Program 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>
using namespace std;
int main(){
	int t;
	cin>>t;
	while(t--){
		int n,key,count=0,i;
		cin>>n;
		int arr[n];
		for(i=0;i<n;i++)
			cin>>arr[i];
		cin>>key;
		for(i=0;i<n;i++) {
			count++;
			if(key==arr[i])
				break;
		 }
		 if(i==n)
			cout<<"Not Present ";	else
		cout<<"Present ";
		cout<<count<<endl;
	 }
	return 0;
}
```

Output:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL
8
34 35 65 31 25 89 64 30
89
Present 6
5
977 354 244 546 355
244
Present 3
6
23 64 13 67 43 56
63
Not Present 6
PS C:\Users\mamga\Desktop\New folder>
```

Program 2 : 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(nlogn), where n is the size of input).

Code:

```
#include<iostream>
using namespace std;

int binarySearch(int arr[], int n, int key, int *count)
{
	int l=0,u=n-1,mid,k=0;

	while(l<=u)
	{
		mid=(l+u)/2;
		(*count)++;
		if(arr[mid]==key)
		{
			k=1;
			break;
		}

		else if(arr[mid]>key)
			u=mid-1;

		else
			l=mid+1;
	}
    return k;
}

int main()
{
	int t;
	cin>>t;

	while(t--)
	{

		int n,key,count=0,k;
		cin>>n;
		int arr[n];

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

```
                cin>>key;
                k=binarySearch(arr,n,key,&count);
                if(k==1)
                        cout<<"Present ";
                else
                        cout<<"\nNot Present ";
                        cout<<"\nTotal Comparisions "<<count<<endl;
        }
  return 0;
}
```

Output:

```
PS C:\Users\mamga\Desktop\New folder> cd "c:\Users\mamga\Desktop\New folder\" ; if ($?) { g++ code2.cpp -o code2 } ; if ($?) { .\code2 }
2
5
5 10 15 20 25
5
Present
Total Comparisions 2
7
1 14 15 17 18 20 111
20
Present
Total Comparisions 2
PS C:\Users\mamga\Desktop\New folder>
```

Program 3 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)

Code:

```
#include<iostream>
using namespace std;

int linearSearch(int arr[], int l, int u, int key, int * count)
{
        for (int i = l; i <= u; i++)
        {
          ( * count) ++;
          if (arr[i] == key) return 1;
        }
    return 0;
}

int newSearch(int arr[], int n, int key, int * count)
 {
    int i = 1, k = 0;
    while (i < n)
    {
         ( * count) ++;

        if (arr[i] > key)
            return linearSearch(arr, i / 2, i, key, count);
        i = 2 * i;

        if (2 * i > n - 1)
            return linearSearch(arr, i / 2, n - 1, key, count);
    }
    return k;
}
```

```
int main()
{
	int t;
	cin >> t;
	while (t--)
	{
		int n, key, count = 0, k;
		cin >> n;
		int arr[n];
		for (int i = 0; i < n; i++)
			cin >> arr[i];
		cin >> key;
		k = newSearch(arr, n, key, & count);
		if (k == 1)
			cout << "Present ";
		else cout << "Not Present ";
		cout << count;
	}
	return 0;
}
```

Output:

```
Total Comparisions 2
PS C:\Users\mamga\Desktop\New folder> cd "c:\Users\mamga\Desktop\New folder\" ; if ($?) { g++ code3.cpp -o code3 } ; if ($?) { .\code3 }
3
8
34 35 65 31 25 89 64 30
89
Present 6
5
977 354 244 546 355
244
Not Present 3
6
23 64 13 67 43 56
63
Not Present 3
PS C:\Users\mamga\Desktop\New folder>
```

Program 4 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))

Code:

```
#include<bits/stdc++.h>
using namespace std;
int modifiedBinarySearch(vector < int > & v, int l, int r, int key, char choice)
{
	int mid, result = -1;

	while (l <= r)
	 {
		mid = l + (r - l) / 2;

		if (v[mid] == key)
		{
			result = mid;
			if (choice == 'l')
				r = mid - 1;

			else
				l = mid + 1;
		}

		else if (v[mid] > key)
			r = mid - 1;

		else
			l = mid + 1;
	}

    return result;
}
```

```
int main()
{
    int t;
    cin >> t;
        while (t--)
        {
                int n, a, l, r, key;
                cin >> n;
                vector < int > v;
                for (int i = 0; i < n; i++)
                {
                        cin >> a;
                        v.push_back(a);
                }
                cin >> key;
                l = modifiedBinarySearch(v, 0, n - 1, key, 'l');
                r = modifiedBinarySearch(v, 0, n - 1, key, 'r');
                if (l == -1)
                        cout << "Key Not Present\n";
                else
                        cout << key << " - " << (r - l + 1) << endl;
        }
        return 0;
}
```

Output:

```
PS C:\Users\mamga\Desktop\New folder> cd "c:\Users\mamga\Desktop\New folder\" ; if ($?) { g++ code4.cpp -o code4 } ; if ($?) { .\code4 }
2
10
235 235 278 278 763 764 790 853 981 981
981
981 - 2
15
1 2 2 3 3 5 5 5 25 75 75 75 97 97 97
75
75 - 3
PS C:\Users\mamga\Desktop\New folder>
```

Program 5 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<bits/stdc++.h> using namespace std;

bool findSequence(int arr[], int n) {
   if(n<3) return false;     int i,j,k;
   //arr[i]+arr[j]==arr[k] -> arr[i]+arr[j]-arr[k]==0     for(k=n-1;k>1;k--) {        i=0,j=k-1;
while(i<j) {
          if(arr[i]+arr[j]-arr[k]==0) {
             cout<<i+1<<", "<<j+1<<", "<<k+1<<endl;                return true;
          }
          else if(arr[i]+arr[j]-arr[k]>0)              j--;          else              i++;
      }
   }
   return false;
}

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];       if(!findSequence(arr, n))
         cout<<"No Sequence Found";
   }
   return 0;
}
```

