}
```

```
}  
selectionSort(a,c);  
}  
}
```

### **OUTPUT**

```
enter the no. of test cases  
3  
enter the no. of elements in an array  
8  
-13 65 -21 76 46 89 45 12  
-21 -13 12 45 46 65 76 89  
Swaps = 8  
Comparison = 28
```

**Q3) 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. (use sorting)**  
(Time Complexity =  $O(n \log n)$ )

**Input Format:**

The first line contains number of test cases, T.

For each test case, there will be two input lines.

First line contains n (the size of array).

Second line contains space-separated integers describing array.

**Output Format:**

The output will have T number of lines.

For each test case, output will be 'YES' if duplicates are present otherwise 'NO'.

```
#include<bits/stdc++.h>
```

```
using namespace std;
```

```
void findDuplicate(int a[], int x)
```

```
{  
    for(int i=0;i<x;i++){  
        for(int j=i+1;j<x;j++){  
            if(a[i]==a[j]){  
                cout<<"YES"<<endl;  
            }  
        }  
    }  
    break;  
}
```

```
cout<<"NO"<<endl;  
}
```

```
int main(){  
    int n;  
    cin>>n;  
    while(n--){  
        int x;  
        cin>>x;  
        int a[x];  
        for(int i=0;i<x;i++){  
            cin>>a[i];  
        }  
        findDuplicate(a ,x);  
    }  
}
```

## **OUTPUT**

```
3  
5  
28 52 83 14 75  
NO
```

## Week 4

**Q1) 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.**

**Input Format:** The first line contains number of test cases, T. For each test case, there will be two input lines.

**First line contains n (the size of array). Second line contains space-separated integers describing array.**

**Output Format:** The output will have T number of lines. For each test case T, there will be three output lines.

**First line will give the sorted array. Second line will give total number of comparisons. Third line will give**

**total number of inversions required.**

```
#include <bits/stdc++.h>
using namespace std;
int c = 0;
void mergeArray(int arr[], int l, int mid, int h)
{
    int n1 = mid - l + 1;
    int n2 = h - mid;
    int a[n1], b[n2];
    for(int i = 0; i < n1; i++)
    {
        a[i] = arr[l + i];
    }
    for(int i = 0; i < n2; i++)
    {
        b[i] = arr[mid + 1 + i];
    }
    int i = 0, j = 0, k = l;
    while(i < n1 && j < n2)
    {
        c++;
        if(a[i] <= b[j])
        {
            arr[k] = a[i];
            i++;
        }
        else
        {
            arr[k] = b[j];
            j++;
        }
        k++;
    }
    while(i < n1)
    {
        arr[k] = a[i];
        i++; k++;
    }
    while(j < n2)
    {
        arr[k] = b[j];
```

```

j++; k++;
}
}
void mergeSort(int arr[], int l, int h)
{
if(l < h)
{
int mid = l + (h - l) / 2;
mergeSort(arr, l, mid);
mergeSort(arr, mid + 1, h);
mergeArray(arr, l, mid, h);
}
}
void display(int arr[], int n)
{
for(int i = 0; i < n; i++)
{
cout<<arr[i]<<" ";
}
cout<<endl;
}
int main()
{
int t;
cin>>t;
while(t--)
{
int n;
cin>>n;
int *arr = new int[n];
for(int i = 0; i < n; i++)
{
cin>>arr[i];
}
mergeSort(arr, 0, n - 1);
display(arr, n);
cout<<"comparisons = "<<c<<endl;
delete []arr;
}
}

```

## **OUTPUT**

```

3
8
23 65 21 76 46 89 45 32
21 23 32 45 46 65 76 89
comparisons = 16

```

**Q2) 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. Pivot element should be selected randomly from the array. Your program should also find number of comparisons and swaps required for sorting the array. Input Format: The first line contains number of test cases, T. For each test case, there will be two input lines.**

**First line contains n (the size of array). Second line contains space-separated integers describing array.**

**Output Format: The output will have T number of lines. For each test case T, there will be three output lines.**

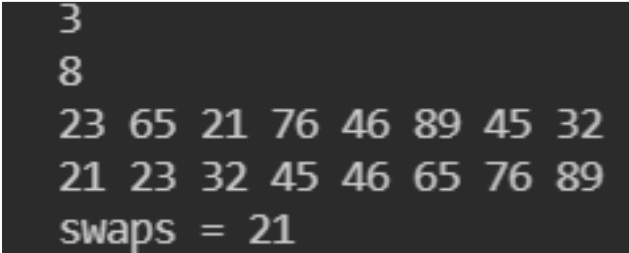
**First line will give the sorted array. Second line will give total number of comparisons. Third line will give total number of swaps required.**

```
#include <bits/stdc++.h>
using namespace std;
int c = 0, s = 0;
int partition(int arr[], int l, int h)
{
    int x = (rand() % (l - h)) + l;
    if(h != x) {
        s++;
        swap(arr[x], arr[h]);
    }
    int pivot = arr[h];
    int i = l - 1;
    for(int j = l; j <= h - 1; j++)
    {
        if(arr[j] <= pivot)
        {
            i++;
            s++;
            swap(arr[i], arr[j]);
        }
    }
    s++;
    swap(arr[i + 1], arr[h]);
    return i + 1;
}
void quickSort(int arr[], int l, int h)
{
    if(l < h)
    {
        int pivot = partition(arr, l, h);
        quickSort(arr, l, pivot - 1);
        quickSort(arr, pivot + 1, h);
    }
}
void display(int arr[], int n)
{
    for(int i = 0; i < n; i++)
    {
        cout<<arr[i]<<" ";
    }
    cout<<endl;
```



```
}  
int main()  
{  
int t;  
cin>>t;  
while(t--)  
{  
int n;  
cin>>n;  
int *arr = new int[n];  
for(int i = 0; i < n; i++)  
{  
cin>>arr[i];  
}  
quickSort(arr, 0, n - 1);  
display(arr, n);  
cout<<"swaps = "<<s<<endl;  
}  
}
```

### **OUTPUT**

A screenshot of a terminal window with a dark background and light gray text. The output shows the number 3 on the first line, 8 on the second line, two rows of eight integers each on the third and fourth lines, and the text 'swaps = 21' on the fifth line.

