



Laboratory Manual

Analysis And Design of Algorithm (CS-402)

For

Second Year Students
Department: Computer Science & Engineering



Department of Computer Science and Engineering

Vision of CSE Department:

The department envisions to nurture students to become technologically proficient, research competent and socially accountable for the welfare of the society.

Mission of the CSE Department:

- I. To provide high quality education through effective teaching-learning process emphasizing active participation of students.
- II. To build scientifically strong engineers to cater to the needs of industry, higher studies, research and startups.
- III. To awaken young minds ingrained with ethical values and professional behaviors for the betterment of the society.

Program Educational Objectives:

Graduates will be able to

- I. Our engineers will demonstrate application of comprehensive technical knowledge for innovation and entrepreneurship.
- II. Our graduates will employ capabilities of solving complex engineering problems to succeed in research and/or higher studies.
- III. Our graduates will exhibit team-work and leadership qualities to meet stakeholder business objectives in their careers.
- IV. Our graduates will evolve in ethical and professional practices and enhance socioeconomic contributions to the society.

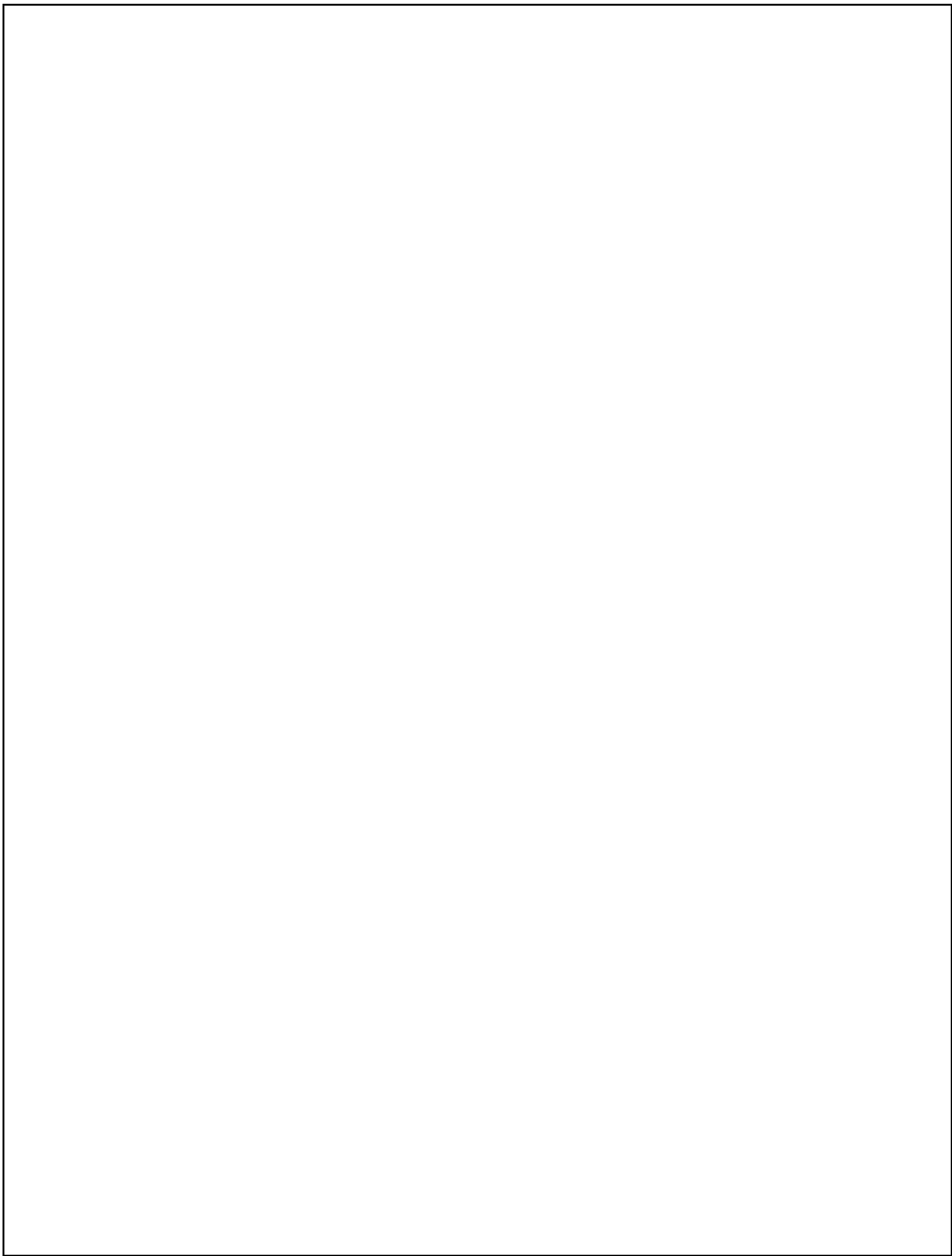
Program Outcomes (POs):