Output:

```
PS C:\Users\mamga\Desktop\New folder> cd "c:\Users\mamga\Desktop\New folder\" ; if ($?) { g++ code5.cpp -o code5 } ; if ($?) { .\code5 }
3
5
1 5 84 209 341
No Sequence Found
10
24 28 48 71 86 89 92 120 194 201
2, 7, 8
15
64 69 82 95 99 107 113 141 171 350 369 400 511 590 666
1, 6, 9
PS C:\Users\mamga\Desktop\New folder>
```

Program 6 Given an array of non-negative 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 <bits/stdc++.h>
using namespace std;
int BinarySearch(vector<int> &v, int l, int r, int key){
   int mid, result = 0;
   while (l <= r){
      mid = l + (r - l) / 2;
      if (v[mid] == key){
         result = 1;
         break;
      }
      else if (v[mid] > key)
         r = mid - 1;
      else
         l = mid + 1;
   }
   return result;
}
int main(){
   int t;
   cin >> t;
   while (t--){
      int n, a, count = 0, key;
      cin >> n;
      vector<int> v;
      for (int i = 0; i < n; i++){
         cin >> a;
         v.push_back(a);
      }
      cin >> key;
      sort(v.begin(), v.end());
      for (int i = n - 1; i > 0; i--)
         count += BinarySearch(v, 0, i - 1, v[i] - key);
      cout << count << endl;
   }
   return 0;
}
```

Output :

```
PS C:\Users\mamga\Desktop\New folder> cd "c:\Users\mamga\Desktop\New folder\" ; if ($?) { g++ code6.cpp -o code6 } ; if ($?) { .\code6 }
2
5
1 51 84 21 31
20
2
10
24 71 16 92 12 28 48 14 20 22
4
4
PS C:\Users\mamga\Desktop\New folder>
```

Program 7 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<bits/stdc++.h>
using namespace std;

void InsertionSort(int arr[], int l, int r, int &compare, int &shifts) {    int i,j,temp;
for(i=l+1;i<=r;i++) {       temp=arr[i];       j=i-1;       while(j>=0){          compare++;
if(arr[j]<temp)             break;          arr[j+1]=arr[j];          shifts++;          j--;
      }
      arr[++j]=temp;
   }
}

int main(){
   int t;    cin>>t;    while(t--){
      int n,compare=0,shifts=0;
      cin>>n;       int arr[n];       for(int i=0;i<n;i++)          cin>>arr[i];
      InsertionSort(arr,0,n-1,compare,shifts);
      for(int i=0;i<n;i++)
         cout<<arr[i]<<" ";
      cout<<"\ncomparisons = "<<compare<<endl;
      cout<<"shifts = "<<shifts<<endl;
   }
   return 0;
}
```

Output:

```
PS C:\Users\mamga\Desktop\New folder> cd "c:\Users\mamga\D
esktop\New folder\" ; if ($?) { g++ code7.cpp -o code7 } ;
 if ($?) { .\code7 }
3
8
-23 65 -31 76 46 89 45 32
-31 -23 32 45 46 65 76 89
comparisons = 19
shifts = 13
10
54 65 34 76 78 97 46 32 51 21
21 32 34 46 51 54 65 76 78 97
comparisons = 34
shifts = 28
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
comparisons = 64
shifts = 54
PS C:\Users\mamga\Desktop\New folder>
```

Program 8 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<bits/stdc++.h>
using namespace std;

void selectionSort(int arr[], int l, int r, int &compare, int &swaps) {    int min;    for(int
i=l;i<r;i++) {        min=i;
      for(int j=i+1;j<=r;j++) {          compare++;          if(arr[min]>arr[j])
            min=j;
      }
      if(min!=i) {
         swap(arr[i],arr[min]);          swaps++;
      }
   }
}

int main(){
   int t;    cin>>t;
   while(t--){
      int n,compare=0,swaps=0;
      cin>>n;       int arr[n];       for(int i=0;i<n;i++)
         cin>>arr[i];
      selectionSort(arr, 0, n-1, compare, swaps);
      for(int i=0;i<n;i++)          cout<<arr[i]<<" ";
      cout<<"\ncomparisons = "<<compare<<endl;       cout<<"swaps ="<<swaps<<endl;
   }
   return 0;
}
```

Output:

```
PS C:\Users\mamga\Desktop\New folder> cd "c:\Users\mamga\D
esktop\New folder\" ; if ($?) { g++ code8.cpp -o code8 } ; if ($?) { .\code8 }
2
8
-13 65 -21 76 46 89 45 12
-21 -13 12 45 46 65 76 89
comparisons = 28
swaps =5
10
64 65 34 76 78 97 46 32 51 21
21 32 34 46 51 64 65 76 78 97
comparisons = 45
swaps =6
PS C:\Users\mamga\Desktop\New folder>
```

Program 9 Given an unsorted array of positive integers, design an algorithm and implement it using a program to find whether there are any duplicate elements the array or not. (use sorting) (Time Complexity = O(n log n))

Code:

```
#include<bits/stdc++.h>
using namespace std;

int main()
{
	int t;
	cin>>t;
	while(t--) {
		int n,i,f=0;
		cin>>n;
		int arr[n];
		for(i=0;i<n;i++)
			cin>>arr[i];
		sort(arr,arr+n);
		for(i=0;i<n-1;i++)
		if(arr[i]==arr[i+1]) {
			f=1;
			break;
		}
		 if(f==0)
			cout<<"NO"<<endl;
		 else
			 cout<<"YES"<<endl;
	 }
	return 0;
}
```

Output:

```
PS C:\Users\mamga\Desktop\New folder> cd "c:\Users\mamga\Desktop\New folder\" ; if ($?) { g++ code9.cpp -o code9 } ; if ($?) { .\code9 }
2
5
28 52 83 14 75
NO
10
75 65 1 65 2 6 86 2 75 8
YES
PS C:\Users\mamga\Desktop\New folder>
```

Program 10: 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 <bits/stdc++.h>
using namespace std;

int MergeSort(int[], int, int, int &);
int Merge(int[], int, int, int, int &);

int MergeSort(int arr[], int l, int r, int &compare)
{
   int inversion = 0, mid;
   if (l < r)
   {
      mid = l + (r - l) / 2;
      inversion = MergeSort(arr, l, mid, compare);
      inversion += MergeSort(arr, mid + 1, r, compare);
      inversion += Merge(arr, l, mid, r, compare);
   }
   return inversion;
}

int Merge(int arr[], int l, int mid, int r, int &compare)
{
   int n1 = mid - l + 1, n2 = r - mid, inversion = 0;
   int L[n1];
   int R[n2];
   int i = l, j = mid + 1, k = 0;
   //creating temp array to store values
   while (i <= mid)
      L[k++] = arr[i++];
   k = 0;
   while (j <= r)
      R[k++] = arr[j++];
   // now merging them
   i = 0, j = 0, k = l;
   while (i < n1 && j < n2)
   {
      compare++;
      if (L[i] <= R[j])
         arr[k++] = L[i++];
      else
      {
```

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

            inversion += n1 - i;
        }
    }
    while (i < n1)
        arr[k++] = L[i++];
    while (j < n2)
        arr[k++] = R[j++];
    return inversion;
}

int main()
{
    int t;
    cin >> t;
    while (t--)
    {
        int n, compare = 0, inversion;
        cin >> n;
        int arr[n];
        for (int i = 0; i < n; i++)
            cin >> arr[i];
        inversion = MergeSort(arr, 0, n - 1, compare);
        for (int i = 0; i < n; i++)
            cout << arr[i] << " ";
        cout << "\nComparisons = " << compare;
        cout << "\nInversions = " << inversion << endl;
    }
    return 0;
}
```

Output:

```
PS C:\Users\mamga\Desktop\New folder> cd "c:\Users\mamga\D
esktop\New folder\" ; if ($?) { g++ code10.cpp -o code10 } ; if ($?) { .\code10 }
2
8
23 65 21 76 46 89 45 32
21 23 32 45 46 65 76 89
Comparisons = 16
Inversions = 13
10
54 65 34 76 78 97 46 32 51 21
21 32 34 46 51 54 65 76 78 97
Comparisons = 22
Inversions = 28
PS C:\Users\mamga\Desktop\New folder>
```

Program 11: 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.

Code:

```
#include<bits/stdc++.h>
using namespace std;

int partition(int arr[], int l, int r, int &compare, int &swaps) {    srand(time(0));    int
pivot=rand()%(r-l+1)+l;    int i=l-1,j=l;    swap(arr[pivot],arr[r]);
   swaps++;    pivot=arr[r];    while(j<=r) {       compare++;       if(arr[j]<=pivot) {
         swaps++;          i++;
         swap(arr[i],arr[j]);
      }       j++;
   }
   return i;
}

void QuickSort(int arr[], int l, int r, int &compare, int &swaps) {    int p;    if(l<r) {
      p=partition(arr, l, r, compare, swaps);       QuickSort(arr, l, p-1, compare, swaps);
      QuickSort(arr, p+1, r, compare, swaps);
   }
}
int main() {
   int t;    cin>>t;    while(t--) {       int n,compare=0, swaps=0;
      cin>>n;       int arr[n];       for(int i=0;i<n;i++)          cin>>arr[i];
      QuickSort(arr,0,n-1,compare,swaps);
      for(int i=0;i<n;i++)
         cout<<arr[i]<<" ";
      cout<<"\nComparisons = "<<compare;
      cout<<"\nSwaps = "<<swaps<<endl;
   }
   return 0;
}
```

Output:

```
PS C:\Users\mamga\Desktop\New folder> cd "c:\Users\mamga\D
esktop\New folder\" ; if ($?) { g++ code11.cpp -o code11 }
 ; if ($?) { .\code11 }
2
8
26 65 21 76 46 89 45 32
21 26 32 45 46 65 76 89
Comparisons = 19
Swaps = 15
10
54 65 34 76 78 97 46 32 51 21
21 32 34 46 51 54 65 76 78 97
Comparisons = 28
Swaps = 22
PS C:\Users\mamga\Desktop\New folder>
```

Program 12: Given an unsorted array of integers, design algorithm and implement it using a program to an find Kth smallest or largest element in the array.(Worst case Time Complexity = O(n))

Code:

```
#include<bits/stdc++.h>
using namespace std;

int partition(int arr[], int l, int r) {
   srand(time(0));    int pivot=rand()%(r-l+1)+l;    int i=l-1,j=l;    swap(arr[pivot],arr[r]);
pivot=arr[r];    while(j<=r) {       if(arr[j]<=pivot) {
         i++;
         swap(arr[i],arr[j]);
      }       j++;
   }
   return i;
}

int KthElement(int arr[], int l, int r, int k) {    int p;    if(l<=r) {       p=partition(arr, l, r);
if(p==k-1)           return p;       else if(p>k-1)
         return KthElement(arr, l, p-1,k);       else
         return KthElement(arr, p+1, r,k);
   }
   return -1;
}

int main() {
   int t;    cin>>t;    while(t--) {       int n,k,kth;       cin>>n;       int arr[n];       for(int
i=0;i<n;i++)           cin>>arr[i];       cin>>k;       if(k>0&&k<=n) {
kth=KthElement(arr,0,n-1,k);           cout<<arr[kth]<<endl;
      }          else
         cout<<"Not Present"<<endl;
   }
   return 0;
}
```

Output:

```
PS C:\Users\mamga\Desktop\New folder> cd "c:\Users\mamga\D
esktop\New folder\" ; if ($?) { g++ code12.cpp -o code12 }
 ; if ($?) { .\code12 }
2
10
123 656 54 765 344 514 765 34 765 234
3
123
15
43 64 13 78 864 346 786 456 21 19 8 434 76 270 601
8
78
PS C:\Users\mamga\Desktop\New folder>
```

Program 13: 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.

Code:

```
#include <bits/stdc++.h>
using namespace std;
int main()
{
   int t;
   cin >> t;
   while (t--)
   {
      int n, count;
      char ch = 'a';
      cin >> n;
      char arr[n];
      int alpha[26] = {0};
      for (int i = 0; i < n; i++)
         cin >> arr[i];
      sort(arr, arr + n);
      for (int i = 0; i < n; i++)
         alpha[arr[i] - 97]++;
      count = alpha[0];
      for (int i = 1; i < 26; i++)
         if (alpha[i] > count)
         {
            count = alpha[i];
            ch = i + 97;
         }
      if (count > 1)
         cout << ch << " - " << count << endl;
      else
         cout << "No Duplicates Present\n";
   }
   return 0;
}
```

Output:

```
PS C:\Users\mamga\Desktop\New folder> cd "c:\Users\mamga\Desktop\New folder\" ; if ($?) { g++ code13.cpp -o code13 } ; if ($?) { .\code13 }
2
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
PS C:\Users\mamga\Desktop\New folder>
```

Program 14: 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)).

Code:

```
#include<bits/stdc++.h>
using namespace std;
int BinarySearch(vector<int> &v , int l, int r, int key) {
 int mid;
while(l<=r){
 mid=l+(r-l)/2;
if(v[mid]==key)
 return mid;
else if(v[mid]>key)
          r=mid-1;
       else
          l=mid+1;
   }
   return -1;
}
int main(){
           int t;    cin>>t;
           while(t--) {
           int i,n,a,k,key;        cin>>n;
           vector<int> v;
           for(i=0;i<n;i++){
                     cin>>a;
                     v.push_back(a);
           }
            cin>>key;
           sort(v.begin(),v.end());
           for(i=0;i<n-1;i++) {
          k=BinarySearch(v,i+1,n-1,key-v[i]);
           if(k!=-1)              break;
      }       if(k!=-1)
          cout<<v[i]<<" "<<v[k]<<endl;
           else
          cout<<"No Such Element Exist\n";
   }
   return 0;
}
```

Output:

```
PS C:\Users\mamga\Desktop\New folder> cd "c:\Users\mamga\Desktop\New folder\" ; if ($?) { g++ code14.cpp -o code14 } ; if ($?) { .\code14 }
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
PS C:\Users\mamga\Desktop\New folder>
```

Program 15: 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))

Code:

```
#include<iostream>
#include<vector>
using namespace std;
int printCommon(vector<int> arr1,vector<int> arr2) {
   int i=0,j=0;
   int len1=arr1.size(),len2=arr2.size();
   while(len1 > i && len2 > j){
      if (arr1[i] < arr2[j]) {
         i++;
      }
      else if(arr2[j] < arr1[i])
         j++;
      else{
         cout<<arr1[i]<<" ";
         i++;
         j++;
      }
   }
}
int main(){
   int l,n,m;
   cin>>n;
   vector<int> a,b;
   for(int i=0;i<n;i++){
      cin>>l;
      a.push_back(l);
   }
   cin>>m;
   for(int i=0;i<m;i++){
      cin>>l;
      b.push_back(l);
   }
   cout<<"Common: ";
   printCommon(a,b);
   return 0;
}
```

Output:

```
PS C:\Users\mamga\Desktop\New folder> cd "c:\Users\mamga\Desktop\New folder\" ; if ($?) { g++ code15.cpp -o code15 } ; if ($?) { .\code15 }
5
5 10 15 20 25
7
3 5 7 10 13 16 25
Common: 5 10 25
PS C:\Users\mamga\Desktop\New folder>
```

Program 16: 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).

Code:

```
#include <bits/stdc++.h>
using namespace std;

bool findpath(int src, int dest, vector<vector<int>> &graph)
{
   if (src == dest)
      return true;

   int n = graph.size();

   vector<bool> visited(n, false);

   visited[src] = true;

   stack<int> s;

   s.push(src);

  while (!s.empty())
   {

      int a = s.top();
      s.pop();

      for (int x : graph[a])
      {

          if (x == dest)
            return true;

         if (!visited[x])
         {
            visited[x] = true;
            s.push(x);
         }
      }
   }
   return false;
}
```

```
int main()
{
   int n, m;
   cin >> n >> m;

   vector<vector<int>> graph(n);

   for (int i = 0; i < m; i++)
   {
      int u, v;
      cin >> u >> v;
      graph[u].push_back(v);
      graph[v].push_back(u);
   }

   int source, dest;
   cin >> source >> dest;

   if (findpath(source, dest, graph))
      cout << "Yes Path Exists";

   else
      cout << "No Such Path Exists";
   return 0;
}
```

Output:

```
PS C:\Users\mamga\Desktop\New folder> cd "c:\Users\mamga\D
esktop\New folder\" ; if ($?) { g++ code16.cpp -o code16 }
 ; if ($?) { .\code16 }
6 8
0 5
0 1
2 4
2 3
3 1
4 3
1 5
1 4
0 4
Yes Path Exists
PS C:\Users\mamga\Desktop\New folder>
```

Program 17: Given a directed graph, design an algorithm and implement it using a program to find whether a cycle exists in the graph or not.

Code:

```
#include <bits/stdc++.h>
using namespace std;

vector<vector<int>> adj;
vector<bool> vis;
vector<int> col;
bool bipart;

//assign color to nodes either 0 or 1
void color(int u, int curr)
{
   if (col[u] != -1 and col[u] != curr)
   {
      bipart = false;
      return;
   }
   col[u] = curr;
   if (vis[u])
      return;
   vis[u] = true;
   for (auto i : adj[u])
   {
      color(i, curr xor 1);
   }
}
```

```
int main()
{
   int n, m;
   cin >> n >> m;
   adj = vector<vector<int>>(n);
   vis = vector<bool>(n, false);
   col = vector<int>(n, -1);
   bipart = true;
   for (int i = 0; i < m; i++)
   {
      int u, v;
      cin >> u >> v;
      adj[u].push_back(v);
      adj[v].push_back(u);
   }
   for (int i = 0; i < n; i++)
   {
      if (!vis[i])
      {
         color(i, 0);
      }
   }
   if (bipart)
      cout << "Yes Bipartite";
   else
      cout << "Not Bipartite";
   return 0;
}
```

Output:

```
PS C:\Users\mamga\Desktop\New folder> cd "c:\Users\mamga\D
esktop\New folder\" ; if ($?) { g++ code17.cpp -o code17 }
 ; if ($?) { .\code17 }
6 6
0 1
1 2
2 3
3 4
4 5
0 5
Yes Bipartite
PS C:\Users\mamga\Desktop\New folder>
```

Program 18: Given a directed graph, design an algorithm and implement it using a program to find whether cycle exists in the graph or not.

Code:

```
#include <bits/stdc++.h>
using namespace std;

bool iscycle(int src, vector<vector<int>> &adj, vector<bool> &vis, int parent)
{
   vis[src] = true;
   for (auto i : adj[src])
   {
      if (i != parent)
      {
         if (vis[i])
            return true;
         if (!vis[i] and iscycle(i, adj, vis, src))
         {
            return true;
         }
      }
   }
   return false;
}
```

```
int main()
{
   int n, m;
   cin >> n >> m;
   vector<vector<int>> adj(n);
   vector<bool> vis(n, false);
   bool cycle = false;
   for (int i = 0; i < m; i++)
   {
      int u, v;
      cin >> u >> v;
      adj[u].push_back(v);
      adj[v].push_back(u);
   }
   // set node 0 as visited
   vis[0] = true;
   for (int i = 0; i < n; i++)
   {
      if (!vis[i] and iscycle(i, adj, vis, -1))
      {
         cycle = true;
         break;
      }
   }
   if (cycle)
      cout << "Yes Cycle Exists";
   else
      cout << "Cycle Not Present";
   return 0;
}
```

Output:

```
PS C:\Users\mamga\Desktop\New folder> cd "c:\Users\mamga\D
esktop\New folder\" ; if ($?) { g++ code18.cpp -o code18 }
 ; if ($?) { .\code18 }
5 6
0 1
1 3
0 3
2 4
0 2
2 4
Yes Cycle Exists
PS C:\Users\mamga\Desktop\New folder>
```

Program 19: 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).

Code:

```
#include <bits/stdc++.h>
using namespace std;

void path(vector<int>& parent, int j){
        if (parent[j] == - 1)
        {
                cout<<j;
                        return;
       }
   printf("%d ", j);
   path(parent, parent[j]);
}

int main()
{
   int n, e;
   cin >> n >> e;
   vector<vector<pair<int, int>>> graph(n + 1);
   for (int i = 0; i < e; i++)
   {
      int s, d, w;
      cin >> s >> d >> w;
      graph[s].push_back({d, w});
      graph[d].push_back({s, w});
   }
   vector<int> dist(n + 1, INT_MAX);
   set<pair<int, int>> s;
   int source;
   cin >> source;
   dist[source] = 0;
   s.insert({0, source});
   vector<int> parent(n + 1, -1);
   while (!s.empty())
```

```
    {
        auto x = *(s.begin());
        s.erase(x);

            for (auto it : graph[x.second])
            {
            if (dist[it.first] > dist[x.second] + it.second)
            {
                s.erase({dist[it.first], it.first});
                dist[it.first] = dist[x.second] + it.second;
                s.insert({dist[it.first], it.first});
                parent[it.first] = x.second;
            }
        }
    }
    for (int i = 1; i < n + 1; i++)
    {
        path(parent, i);
        cout << " : " << dist[i] << endl;
    }
    return 0;
}
```

Output:

```
PS C:\Users\mamga\Desktop\New folder>
                                    > cd "c:\Users\mamga\D
esktop\New folder\" ; if ($?) { g++ code19.cpp -o code19 }
 ; if ($?) { .\code19 }
5 6
1 2 4
2 3 2
3 1 1
2 5 4
3 4 4
4 5 4
1
1 : 0
2 3 1 : 3
3 1 : 1
4 3 1 : 5
5 2 3 1 : 7
PS C:\Users\mamga\Desktop\New folder>
```

Program 20: Design an algorithm and implement it using a program to solve previous question's problem using Bellman- Ford's shortest path algorithm.

Code:

```
#include <bits/stdc++.h>
using namespace std;

void path(vector<int> &parent, int j)
{
   if (parent[j] == -1)
   {
      cout << j;
      return;
   }
   printf("%d ", j);
   path(parent, parent[j]);
}

int main()
{
   int n, e;
   cin >> n >> e;
   vector<vector<int>> edges;
   for (int i = 0; i < e; i++)
   {
      int u, v, w;
      cin >> u >> v >> w;
      edges.push_back({u, v, w});
      edges.push_back({v, u, w});
   }

   vector<int> parent(n + 1, -1); //initialize distance array
   vector<int> dist(n+1, 1e9);

   int src;
   cin >> src;    //input source
   dist[src] = 0; // initialize source distance to 0