```
3  
8  
23 65 21 76 46 89 45 32  
21 23 32 45 46 65 76 89  
swaps = 21
```

**Q3) 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. (Worst case Time Complexity =  $O(n)$ )**

**Input Format: The first line contains number of test cases, T. For each test case, there will be three input**

**lines. First line contains n (the size of array). Second line contains space-separated integers describing array.**

**Third line contains K.**

**Output Format: The output will have T number of lines. For each test case, output will be the Kth smallest or**

**largest array element. If no Kth element is present, output should be “not present”.**

```
#include<bits/stdc++.h>
using namespace std;
int main()
{
    int t;
    cin>>t;
    while(t--)
    {
        int n;
        cin>>n;
        int *arr = new int[n];
        for(int i = 0; i < n; i++)
        {
            cin>>arr[i];
        }
        int k;
        cin>>k;
        priority_queue<int> pq;
        for(int i = 0; i < k; i++)
        {
            pq.push(arr[i]);
        }
        for(int i = k; i < n; i++)
        {
            pq.push(arr[i]);
            if(pq.size() > k) pq.pop();
        }
        if(pq.empty()) cout<<"Not present"<<endl;
        else cout<<pq.top()<<endl;
        delete []arr;
    }
}
```

## **OUTPUT**

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

## Week 5

**Q1) Given an unsorted array of alphabets containing duplicate elements. Design an algorithm and implement it using a program to find which alphabet has maximum number of occurrences and print it. (Time Complexity =  $O(n)$ ) (Hint: Use counting sort)**

**Input Format:** The first line contains number of test cases, T. For each test case, there will be two input lines.

**First line contains n (the size of array). Second line contains space-separated integers describing array.**

**Output:** The output will have T number of lines. For each test case, output will be the array element which

**has maximum occurrences and its total number of occurrences. If no duplicates are present (i.e. all the**

**elements occur only once), output should be “No Duplicates Present”.**

```
#include<iostream>
using namespace std;
char findmax(char a[], int n)
{
    char m=a[0];
    for(int i=1;i<n;i++)
    {
        if(a[i]>m)
        {
            m=a[i];
        }
    }
    return m;
}
void count_sort(char a[],int n)
{
    int i,*c,j,dup,temp;
    char max=findmax(a,n);
    int m=int(max);
    c = new int[m+1];
    for(i=0;i<m+1;i++)
    {
        c[i]=0;
    }
    for(int i=0;i<n;i++)
    {
        c[int(a[i])]++;
    }
    dup=c[i];
    for(int i=0;i<m+1;i++)
    {
        if(c[i]>dup)
        {
            dup=c[i];
            temp=i;
        }
    }
    char ch=(char)temp;
    if(dup>1)
    {
        cout<<ch<<" : "<<dup<<endl;
    }
}
```

```

}
else
{
cout<<"No duplicates found"<<endl;
}
}
int main()
{
int t,n;
cout<<"enter number of test case"<<endl;
cin>>t;
while(t>0)
{
cout<<"enter number of elements in array"<<endl;
cin>>n;
char a[n];
cout<<"enter "<<n<<" number of elements"<<endl;
for(int i=0;i<n;i++)
{
cin>>a[i];
}
count_sort(a,n);
t--;
}
}

```

## **OUTPUT**

```

3
10
a e d w a d q a f p
a - 3

```

**Q2) Given an unsorted array of integers, design an algorithm and implement it using a program to find**

**whether two elements exist such that their sum is equal to the given key element.**

**(Time Complexity =  $O(n \log n)$ )**

**Input Format: The first line contains number of test cases, T. For each test case, there will be two input lines.**

**First line contains n (the size of array). Second line contains space-separated integers describing array. Third**

**line contains key**

**Output Format: The output will have T number of lines. For each test case, output will be the elements arr[i]**

**and arr[j] such that arr[i]+arr[j] = key if exist otherwise print 'No Such Elements Exist'.**

```
#include <bits/stdc++.h>
```

```
using namespace std;
```

```
int main()
```

```
{
```

```
int t;
```

```
cin >>t;
```

```
while(t--)
```

```
{
```

```
int n,sum,flag=0;
```

```
cin >> n;
```

```
int ar[n];
```

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

```
cin >> ar[i];
```

```
cin >> sum;
```

```
sort(ar,ar+n);
```

```
int a=0,b=n-1;
```

```
while(a<b)
```

```
{
```

```
if(ar[a]+ar[b]>sum)
```

```
--b;
```

```
else if(ar[a]+ar[b]<sum)
```

```
++a;
```

```
else
```

```
{
```

```
cout << ar[a] << "&" << ar[b] << ", ";++a;--b;++flag;
```

```
}
```

```
}
```

```
if(!flag)
```

```
cout << "NO Such Pair Exist" << "\n";
```

```
}
```

```
}
```

## **OUTPUT**

```
2
10
64 28 97 40 12 72 84 24 38 10
50
10&40, 12&38,
```

**Q3) You have been given two sorted integer arrays of size m and n. Design an algorithm and implement it using a program to find list of elements which are common to both. (Time Complexity =  $O(m+n)$ )**

**Input Format:** First line contains m (the size of first array). Second line contains m space-separated integers

describing first array. Third line contains n (the size of second array). Fourth line contains n space-separated integers describing second array.

**Output Format:** Output will be the list of elements which are common to both.