Engineering Graduates will be able to:

1. **Engineering knowledge:** Apply the knowledge of mathematics, science, engineering Fundamentals, and an engineering specialization to the solution of complex engineering problems.
2. **Problem analysis:** Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
3. **Design/development of solutions:** Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
4. **Conduct investigations of complex problems:** Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
5. **Modern tool usage:** Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
6. **The engineer and society:** Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
7. **Environment and sustainability:** Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
8. **Ethics:** Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
9. **Individual and team work:** Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
10. **Communication:** Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
11. **Project management and finance:** Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
12. **Life-long learning:** Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

List of Program

S. No.	List	Course Outcome	Page No.
1	Write a program for Iterative and Recursive Binary Search.	CO1	1-4
2	Write a program for Merge Sort	CO1	5-7
3	Write a program for Quick Sort.	CO1	8-9
4	Write a program for Matrix Multiplication.	CO1	10-12
5	Write a program for Strassen's Matrix Multiplication.	CO1	13-15
6	Write a program for optimal merge pattern.	CO2	16-19
7	Write a program for Huffman coding.	CO2	20-23
8	Write a program for minimum spanning trees using Kruskal's algorithm.	CO2	24-26
9	Write a program for minimum spanning trees using Prim's algorithm.	CO2	27-29
10	Write a program for job sequencing problem with deadline	CO2	30-32
11	Write a program for single sources shortest path algorithm.	CO2	33-35
12	Write a program for Floyd-Warshall algorithm.	CO3	36-38
13	Write a program for Hamiltonian cycle problem.	CO4	39-41
14	Write a program for traveling salesman problem.	CO5	42-44
15	Write a program for fractional knapsack.	CO5	45-47





Program-1

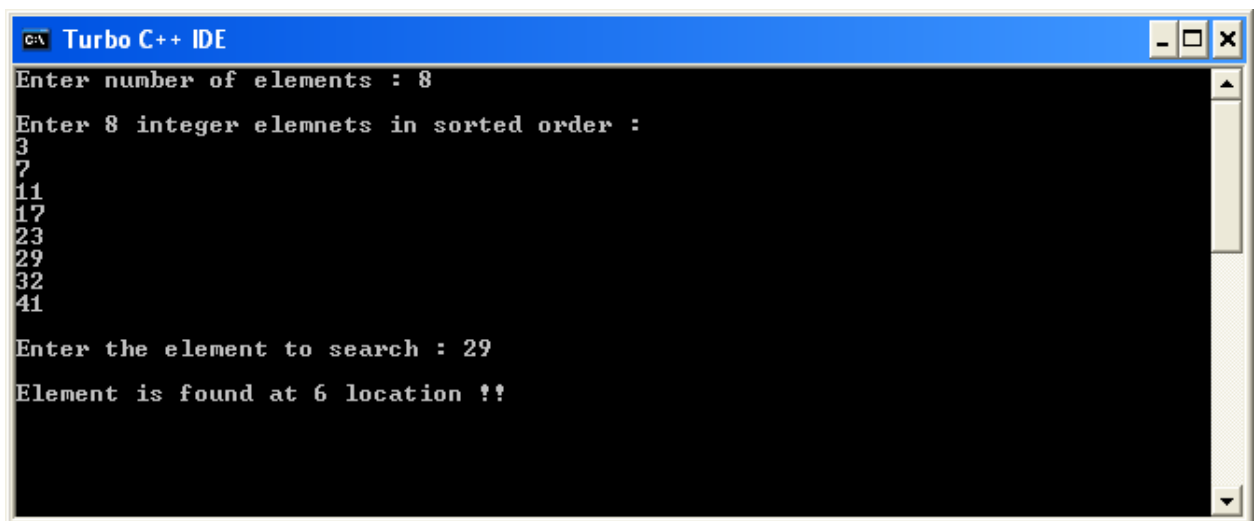
Write a program for Iterative and Recursive Binary Search.