   //iterate n-1 times to relax each edge
   bool negative_cycle;
   for(int i=1;i<n;i++)
   {

      negative_cycle = false; //to detect -ve cycle
```

```
    for (auto it : edges)
    {
        int u, v, w;
        u = it[0];
        v = it[1];
        w = it[2];
        if (dist[v] > dist[u] + w)
        {
            dist[v] = dist[u] + w;
            parent[v] = u;
            negative_cycle = true;
        }
    }
  }

  if (negative_cycle)
    cout << "negative cycle present";
  else
  {
    for (int i = 1; i < n + 1; i++)
    {
        path(parent, i);
        cout << " : " << dist[i] << endl;
    }
  }
  return 0;
}
```

Output:

```
PS C:\Users\mamga\Desktop\New folder> cd "c:\Users\mamga\Des
ktop\New folder\" ; if ($?) { g++ code20.cpp -o code20 } ; i
f ($?) { .\code20 }
5 6
1 2 4
2 3 2
3 1 1
2 5 4
3 4 4
4 5 4
1
1 : 0
2 3 1 : 3
3 1 : 1
4 3 1 : 5
5 2 3 1 : 7
PS C:\Users\mamga\Desktop\New folder>
```

Program 21: 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.

Code:

```
#include <bits/stdc++.h>
using namespace std;

#define V 100

#define INF INT_MAX

int arr[100][100];

int shortestpath(int arr[][V], int u, int v, int k, int n)
{

if (k == 0 && u == v)
      return 0;

if (k == 1 && arr[u][v] != INF)
      return arr[u][v];

if (k <= 0)
      return INF;

int res = INF;


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

{

if (arr[u][i] != INF && u != i && v != i) {
          int rec_res = shortestpath(arr, i, v, k - 1, n);
          if (rec_res != INF)
             res = min(res, arr[u][i] + rec_res);
      }
   }
   return res;
}
```

```
int main()
{
   int n;
   cout << "for values INF enter -1" << endl;
   cin >> n;
   int a;
   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;
      }
   }
   int u, v, k;
   cin >> u >> v >> k;

   cout << "weight of the shortest path is " << shortestpath(arr, u - 1, v - 1, k, n) << endl; // 0
indexing is followed
   return 0;
}
```

Output:

```
PS C:\Users\mamga\Desktop\New folder> cd "c:\Users\mamga\D
esktop\New folder\" ; if ($?) { g++ code21.cpp -o code21 }
 ; if ($?) { .\code21 }
for values INF enter -1
4
0 10 3 2
-1 -1 0 7
0 -1 -1 6
-1 -1 -1 0
1 4 2
weight of the shortest path is 9
PS C:\Users\mamga\Desktop\New folder>
```

Program 22: 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).

Code:

```
#include <bits/stdc++.h>
using namespace std;

int main()
{
   int nodes, edges;
   cin >> nodes;
   cin >> edges;

   vector<pair<int, int>> graph[nodes];
   int source, destination, weight;

   for (int i = 0; i < edges; i++)
   {
      cin >> source >> destination >> weight;
      graph[source].push_back(make_pair(destination, weight));
      graph[destination].push_back(make_pair(source, weight));
   }

   int key[nodes]; //to select the min weight
   int parent[nodes];// to store the parent node
   bool mst[nodes]; // to construct the path

   for (int i = 0; i < nodes; i++)
   {
      key[i] = INT_MAX;
      mst[i] = false;
      parent[i] = -1;
   }

   // priority queue APPROACH
   priority_queue<pair<int, int>, vector<pair<int, int>>,
                  greater<pair<int, int>>>
      pq;
   key[0] = 0; //select a node to start from   parent[0] = 0;
   pq.push({0, 0}); //{weight, index of starting node}
   for(int i=0;i< nodes-1;i++)
```

```
    {
        int u = pq.top().second; //get the index of top node
        pq.pop();//remove the node from queue
        mst[u] = true;//set mst as true for the node u

        for (auto it : graph[i])
        {
            int dest = it.first;
            int wt = it.second;
            if (mst[dest] == false && wt < key[dest]) //check if the parent array needs to be
changed
            {
                parent[dest] = u;
                pq.push({key[dest], dest});
                key[dest] = wt;
            }
        }
    }

    int mstwt = 0;
    // to print the list with minimun weight

    for (int i = 0; i < nodes; i++)
        mstwt += key[i];
    cout << "Min Spanning Weight is: " << mstwt;
    return 0;
}
```

Output:

```
PS C:\Users\mamga\Desktop\New folder> cd "c:\Users\mamga\D
esktop\New folder\" ; if ($?) { g++ code22.cpp -o code22 }
 ; if ($?) { .\code22 }
9 15
0 1 18
1 2 10
2 3 4
3 4 10
4 5 5
5 6 1
6 7 6
7 8 5
8 2 1
6 8 9
5 2 3
3 5 5
7 0 9
7 1 6
0 2 10
Min Spanning Weight is: 53
PS C:\Users\mamga\Desktop\New folder>
```

Program 23: Implement the previous problem using Kruskal's algorithm.

Code:

```
#include <bits/stdc++.h>
using namespace std;

vector<int> parent(100);
vector<int> sz(100);

void make_set(int v)
{
   parent[v] = v;
   sz[v] = 1;
}

int find_set(int v)
{
   if (v == parent[v])
      return v;
   return parent[v] = find_set(parent[v]);
}

void union_set(int a, int b)
{
   a = find_set(a);
   b = find_set(b);
   if (a != b)
   { //dont belong to same set
      if (sz[a] < sz[b])
         swap(a, b);
      parent[b] = a;
      sz[a] += sz[b];
   }
}

int main()
{
   int n, e;
   cin >> n >> e;
   for (int i = 0; i < n; i++)
      make_set(i);
   vector<vector<int>> graph;
   for (int i = 0; i < e; i++)
   {
      int u, v, w;
      cin >> u >> v >> w;
      graph.push_back({w, u, v});
      graph.push_back({w, v, u});
   }
```

```
    sort(graph.begin(), graph.end()); //sort according to weight
    int total_weight = 0;
    for (auto i : graph)
    {
        int w = i[0];
        int u = i[1];
        int v = i[2];
        int x = find_set(u);
        int y = find_set(v);
        if (x == y)
        {
            continue;
        }
        else
        {
            total_weight += w;
            union_set(u, v); //add to set
        }
    }
    cout << "Minimum Spanning Weight is: " << total_weight;
    return 0;
}
```

Output:

```
PS C:\Users\mamga\Desktop\New folder> cd "c:\Users\mamga\D
esktop\New folder\" ; if ($?) { g++ code23.cpp -o code23 }
 ; if ($?) { .\code23 }
9 15
0 1 8
1 2 10
2 3 4
3 4 10
4 5 5
5 6 1
6 7 6
7 8 5
8 2 1
6 8 9
5 2 3
3 5 5
7 0 9
7 1 6
0 2 10
Minimum Spanning Weight is: 33
PS C:\Users\mamga\Desktop\New folder>
```

Program 24: Assume that the 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 which have the highest construction cost. Design an algorithm and implement it using a program to find the maximum budget required for the project.

Code:

```
#include <bits/stdc++.h>
using namespace std;

bool compare(const pair<int, int> &a, const pair<int, int> &b)
{
   return b.first > a.first;
}

vector<int> parent(100);
vector<int> sz(100);

void make_set(int v)
{
   parent[v] = v;
   sz[v] = 1;
}

int find_set(int v)
{
   if (v == parent[v])
      return v;
   return parent[v] = find_set(parent[v]);
}

void union_set(int a, int b)
{
   a = find_set(a);
   b = find_set(b);
   if (a != b)
   { //dont belong to same set
      if (sz[a] < sz[b])
         swap(a, b);
      parent[b] = a;
      sz[a] += sz[b];
   }
}
int main()
{
   int n, e;
   cin >> n >> e;
   for (int i = 0; i < n; i++)
```

```
    make_set(i);
  vector<vector<int>> graph;
  for (int i = 0; i < e; i++)
  {
    int u, v, w;
    cin >> u >> v >> w;
    graph.push_back({w, u, v});
    graph.push_back({w, v, u});
  }
  sort(graph.rbegin(), graph.rend()); //sort according to weight
  int total_weight = 0;
  for (auto i : graph)
  {
    int w = i[0];
    int u = i[1];
    int v = i[2];
    int x = find_set(u);
    int y = find_set(v);
    if (x == y)
    {
      continue;
    }
    else
    {
      total_weight += w;
      union_set(u, v); //add to set
    }
  }
  cout << "Maximum Spanning Weight is: " << total_weight;
  return 0;
}
```

Output:

```
PS C:\Users\mamga\Desktop\New folder> cd "c:\Users\mamga\D
esktop\New folder\" ; if ($?) { g++ code24.cpp -o code24 }
 ; if ($?) { .\code24 }
9 15
0 1 8
1 2 10
2 3 4
3 4 10
4 5 5
5 6 1
6 7 6
7 8 5
8 2 1
6 8 9
5 2 3
3 5 5
7 0 9
7 1 6
0 2 10
Maximum Spanning Weight is: 63
PS C:\Users\mamga\Desktop\New folder>
```

Program 25: Given a graph, Design an algorithm and implement it using a program to implement FloydWarshall all pair shortest path algorithm.

Code:

```
#include <bits/stdc++.h>
using namespace std;
#define INF 1e9

int main()
{
   int n;
   cout << "for values INF enter -1" << endl;
   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:

```
PS C:\Users\mamga\Desktop\New folder> cd "c:\Users\mamga\D
esktop\New folder\" ; if ($?) { g++ code25.cpp -o code25 }
 ; if ($?) { .\code25 }
for values INF enter -1
5
0 10 5 5 -1
-1 0 5 5 5
-1 -1 -1 0 10
-1 -1 -1 0 20
-1 -1 -1 5 0
Shortest Distance Matrix:
0 10 5 5 15
INF 0 5 5 5
INF INF INF 0 10
INF INF INF 0 20
INF INF INF 5 0
PS C:\Users\mamga\Desktop\New folder>
```

Program 26: 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 xi of item i, where 0 ≤xi≤ 1.

Code:

```
#include <bits/stdc++.h>
using namespace std;

int main()
{
   int n;
   cin >> n;


   vector<double> items(n);
   vector<double> val(n);
   vector<vector<double>> job; //to store pair of


   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], i + 1});
   }
   double k;
   cin >> k;


   sort(job.rbegin(), job.rend()); //sort acc to val per wt
   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.first << " - " << it.second << endl;
    return 0;
}
```

Output:

```
PS C:\Users\mamga\Desktop\New folder> cd "c:\Users\mamga\Desktop\New folder\" ; if ($?) { g++ code26.cpp -o code26 } ; if ($?) { .\code26 }
ktop\New folder\" ; if ($?) { g++ code26.cpp -o code26 } ; i
f ($?) { .\code26 }                                                    [-Wnarrowing]
code26.cpp: In function 'int main()':                  from 'int' to 'double' inside { }
code26.cpp:18:55: warning: narrowing conversion of '(i + 1)'
 from 'int' to 'double' inside { } [-Wnarrowing]           ;                    [-Wnarrowing]
        job.push_back({val[i] / items[i], items[i], i + 1})
;                                                      from 'int' to 'double' inside { }
                                                ~~^~~
code26.cpp:18:55: warning: narrowing conversion of '(i + 1)' from 'int' to 'double' inside { } [-Wnarrowing]
6
6 10 3 5 1 3
6 2 1 8 3 5
16
Maximum Value is: 22.3333
Item - Weight
1 - 5
3 - 6
5 - 4
6 - 1
1 - 3
PS C:\Users\mamga\Desktop\New folder>
```

Program 27: 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).