```
#include <bits/stdc++.h>
using namespace std;
int main()
{
    int n,m;
    cin >> n ;
    int ar[n];
    for(int i=0;i<n;++i)
    cin >> ar[i];
    cin >> m;
    int ar2[m];
    for (int i = 0; i < m; ++i)
    cin >> ar2[i];
    int a=0,b=0;
    while(a<n && b<m)
    {
        if(ar[a] < ar2[b])
        ++a;
        else if(ar[a] > ar2[b])
        ++b;
        else
        {
            cout << ar[a] << " ";++a;++b;}
        }
    cout << "\n";
}
```

## **OUTPUT**

```
7
10 10 34 39 55 76 85
12
10 10 11 30 30 34 34 51 55 69 72 89
10 10 34 55
```

## Week 6

**Q1) Given a (directed/undirected) graph, design an algorithm and implement it using a program to find if a path exists between two given vertices or not. (Hint: use DFS)**

**Input Format:** Input will be the graph in the form of adjacency matrix or adjacency list. Source vertex number and destination vertex number is also provided as an input.

**Output Format:** Output will be 'Yes Path Exists' if path exists, otherwise print 'No Such Path Exists'.

```
#include<iostream>
#include<fstream>
#include<vector>
#include<stack>
using namespace std;
bool isPathDFS(vector<vector<int>>& graphmat, int source, int des, int size)
{
    bool visited[size];
    stack<int> st;
    visited[source-1] = true;
    st.push(source-1);
    while(!st.empty()){
        int cr = st.top();
        st.pop();
        if(cr+1 == des)
            return true;
        for(int i=0;i<size;i++){
            if(graphmat[source][i] == 1 && !visited[i]){
                visited[i] = true;
                st.push(i);
            }
        }
    }
    return false;
}
int main()
{
    ifstream file;
    file.open("input1.txt");
    if(!file)
    {
        cout<<"File not found";
        return 0;
    }
    int v;
    file>>v;
    vector<vector<int>> graphmat(v,vector<int>(v));
    for(int i=0;i<v;i++)
        for(int j=0;j<v;j++)
            file>>graphmat[i][j];
    int source,des;
    file>>source;
    file>>des;
    if(isPathDFS(graphmat,source,des,v))
        cout<<"Yes path exists"<<endl;
    else
        cout<<"No such path exists"<<endl;
}
```

## INPUT

```
5
0 1 1 0 0
1 0 1 1 1
1 1 0 1 0
0 1 1 0 1
0 1 0 1 0
1 5
```

## OUTPUT

```
Yes path exixts
```

```
...Program finished with exit code 0
Press ENTER to exit console.
```



**Q2) Given a graph, design an algorithm and implement it using a program to find if a graph is bipartite or not. (Hint: use BFS)**

**Input Format:** Input will be the graph in the form of adjacency matrix or adjacency list.

**Output Format:** Output will be 'Yes Bipartite' if graph is bipartite, otherwise print 'Not Bipartite'.

```
#include<iostream>
#include<vector>
#include<queue>
#include<fstream>
using namespace std;
bool isbipartite(vector<vector<int>>& graph, int s)
{
    int V = graph.size();
    vector<int> visited(V);
    for(int i=0; i<V;i++)
        visited[i] = -1;
    visited[s] = 1;
    queue<int> qu;
    qu.push(s);
    while(!qu.empty())
    {
        int u = qu.front();
        qu.pop();
        if(graph[u][u] == 1)
            return false;
        for(int v=0; v<V;v++)
        {
            if(graph[u][v] && visited[v] == -1)
            {
                visited[v] = 1 - visited[u];
                qu.push(v);
            }
            else if(graph[u][v] && visited[v] == visited[u])
                return false;
        }
    }
    return true;
}
int main()
{
    int n;
    ifstream file;
    file.open("input2.txt");
    if(!file)
    {
        cout<<"File not found";
        return 0;
    }
    file>>n;
    vector<vector<int>> graph(n,vector<int> (n,0));
    for(int i=0;i<n;i++)
        for(int j=0;j<n;j++)
            file>>graph[i][j];
    if(isbipartite(graph,0))
        cout<<"Bipartite"<<endl;
    else
```

```
        cout<<"Not Bipartite"<<endl;  
    return 0;  
}
```

### **INPUT**

```
5  
0 1 1 0 0  
1 0 1 1 1  
1 1 0 1 0  
0 1 1 0 1  
0 1 0 1 0
```

### **OUTPUT**

```
Not Bipartite  
  
...Program finished with exit code 0  
Press ENTER to exit console.
```

## Week 7

**Q1) After end term examination, Akshay wants to party with his friends. All his friends are living as paying guest and it has been decided to first gather at Akshay's house and then move towards party location. The problem is that no one knows the exact address of his house in the city. Akshay as a computer science wizard knows how to apply his theory subjects in his real life and came up with an amazing idea to help his friends. He draws a graph by looking in to location of his house and his friends' location (as a node in the graph) on a map. He wishes to find out shortest distance and path covering that distance from each of his friend's location to his house and then whatsapp them this path so that they can reach his house in minimum time. Akshay has developed the program that implements Dijkstra's algorithm but not sure about correctness of results. Can you also implement the same algorithm and verify the correctness of Akshay's results? (Hint: Print shortest path and distance from friends' location to Akshay's house)**

```
#include<iostream>
#include<limits.h>
using namespace std;
int minDistance(int V,int dist[],bool shortpath[]){
    int min= INT_MAX,m_index;
    for(int v=0;v<V;v++){
        if(!shortpath[v]&&dist[v]<=min){
            min=dist[v];
            m_index=v;
        }
    }
    return m_index;
}
void dijkstra(int **g,int V,int s){
    int dist[V];
    int parent[V];
    bool shortpath[V];
    for(int i=0;i<V;i++){
        dist[i]=INT_MAX;
        shortpath[i]=false;
        parent[i]=0;
    }
    dist[s]=0;
    for(int i=0;i<V;i++){
        int u=minDistance(V,dist,shortpath);
        shortpath[u]=true;
        for(int v=0;v<V;v++){
            if(!shortpath[v] && g[u][v] && dist[v]>dist[u]+g[u][v]){
                dist[v]=dist[u]+g[u][v];
                parent[v]=u;
            }
        }
    }
    for(int i=0;i<V;i++){
        cout<<"Node "<<i+1<<" | Path: ";
        int j=i;
        while(j!=s){
            cout<<parent[j]+1<<"<-";
            j=parent[j];
        }
        cout<<" | Distance: "<<dist[i]<<endl;
    }
}
```