1.a //Iterative binary search

```
#include <iostream.h>
#include<conio.h>
int IterativeBinarySearch(int A[], int key, int l, int r)
{
    while (l <= r)
    {
        int m = (l + r) / 2;
        if (key < A[m])
        {
            r = m - 1;
        }
        else
            if (key > A[m])
            {
                l = m + 1;
            }
        else
        {
            return m;
        }
    }
    return -1;
}
void main( )
{
    clrscr( );
    int arr[20],n,i,e,loc;
    cout<<"Enter number of elements : ";
    cin>>n;
    cout<<"Enter "<<n<<" integer elemnets in sorted order :\n";
    for(i=0;i<n;i++)
        cin>>arr[i];
    cout<<"Enter the element to search : ";
```

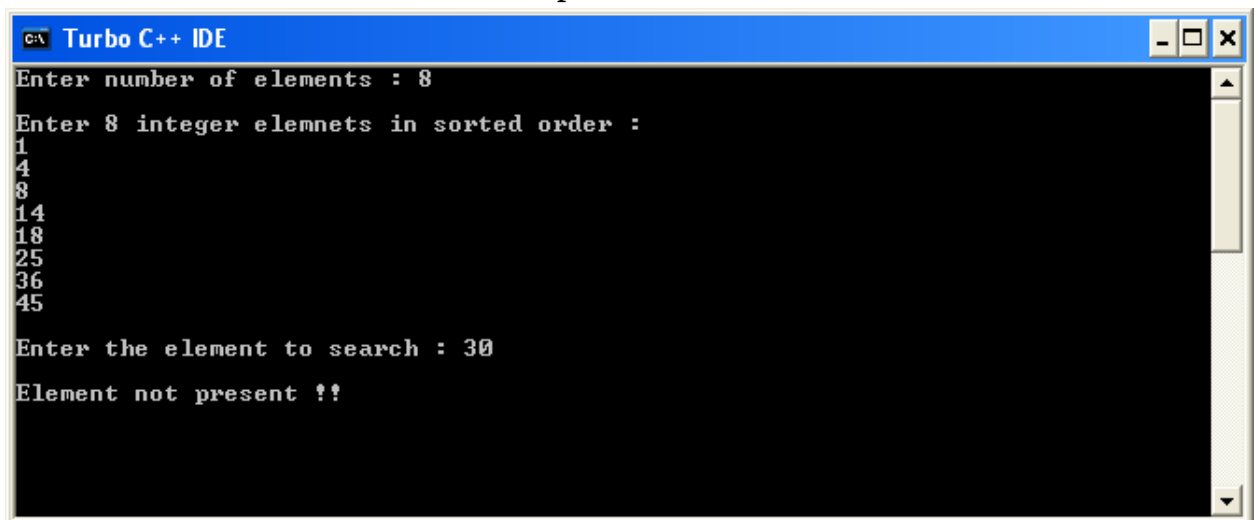
```
cin>>e;
loc = IterativeBinarySearch(arr, e, 0, n-1);
if(loc!=-1)
    cout<<"Element not present !!";
else
    cout<<"Element os found at "<<loc+1<<" location";
getch( );
}
```