Code:

```
#include <bits/stdc++.h>
using namespace std;
int main(){
   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();
      ans += e1 + e2;
      minheap.push(e1 + e2);
   }
   cout << ans;
   return 0;
}
```

Output:

```
PS C:\Users\mamga\Desktop\New folder> cd "c:\Users\mamga\D
esktop\New folder\" ; if ($?) { g++ code27.cpp -o code27 }
 ; if ($?) { .\code27 }
10
20 5 55 60 65 47 80 10 63 75
1490
PS C:\Users\mamga\Desktop\New folder>
```

Program 28: 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.

Code:

```
#include <bits/stdc++.h>
using namespace std;
int main(){
   int n;
   cin >> n;
   vector<int> st(n), dline(n);
   vector<vector<int>> activity;
   for (int i = 0; i < n; i++)
      cin >> st[i];
   for (int i = 0; i < n; i++){
      cin >> dline[i];
      activity.push_back({dline[i], st[i], i + 1});
   }
   sort(activity.begin(), activity.end());
   vector<int> selected;
   int count = 0;
   int currentEnd = -1;
   for (int i = 0; i < n; i++){
      if (activity[i][1] > currentEnd){
         count++;
         currentEnd = activity[i][0];
         selected.push_back(activity[i][2]);
      }
   }
   cout << "No. of non-conflictin activities: " << count << endl;
   cout << "List of selected activites: ";
   for (auto i : selected)
      cout << i << " ";
   return 0;
}
```

Output:

```
PS C:\Users\mamga\Desktop\New folder> cd "c:\Users\mamga\D
esktop\New folder\" ; if ($?) { g++ code28.cpp -o code28 }
 ; if ($?) { .\code28 }
10
1 3 0 5 3 5 8 8 2 12
4 5 6 7 9 9 11 12 14 16
No. of non-conflictin activities: 4
List of selected activites: 1 4 7 10
PS C:\Users\mamga\Desktop\New folder> []
```

Program 29: Given a long list of tasks. Each task take 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.

Code:

```
#include <bits/stdc++.h>
using namespace std;

bool compare(pair<int, int> a, pair<int, int> b)
{
   return a.first > b.first;
}

int main()
{
   int n;
   cin >> n;


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


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

   vector<pair<int, int>> jobs;

    int profit, deadline;

   for (int i = 0; i < n; i++)
   {
      jobs.push_back(make_pair(p[i], d[i]));
   }


   sort(jobs.begin(), jobs.end(), compare);

   int maxEndTime = 0;

   for (int i = 0; i < n; i++)
   {
      if (jobs[i].second > maxEndTime)
         maxEndTime = jobs[i].second;
   }
```

```
    vector<int> ans;
    int fill[maxEndTime];
    int count = 0, maxProfit = 0;


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

    for (int i = 0; i < n; i++)
    {
       int j = jobs[i].second - 1;
       while (j >= 0 && fill[i] != -1)
          j--;

       if (j >= 0 && fill[j] == -1)
       {
          fill[j] = i;
          ans.push_back(i);
          count++;
          maxProfit = maxProfit + jobs[i].first;
       }
    }


    cout << "Maximum no of tasks : " << count << endl;
    cout << "Selected task numbers : ";

    for (int i = 0; i < ans.size(); i++)
       cout << ans[i] << " ";

    return 0;
}
```

Output:

```
PS C:\Users\mamga\Desktop\New folder> cd "c:\Users\mamga\D
esktop\New folder\" ; if ($?) { g++ code29.cpp -o code29 }
 ; if ($?) { .\code29 }
7
7
2 1 3 2 2 2 1
2 3 8 6 2 5 3
Maximum no of tasks : 4
Selected task numbers : 0 2 3 6
PS C:\Users\mamga\Desktop\New folder>
```

Program 30: 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.

Code:

```
#include <bits/stdc++.h>
using namespace std;

string majorityElement(int *arr, int n)
{
   int count = 1, max_ele = -1, temp = arr[0], ele, f = 0;


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

      if (temp == arr[i])
      {
         count++;
      }

      else
      {
         count = 1;
         temp = arr[i];
      }

      if (max_ele < count)
      {
         max_ele = count;
         ele = arr[i];

         if (max_ele > (n / 2))
         {
            f = 1;
            break;
         }
      }
   }
   return (f == 1 ? "yes" : "no");
}
```

```
int main()
{
   int n;
   cin >> n;

   int arr[n];

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

    sort(arr, arr + n);

   cout << majorityElement(arr, n) << endl;

   if (n % 2 == 0)
      cout << (arr[n / 2 - 1] + arr[n / 2]) / 2;
   else
      cout << arr[n / 2];
   return 0;
}
```

Output:

```
PS C:\Users\mamga\Desktop\New folder> cd "c:\Users\mamga\D
esktop\New folder\" ; if ($?) { g++ code30.cpp -o code30 }
 ; if ($?) { .\code30 }
9
4 4 2 3 2 2 3 2 2
yes
2
PS C:\Users\mamga\Desktop\New folder>
```