```

}
int main(){
    int v,s;
    cin>>v;
    int **g = new int*[v];
    for(int i=0;i<v;i++)
        g[i]= new int[v];

    for(int i=0;i<v;i++)
        for(int j=0;j<v;j++)
            cin>>g[i][j];
    cin>>s;
    dijkstra(g,v,s-1);
    return 0;
}

```

## OUTPUT

```

5
0 4 1 0 0
0 0 0 0 4
0 2 0 4 0
0 0 0 0 4
0 0 0 0 0
1
Node 1 | Path: | Distance: 0
Node 2 | Path: 3<-1<- | Distance: 3
Node 3 | Path: 1<- | Distance: 1
Node 4 | Path: 3<-1<- | Distance: 5
Node 5 | Path: 2<-3<-1<- | Distance: 7

...Program finished with exit code 0
Press ENTER to exit console.

```

**Q2) Design an algorithm and implement it using a program to solve previous question's problem using Bellman- Ford's shortest path algorithm.**

**Input Format:** Input will be the graph in the form of adjacency matrix or adjacency list.

Source vertex number is also provided as an input.

**Output Format:** Output will contain V lines. Each line will represent the whole path from destination vertex number to source vertex number along with minimum path weight.

```
#include<iostream>
#include<limits.h>
using namespace std;
void bellman_ford(int **g,int V,int s){
    int dist[V];
    int parent[V];
    bool shortpath[V];
    for(int i=0;i<V;i++){
        dist[i]=INT_MAX;
        parent[i]=0;
    }
    dist[s]=0;
    for(int i=0;i<V-1;i++){
        for(int u=s;u<V;u++){
            for(int v=0;v<V;v++){
                if(dist[u]!=INT_MAX && g[u][v] && dist[v]>dist[u]+g[u][v]){
                    dist[v]=dist[u]+g[u][v];
                    parent[v]=u;
                }
            }
        }
    }
    for(int u=0;u<V;u++){
        for(int v=0;v<V;v++){
            if(g[u][v] && dist[v]>dist[u]+g[u][v]){
                cout<<"-ve cycle exist.";
                return;
            }
        }
    }
    for(int i=0;i<V;i++){
        cout<<"Node "<<i+1<<" | Path: ";
        int j=i;
        while(j!=s){
            cout<<parent[j]+1<<"<-";
            j=parent[j];
        }
        cout<<" | Distance: "<<dist[i]<<endl;
    }
}

int main(){
    int v,s;
    cin>>v;
    int **g = new int*[v];
    for(int i=0;i<v;i++)
        g[i]= new int[v];
    for(int i=0;i<v;i++)
        for(int j=0;j<v;j++)
            cin>>g[i][j];
```

```
    cin>>s;
    bellman_ford(g,v,s-1);
    return 0;
}
```

## OUTPUT

```
5
0 4 1 0 0
0 0 0 0 4
0 2 0 4 0
0 0 0 0 4
0 0 0 0 0
1
Node 1 | Path: | Distance: 0
Node 2 | Path: 3<-1<- | Distance: 3
Node 3 | Path: 1<- | Distance: 1
Node 4 | Path: 3<-1<- | Distance: 5
Node 5 | Path: 2<-3<-1<- | Distance: 7

...Program finished with exit code 0
Press ENTER to exit console.
```

**Q3) Given a directed graph with two vertices ( source and destination). Design an algorithm and implement it using a program to find the weight of the shortest path from source to destination with exactly k edges on the path.**

**Input Format:** First input line will obtain number of vertices V present in the graph. Graph in the form of adjacency matrix or adjacency list is taken as an input in next V lines. Next input line will obtain source and destination vertex number. Last input line will obtain value k.

**Output Format:**

Output will be the weight of shortest path from source to destination having exactly k edges. If no path is available then print "no path of length k is available".

```
#include<iostream>
#include<limits.h>
#include<algorithm>
using namespace std;
int shortest_path_k_edge(int **g,int v,int s,int d,int k){
    int s_path_weight[v][v][k+1];
    for(int e=0;e<=k;e++){
        for(int i=0;i<v;i++){
            for(int j=0;j<v;j++){
                s_path_weight[i][j][e]=INT_MAX;
                if(e==0&&i==j)
                    s_path_weight[i][j][e]=0;
                if(e==1&&g[i][j])
                    s_path_weight[i][j][e]=g[i][j];
                if(e>1){
                    for(int a=0;a<v;a++){
                        if(g[i][a] && i!=a && j!=a && s_path_weight[a][j][e-1]!=INT_MAX)
                            s_path_weight[i][j][e] = min(s_path_weight[i][j][e],g[i][a]+s_path_weight[a][j][e-1]);
                    }
                }
            }
        }
    }
    return s_path_weight[s][d][k];
}
int main(){
    int v,s,d,k;
    cin>>v;
    int **g = new int*[v];
    for(int i=0;i<v;i++){
        g[i]= new int[v];

        for(int i=0;i<v;i++)
            for(int j=0;j<v;j++)
                cin>>g[i][j];
    }
    cin>>s>>d>>k;
    int res = shortest_path_k_edge(g,v,s-1,d-1,k);
    if(res==INT_MAX)
        cout<<"NO PATH EXIST.";
    else
        cout<<"Weight of shortest path: "<<res;
    return 0;
}
```

## OUTPUT