Output:



```
Turbo C++ IDE
Enter number of elements : 8
Enter 8 integer elemnets in sorted order :
3
7
11
17
23
29
32
41
Enter the element to search : 29
Element is found at 6 location !!
```

In case of positive search



```
Turbo C++ IDE
Enter number of elements : 8
Enter 8 integer elemnets in sorted order :
1
4
8
14
18
25
36
45
Enter the element to search : 30
Element not present !!
```

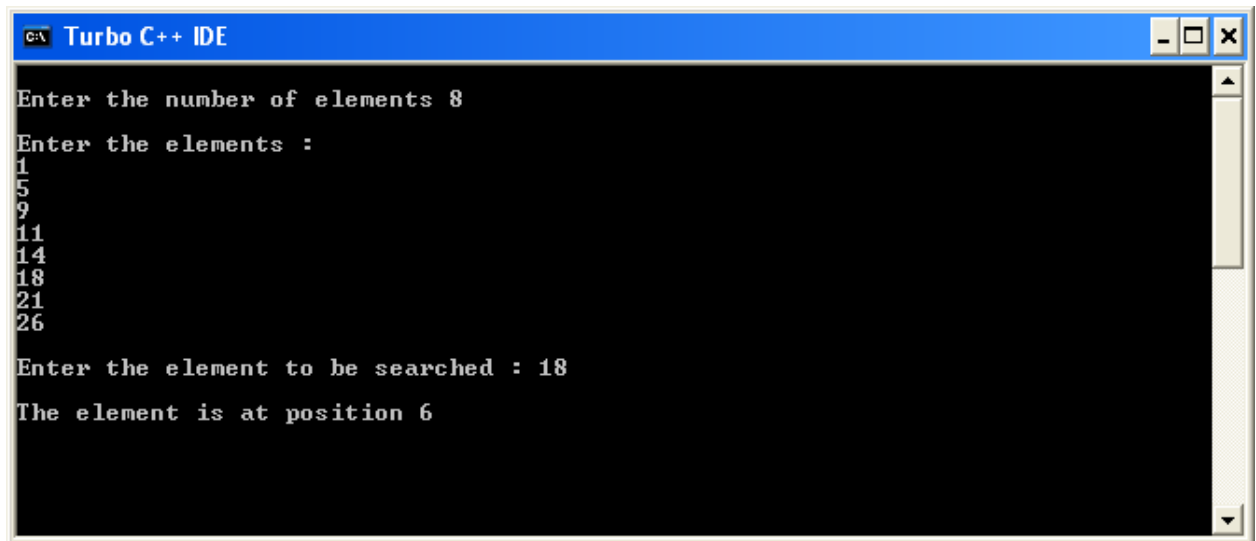
In case of negative search

1..b: //Recursive binary search

```
#include<iostream.h>
#include<conio.h>
int binarysearch(int a[],int n,int low,int high)
{
    int mid;
    if (low > high)
        return -1;
    mid = (low + high)/2;
    if(n == a[mid])
    {
        cout<<"The element is at position "<<mid+1;
        return 0;
    }
    if(n < a[mid])
    {
        high = mid - 1;
        binarysearch(a,n,low,high);
    }
    if(n > a[mid])
    {
        low = mid + 1;
        binarysearch(a,n,low,high);
    }
}
void main( )
{
    clrscr( );
    int a[50];
    int n,e,i,result;
    cout<<"Enter the number of elements ";
    cin>>n;
    cout<<"Enter the elements :\n";
    for(i=0;i<n;i++)
        cin>>a[i];
    cout<<"Enter the element to be searched : ";
    cin>>e;
```

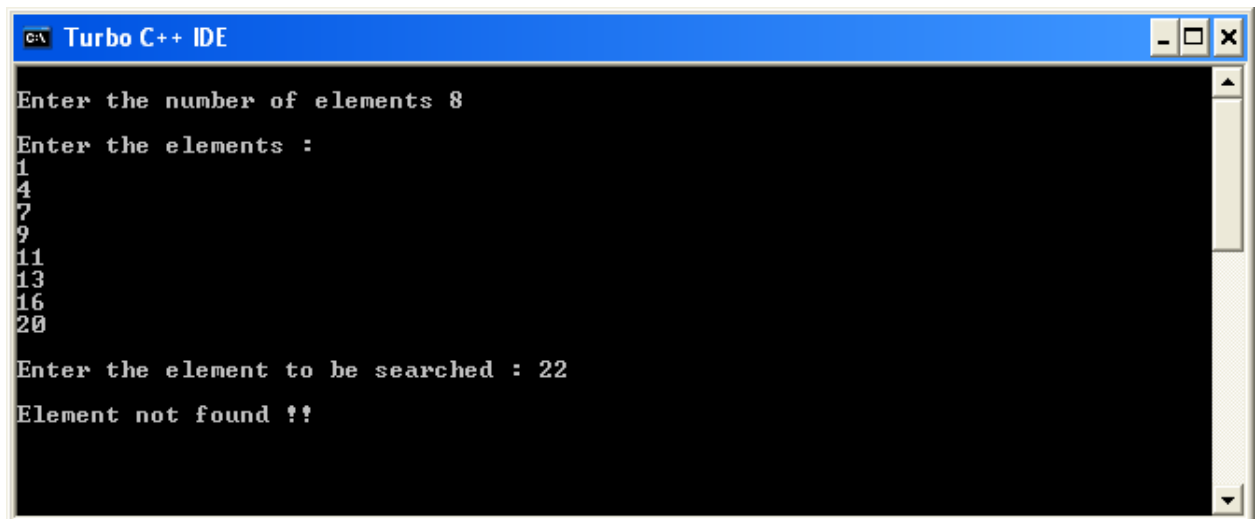
```
result = binarysearch(a,e,0,n-1);  
if(result == -1)  
    cout<<"Element not found !!";  
    getch( );  
}
```

Output:



```
Turbo C++ IDE  
Enter the number of elements 8  
Enter the elements :  
1  
5  
9  
11  
14  
18  
21  
26  
Enter the element to be searched : 18  
The element is at position 6
```

In case of positive search



```
Turbo C++ IDE  
Enter the number of elements 8  
Enter the elements :  
1  
4  
7  
9  
11  
13  
16  
20  
Enter the element to be searched : 22  
Element not found !!
```

In case of negative search

Program-2

Write a program for Merge sort.

```
//merge sort
#include<iostream.h>
#include<conio.h>
void main( )
{
clrscr( );
int a[100],b[100],c[100],n,m,i,j,k,s;
cout<<"\n\nEnter No. of Elements in First Array : ";
cin>>n;
cout<<"\n\nEnter Elements in Sorted Order:\n";
for(i=1;i<=n;i++)
{
cin>>a[i];
}
cout<<"\n\nEnter No. of Elements in Second Array : ";
cin>>m;
cout<<"\n\nEnter Elements in Sorted Order:\n";
for(i=1;i<=m;i++)
{
cin>>b[i];
}
i=1,j=1;
for(k=1;k<=n+m;k++)
{
if(a[i]<b[j])          // Compare Elements of Array A and Array B
{
c[k]=a[i];
i++;
if(i>n)
goto b;
}
else
{
c[k]=b[j];
```

```
j++;  
if(j>m)  
goto a;  
}  
}  
a:  
for(s=i;s<=n;s++)    // Copy the Remaining Elements of Array A to C  
{  
    k++;  
    c[k]=a[s];  
}  
b:  
for(s=j;s<=m;s++)    // Copy the Remaining Elements of Array B to C  
{  
    k++;  
    c[k]=b[s];  
}  
cout<<"\n\nAfter Merging Two Arrays:\n";  
for(k=1;k<=n+m;k++)  
{  
    cout<<c[k]<<endl;  
}  
getch( );  
}
```

Output

```
C:\ Turbo C++ IDE
Enter No. of Elements in First Array : 5
Enter Elements in Sorted Order:
10
24
35
42
67
Enter No. of Elements in Second Array : 8
Enter Elements in Sorted Order:
11
22
38
44
49
51
59
71
After Merging Two Arrays:
10
11
22
24
35
38
42
44
49
51
59
67
71
```