```
4
0 10 3 2
0 0 0 7
0 0 0 6
0 0 0 0
1 4
2
Weight of shortest path: 9

...Program finished with exit code 0
Press ENTER to exit console.
```



## Week 8

**Q1) Assume that a project of road construction to connect some cities is given to your friend. Map of these cities and roads which will connect them (after construction) is provided to him in the form of a graph. Certain amount of rupees is associated with construction of each road. Your friend has to calculate the minimum budget required for this project. The budget should be designed in such a way that the cost of connecting the cities should be minimum and number of roads required to connect all the cities should be minimum (if there are N cities then only N-1 roads need to be constructed). He asks you for help. Now, you have to help your friend by designing an algorithm which will find minimum cost required to connect these cities. (use Prim's algorithm)**

```
#include<bits/stdc++.h>
using namespace std;
int prim(vector<vector<int>>& v, int n) {
    vector<bool>vis(n, false);
    vector<int>wei(n, INT_MAX);
    priority_queue<pair<int, int>, vector<pair<int, int>>, greater<pair<int, int>>>mh;
    int s = 0;
    wei[s] = 0;
    mh.push(make_pair(wei[s], s));
    while (!mh.empty()) {
        int i = mh.top().second;
        mh.pop();
        if (!vis[i]) {
            vis[i] = true;
            for (int j = 0; j < n; j++) {
                if (!vis[j] && v[i][j] != 0 && v[i][j] < wei[j]) {
                    wei[j] = v[i][j];
                    mh.push(make_pair(wei[j], j));
                }
            }
        }
    }
    int sum = 0;
    for (auto i : wei)
        sum = sum + i;
    return sum;
}
int main() {
#ifdef ONLINE_JUDGE
    freopen("input_1.txt", "r", stdin);
    freopen("output_1.txt", "w", stdout);
#endif
    int n, t;
    cin >> n;
    vector<vector<int>>v;
    vector<int>vec;
    for (int i = 0; i < n; i++) {
        vec.clear();
        for (int j = 0; j < n; j++) {
            cin >> t;
            vec.push_back(t);
        }
        v.push_back(vec);
    }
    cout << "Minimum spanning weight : " << prim(v, n);
```

}

## OUTPUT

```
main.cpp  input_1.txt  output_1.txt
1 7
2 0 0 7 5 0 0 0
3 0 0 8 5 0 0 0
4 7 8 0 9 7 0 0
5 5 0 9 0 15 6 0
6 0 5 7 15 0 8 9
7 0 0 0 6 8 0 11
8 0 0 0 0 9 11 0
```

```
main.cpp  input_1.txt  output_1.txt
1 Minimum spanning weight : 39
```

**Q2) Implement the previous problem using Kruskal's algorithm.**

**Input Format:** The first line of input takes number of vertices in the graph. Input will be the graph in the form of adjacency matrix or adjacency list.

**Output Format:** Output will be minimum spanning weight.

```
#include<bits/stdc++.h>
using namespace std;
int fp(vector<int>p, int i) {
    if (p[i] < 0)
        return i;
    return fp(p, p[i]);
}
bool unionbweig(vector<int>& p, int u, int v) {
    int pu = fp(p, u);
    int pv = fp(p, v);
    if (pu != pv) {
        if (p[pu] < p[pv]) {
            p[pu] += p[pv];
            p[pv] = pu;
        }
        else {
            p[pv] += p[pu];
            p[pu] = pv;
        }
        return true;
    }
    return false;
}
int krus(vector<vector<int>>& v, int n) {
    int ans = 0;
    vector<pair<int, pair<int, int>>> g;
    for (int i = 0; i < n; ++i)
        for (int j = 0; j < n; ++j)
            if (v[i][j] != 0)
                g.push_back(make_pair(v[i][j], make_pair(i, j)));
    sort(g.begin(), g.end());
    vector<int>p(n, -1);
    for (auto i : g) {
        int u = i.second.first;
        int v = i.second.second;
        int w = i.first;
        if (unionbweig(p, u, v))
            ans = ans + w;
    }
    return ans;
}
int main() {
#ifdef ONLINE_JUDGE
    freopen("input_2.txt", "r", stdin);
    freopen("output_2.txt", "w", stdout);
#endif
    int n, t;
    cin >> n;
    vector<vector<int>>v;
    vector<int>vec;
    for (int i = 0; i < n; i++) {
```

```

    vec.clear();
    for (int j = 0;j < n;j++) {
        cin >> t;
        vec.push_back(t);
    }
    v.push_back(vec);
}
cout << "Minimum spanning weight : " << krus(v, n);
}

```

## OUTPUT

main.cpp	input_2.txt	output_2.txt
1	7	
2	0 0 7 5 0 0 0	
3	0 0 8 5 0 0 0	
4	7 8 0 9 7 0 0	
5	5 0 9 0 15 6 0	
6	0 5 7 15 0 8 9	
7	0 0 0 6 8 0 11	
8	0 0 0 0 9 11 0	

main.cpp	input_2.txt	output_2.txt
1	Minimum spanning weight : 37	

**Q3) Assume that same road construction project is given to another person. The amount he will earn from this project is directly proportional to the budget of the project. This person is greedy, so he decided to maximize the budget by constructing those roads who have highest construction cost. Design an algorithm and implement it using a program to find the maximum budget required for the project.**

**Input Format:** The first line of input takes number of vertices in the graph. Input will be the graph in the form of adjacency matrix or adjacency list.

**Output Format:** Output will be maximum spanning weight.

```
#include<bits/stdc++.h>
using namespace std;
int fp(vector<int>p, int i)
{
    if (p[i] < 0)
        return i;
    return fp(p, p[i]);
}
bool unionbweig(vector<int>& p, int u, int v)
{
    int pu = fp(p, u);
    int pv = fp(p, v);
    if (pu != pv)
    {
        if (p[pu] <= p[pv])
        {
            p[pu] += p[pv];
            p[pv] = pu;
        }
        else
        {
            p[pv] += p[pu];
            p[pu] = pv;
        }
    }
    return true;
}
return false;
}
int krus(vector<vector<int>>& v, int n)
{
    int ans = 0;
    vector<pair<int, pair<int, int>>> g;
    for (int i = 0; i < n; ++i)
        for (int j = 0; j < n; ++j)
            if (v[i][j] != 0)
                g.push_back(make_pair(v[i][j], make_pair(i, j)));
    sort(g.begin(), g.end(), greater<pair<int, pair<int, int>>>());
    vector<int>p(n, -1);
    for (auto i : g)
    {
        int u = i.second.first;
        int v = i.second.second;
        int w = i.first;
        if (unionbweig(p, u, v))
            ans = ans + w;
    }
    return ans;
}
```

```

}
int main() {
#ifdef ONLINE_JUDGE
    freopen("input_3.txt", "r", stdin);
    freopen("output_3.txt", "w", stdout);
#endif
    int n, t;
    cin >> n;
    vector<vector<int>>>v;
    vector<int>vec;
    for (int i = 0;i < n;i++)
    {
        vec.clear();
        for (int j = 0;j < n;j++)
        {
            cin >> t;
            vec.push_back(t);
        }
        v.push_back(vec);
    }
    cout << "Maximum spanning weight : " << krus(v, n);
}

```

## OUTPUT

main.cpp	input_3.txt	output_3.txt
1	7	
2	0 0 7 5 0 0 0	
3	0 0 8 5 0 0 0	
4	7 8 0 9 7 0 0	
5	5 0 9 0 15 6 0	
6	0 5 7 15 0 8 9	
7	0 0 0 6 8 0 11	
8	0 0 0 0 9 11 0	

main.cpp	input_3.txt	output_3.txt
1	Maximum spanning weight : 59	

## Week 9

**Q1) Given a graph, Design an algorithm and implement it using a program to implement Floyd-Warshall all pair shortest path algorithm.**

**Input Format:** The first line of input takes number of vertices in the graph. Input will be the graph in the form of adjacency matrix or adjacency list. If a direct edge is not present between any pair of vertex (u,v), then this entry is shown as AdjM[u,v] = INF.

**Output Format:** Output will be shortest distance matrix in the form of V X V matrix, where each entry (u,v) represents shortest distance between vertex u and vertex v.

```
#include<bits/stdc++.h>
using namespace std;
#define INF 1e9
int main() {
#ifdef ONLINE_JUDGE
    freopen("input_1.txt", "r", stdin);
    freopen("output_1.txt", "w", stdout);
#endif
    int n;
    cin >> n;
    int a;
    int arr[n][n], dist[n][n];
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            cin >> a;
            if (a < 0) {
                arr[i][j] = INF;
            }
            else
                arr[i][j] = a;
            dist[i][j] = arr[i][j];
        }
    }
    for (int k = 0; k < n; k++) {
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                if (dist[i][k] + dist[k][j] < dist[i][j]) {
                    dist[i][j] = dist[i][k] + dist[k][j];
                }
            }
        }
    }
    cout << "Shortest Distance Matrix: " << endl;
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            if (dist[i][j] == INF) {
                cout << "INF ";
            }
            else
                cout << dist[i][j] << " ";
        }
        cout << endl;
    }
    return 0;
}
```

## OUTPUT

```
main.cpp  input_1.txt  output_1.txt
1 5
2 0 10 5 5 INF
3 INF 0 5 5 5
4 INF INF 0 INF 10
5 INF INF INF 0 20
6 INF INF INF 5 0
```

```
main.cpp  input_1.txt  output_1.txt
1 Shortest Distance Matrix:
2 0 10 15 5 15
3 INF 0 5 5 5
4 INF INF 0 15 10
5 INF INF INF 0 20
6 INF INF INF 5 0
7
```



**Q2) Given a knapsack of maximum capacity  $w$ .  $N$  items are provided, each having its own value and weight. You have to Design an algorithm and implement it using a program to find the list of the selected items such that the final selected content has weight  $w$  and has maximum value. You can take fractions of items,i.e. the items can be broken into smaller pieces so that you have to carry only a fraction  $X_i$  of item  $i$ , where  $0 \leq x_i \leq 1$ .**

**Input Format:** First input line will take number of items  $N$  which are provided. Second input line will contain  $N$  space-separated array containing weights of all  $N$  items. Third input line will contain  $N$  space-separated array containing values of all  $N$  items. Last line of the input will take the maximum capacity  $w$  of knapsack.

**Output Format:** First output line will give maximum value that can be achieved. Next Line of output will give list of items selected along with their fraction of amount which has been taken.

```
#include<bits/stdc++.h>
using namespace std;
int main() {
#ifdef ONLINE_JUDGE
    freopen("input_2.txt", "r", stdin);
    freopen("output_2.txt", "w", stdout);
#endif
    int n;
    cin >> n;
    vector<double> items(n);
    vector<double> val(n);
    vector<vector<double>>> job;
    for (int i = 0; i < n; i++) {
        cin >> items[i];
    }
    for (int i = 0; i < n; i++) {
        cin >> val[i];
        job.push_back({ val[i] / items[i], items[i], (double)(i + 1) });
    }
    double k;
    cin >> k;
    sort(job.rbegin(), job.rend());
    vector<pair<double, double>> ls;
    float profit = 0;
    for (int i = 0; i < n; i++) {
        if (job[i][1] >= k) {
            profit += k * job[i][0];
            ls.push_back(make_pair(k, job[i][2]));
            break;
        }
        else {
            profit += job[i][1] * job[i][0];
        }
        ls.push_back(make_pair(job[i][1], job[i][2]));
        k = k - job[i][1];
    }
    cout << "Maximum Value is: " << profit << endl;
    cout << "Item - Weight" << endl;
    for (auto it : ls)
        cout << it.second << " - " << it.first << endl;
    return 0;
}
```

## OUTPUT

main.cpp	input_2.txt	output_2.txt
1	6	
2	6 10 3 5 1 3	
3	6 2 1 8 3 5	
4	16	

main.cpp	input_2.txt	output_2.txt
1	Maximum Value is: 22.3333	
2	Item - Weight	
3	5 - 1	
4	6 - 3	
5	4 - 5	
6	1 - 6	
7	3 - 1	

**Q3) Given an array of elements. Assume  $arr[i]$  represents the size of file  $i$ . Write an algorithm and a program to merge all these files into single file with minimum computation. For given two files A and B with sizes  $m$  and  $n$ , computation cost of merging them is  $O(m+n)$ . (Hint: use greedy approach)**

**Input Format:** First line will take the size  $n$  of the array. Second line will take array  $s$  as input.

**Output Format:** Output will be the minimum computation cost required to merge all the elements of the array.