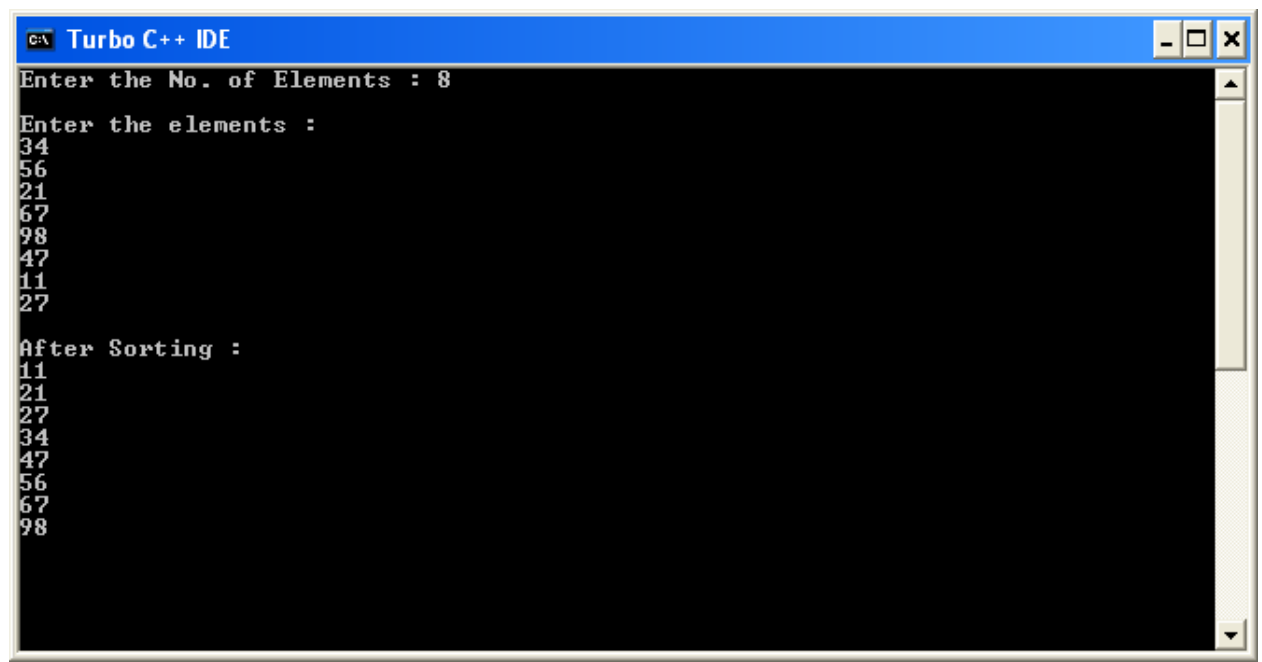
Program-3

Write a program for Quick sort.

```
//quick sort
#include<iostream.h>
#include<conio.h>
int Partition(int a[], int beg, int end)
{
    int p=beg, pivot=a[beg], loc;
    for(loc=beg+1;loc<=end;loc++)
    {
        if(pivot>a[loc])
        {
            a[p]=a[loc];
            a[loc]=a[p+1];
            a[p+1]=pivot;
            p=p+1;
        }
    }
    return p;
}
void QuickSort(int a[], int beg, int end)
{
    if(beg<end)
    {
        int p=Partition(a,beg,end);
        QuickSort(a,beg,p-1);
        QuickSort(a,p+1,end);
    }
}
void main( )
{
    clrscr( );
    int a[100],i,n,beg,end;
    cout<<"Enter the No. of Elements : ";
    cin>>n;
    cout<<"\nEnter the elements : \n";
    for(i=1;i<=n;i++)
```

```
cin>>a[i];
beg=1;
end=n;
QuickSort(a,beg,end);
cout<<"\nAfter Sorting : \n";
for(i=1;i<=n;i++)
    cout<<a[i]<<endl;
getch( );
}
```