```
#include<bits/stdc++.h>
using namespace std;
int main() {
#ifdef ONLINE_JUDGE
    freopen("input_3.txt", "r", stdin);
    freopen("output_3.txt", "w", stdout);
#endif
    int n;
    cin >> n;
    vector<int> a(n);
    for (int i = 0; i < n; i++) {
        cin >> a[i];
    }
    priority_queue<int, vector<int>, greater<int>> minheap;
    for (int i = 0; i < n; i++) {
        minheap.push(a[i]);
    }
    int ans = 0;
    while (minheap.size() > 1) {
        int e1 = minheap.top();
        minheap.pop();
        int e2 = minheap.top();
        minheap.pop();
        //cout << e1 << ' ' << e2 << "\n";
        ans += e1 + e2;
        minheap.push(e1 + e2);
    }
    cout << ans ;
    return 0;
}
```

## **OUTPUT**

main.cpp	input_3.txt	output_3.txt
1	10	
2	10 5 100 50 20 15 5 20 100 10	

main.cpp	input_3.txt	output_3.txt
1	960	

## Week 10

**Q1) Given a list of activities with their starting time and finishing time. Your goal is to select maximum number of activities that can be performed by a single person such that selected activities must be non-conflicting. Any activity is said to be non-conflicting if starting time of an activity is greater than or equal to the finishing time of the other activity. Assume that a person can only work on a single activity at a time.**

**Input Format:** First line of input will take number of activities N.

Second line will take N space-separated values defining starting time for all the N activities.

Third line of input will take N space-separated values defining finishing time for all the N activities.

**Output Format:** Output will be the number of non-conflicting activities and the list of selected activities.

```
#include<bits/stdc++.h>
```

```
using namespace std;
```

```
int main() {
```

```
#ifndef ONLINE_JUDGE
```

```
    freopen("input_1.txt", "r", stdin);
```

```
    freopen("output_1.txt", "w", stdout);
```

```
#endif
```

```
    int n;
```

```
    cin >> n;
```

```
    vector<vector<int>> v(n, vector<int>(3));
```

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

```
        cin >> v[i][0];
```

```
        v[i][2] = i;
```

```
    }
```

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

```
        cin >> v[i][1];
```

```
    }
```

```
    sort(v.begin(), v.end(), [&](vector<int>& a, vector<int>& b) {
```

```
        return a[1] < b[1];
```

```
    });
```

```
    int take = 1;
```

```
    int end = v[0][1];
```

```
    vector<int> ans;
```

```
    ans.push_back(v[0][2] + 1);
```

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

```
        if (v[i][0] >= end) {
```

```
            take++;
```

```
            end = v[i][1];
```

```
            ans.push_back(v[i][2] + 1);
```

```
        }
```

```
    }
```

```
    cout << "No. of non-conflicting activities: " << take << endl << "List of selected activities: ";
```

```
    for (int i : ans)
```

```
        cout << i << ' ';
```

```
}
```

## OUTPUT

main.cpp	input_1.txt	output_1.txt
1	10	
2	1 3 0 5 3 5 8 8 2 12	
3	4 5 6 7 9 9 11 12 14 16	

main.cpp	input_1.txt	output_1.txt
1	No. of non-conflicting activities: 4	
2	List of selected activities: 1 4 7 10	

**Q2) Given a long list of tasks. Each task takes specific time to accomplish it and each task has a deadline associated with it. You have to design an algorithm and implement it using a program to find maximum number of tasks that can be completed without crossing their deadlines and also find list of selected tasks.**

**Input Format:** First line will give total number of tasks n. Second line of input will give n space-separated elements of array representing time taken by each task. Third line of input will give n space-separated elements of array representing deadline associated with each task.

**Output Format:** Output will be the total number of maximum tasks that can be completed.

```
#include<bits/stdc++.h>
using namespace std;
int main() {
#ifdef ONLINE_JUDGE
    freopen("input_2.txt", "r", stdin);
    freopen("output_2.txt", "w", stdout);
#endif
    int n;
    cin >> n;
    vector < vector <int> > v(n, vector <int>(3));
    for (int i = 0;i < n;i++) {
        cin >> v[i][0];
        v[i][2] = i;
    }
    for (int i = 0;i < n;i++) {
        cin >> v[i][1];
    }
    sort(v.begin(), v.end(), [&](vector <int> a, vector <int> b) {
        if (a[1] == b[2])
            return a[0] < b[0];
        return a[1] < b[1];
    });
    priority_queue < pair <int, int> > maxheap;
    int current_time = 0;
    vector <int> selected(n, 0);

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

        if (current_time + v[i][0] <= v[i][1]) {
            current_time += v[i][0];
            maxheap.push({ v[i][0] , v[i][2] });
            selected[v[i][2]] = 1;
        }
        else if (maxheap.size()) {
            if (current_time - maxheap.top().first + v[i][0] <= v[i][1] and maxheap.top().first > v[i][0]) {
                selected[maxheap.top().second] = 0;
                maxheap.pop();
                current_time = current_time - maxheap.top().first + v[i][0];
                selected[v[i][2]] = 1;
                maxheap.push({ v[i][0] , v[i][2] });
            }
        }
    }
    int total = 0;
    vector <int> ans;
    for (int i = 0;i < n;i++) {
        total += selected[i];
    }
}
```

```

    if (selected[i])
        ans.push_back(i + 1);
}
cout << "Max number of tasks = " << total << endl << "Selected task numbers : ";
for (int i : ans)
    cout << i << ' ';
}

```

## OUTPUT

main.cpp	input_2.txt	output_2.txt
1	7	
2	2 1 3 2 2 2 1	
3	2 3 8 6 2 5 3	

main.cpp	input_2.txt	output_2.txt
1	Max number of tasks = 4	
2	Selected task numbers : 2 3 6 7	

**Q3) Given an unsorted array of elements, design an algorithm and implement it using a program to find whether majority element exists or not. Also find median of the array. A majority element is an element that appears more than  $n/2$  times, where  $n$  is the size of array.**

**Input Format:** First line of input will give size  $n$  of array. Second line of input will take  $n$  space-separated elements of array.

**Output Format:** First line of output will be 'yes' if majority element exists, otherwise print 'no'. Second line of output will print median of the array.

```
#include<bits/stdc++.h>
using namespace std;
void findMaj(int arr[], int n) {
    int maxcount = 0;
    int index = -1;
    for (int i = 0; i < n; i++) {
        int count = 0;
        for (int j = 0; j < n; j++) {
            if (arr[i] == arr[j])
                count++;
        }
        if (count > maxcount) {
            maxcount = count;
            index = i;
        }
    }
    if (maxcount > n / 2)
        // cout << "Majority Element => " << arr[index] << endl;
        cout << "yes\n";
    else
        cout << "no\n";
}
double findMedian(int a[], int n) {
    sort(a, a + n);
    if (n % 2 != 0)
        return (double)a[n / 2];
    return (double)(a[n - 1 / 2] + a[n / 2]) / 2.0;
}
int main() {
#ifdef ONLINE_JUDGE
    freopen("input_3.txt", "r", stdin);
    freopen("output_3.txt", "w", stdout);
#endif
    int n;
    cin >> n;
    int arr[n];
    for (int i = 0; i < n; i++)
        cin >> arr[i];
    findMaj(arr, n);
    cout << findMedian(arr, n) << endl;
}
```

## OUTPUT

main.cpp	input_3.txt	output_3.txt	main.cpp	input_3.txt	output_3.txt
1	9		1	yes	
2	4 4 2 3 2 2 3 2 2		2	2	

## Week 11

**Q1) Given a set of available types of coins. Let suppose you have infinite supply of each type of coin. For a given value N, you have to Design an algorithm and implement it using a program to find number of ways in which these coins can be added to make sum value equals to N.**

**Input Format:** First line of input will take number of coins that are available. Second line of input will take the value of each coin. Third line of input will take the value N for which you need to find sum.

**Output Format:** Output will be the number of ways.

```
#include<bits/stdc++.h>
using namespace std;
int memo(int i, int current, vector <vector <int> >& dp, vector <int>& coin, int n) {
    if (i == n) {
        return (current == 0);
    }
    if (current < 0)
        return 0;
    if (current == 0)
        return 1;
    if (dp[i][current] != -1)
        return dp[i][current];
    return dp[i][current] = memo(i + 1, current, dp, coin , n) + memo(i, current - coin[i], dp, coin , n);
}
int main() {
#ifdef ONLINE_JUDGE
    freopen("input2.txt", "r", stdin);
    freopen("output2.txt", "w", stdout);
#endif
    int n;
    cin >> n;
    int target;

    vector <int> coins(n);
    for (int i = 0; i < n; i++) {
        cin >> coins[i];
    }
    cin >> target;
    vector <vector <int> > dp(n, vector <int>(target + 1, -1));
    cout << memo(0 , target, dp, coins , n);
}
```

## OUTPUT

main.cpp	input2.txt	output2.txt
1	4	
2	2 5 6 3	
3	10	

main.cpp	input2.txt	output2.txt
1	5	



**Q2) Given a set of elements, you have to partition the set into two subsets such that the sum of elements in both subsets is same. Design an algorithm and implement it using a program to solve this problem.**

**Input Format:** First line of input will take number of elements n present in the set. Second line of input will take n space-separated elements of the set.

**Output Format:** Output will be 'yes' if two such subsets found otherwise print 'no'.

```
#include<bits/stdc++.h>
using namespace std;
int memo(int i, int current, vector < vector <int> >& dp, vector <int>arr, int n) {
    if (current == 0)
        return 1;
    if (current < 0)
        return 0;
    if (i == n)
        return 0;
    if (dp[i][current] != -1)
        return dp[i][current];
    return dp[i][current] = memo(i + 1, current - arr[i], dp, arr, n) || memo(i + 1, current, dp, arr, n);
}
int main() {
#ifdef ONLINE_JUDGE
    freopen("input3.txt", "r", stdin);
    freopen("output3.txt", "w", stdout);
#endif
    int n;
    cin >> n;
    vector <int> arr(n);
    int sum = 0;
    for (int i = 0; i < n; i++) {
        cin >> arr[i];
        sum += arr[i];
    }
    if (sum % 2 == 1) {
        cout << "No" << endl;
    }
    else {
        vector < vector <int> > dp(n + 1, vector <int>(sum / 2 + 1, -1));

        if (memo(0, sum / 2, dp, arr, n)) {
            cout << "Yes" << endl;
        }
        else {
            cout << "No" << endl;
        }
    }
}
```

## OUTPUT

main.cpp	input3.txt	output3.txt
1 7		1 Yes
2 1 5 4 11 5 14 10		2