Output



The screenshot shows the Turbo C++ IDE window with the following text:

```
C:\ Turbo C++ IDE
Enter the No. of Elements : 8
Enter the elements :
34
56
21
67
98
47
11
27
After Sorting :
11
21
27
34
47
56
67
98
```

Program-4

Write a program for Matrix multiplication

```
#include<iostream.h>
#include<conio.h>
void main( )
{
    clrscr( );
    int a[3][3],b[3][3],c[3][3],i,j,k,s=0;
    //input of first matrix
    cout<<"Enter 9 elements for the first matrix\n";
    for(i=0;i<3;i++)
    {
        for(j=0;j<3;j++)
            cin>>a[i][j];
    }
    //input of second matrix
    cout<<"\nEnter 9 elements for the second matrix\n";
    for(i=0;i<3;i++)
    {
        for(j=0;j<3;j++)
            { cin>>b[i][j];
        }
    }
    //display first matrix
    cout<<"\nMatrix 1 is as follows:\n\n";
    for(i=0;i<3;i++)
    {
        for(j=0;j<3;j++)
            { cout<<a[i][j]<<"\t";
        }
        cout<<endl;
    }
    //display Second matrix
    cout<<"\nMatrix 2 is as follows:\n\n";
    for(i=0;i<3;i++)
    {
        for(j=0;j<3;j++)
```

```
{cout<<b[i][j]<<"\t";
}
cout<<endl;
}
//calculate multiplication
for(i=0;i<3;i++)
{
for(j=0;j<3;j++)
{
for(k=0;k<3;k++)
{
s+=a[i][k]*b[k][j];
c[i][j]=s;
}
s=0;
}
}
//display multiplication
cout<<"\nMatrix multiplication is as follows:\n\n";
for(i=0;i<3;i++)
{
for(j=0;j<3;j++)
{ cout<<c[i][j]<<"\t";
}
cout<<endl;
}
getch( );
}
```

Output

```
Turbo C++ IDE
Enter 9 elements for the first matrix
1
1
1
1
1
1
1
1
1
Enter 9 elements for the second matrix
2
2
2
2
2
2
2
2
2
Matrix 1 is as follows:
1      1      1
1      1      1
1      1      1
Matrix 2 is as follows:
2      2      2
2      2      2
2      2      2
Matrix multiplication is as follows:
6      6      6
6      6      6
6      6      6
```

Program-5

Write a program for Strassen's Matrix multiplication

```
#include<iostream.h>
#include<conio.h>
void main()
{
    clrscr();
    int a[2][2],b[2][2],c[2][2],i,j;
    int m1,m2,m3,m4,m5,m6,m7;
    cout<<"Enter the 4 elements of first matrix: \n";
    for(i=0;i<2;i++)
        for(j=0;j<2;j++)
            cin>>a[i][j];
    cout<<"\nEnter the 4 elements of second matrix: \n";
    for(i=0;i<2;i++)
        for(j=0;j<2;j++)
            cin>>b[i][j];
    cout<<"\nThe first matrix is\n\n";
    for(i=0;i<2;i++)
    {
        for(j=0;j<2;j++)
            cout<<a[i][j]<<" ";
        cout<<endl;
    }
    cout<<"\nThe second matrix is\n\n";
    for(i=0;i<2;i++)
    {
        for(j=0;j<2;j++)
            cout<<b[i][j]<<" ";
        cout<<endl;
    }
    m1= (a[0][0] + a[1][1])*(b[0][0]+b[1][1]);
    m2= (a[1][0]+a[1][1])*b[0][0];
    m3= a[0][0]*(b[0][1]-b[1][1]);
    m4= a[1][1]*(b[1][0]-b[0][0]);
    m5= (a[0][0]+a[0][1])*b[1][1];
    m6= (a[1][0]-a[0][0])*(b[0][0]+b[0][1]);
```



```
m7= (a[0][1]-a[1][1])*(b[1][0]+b[1][1]);
c[0][0]=m1+m4-m5+m7;
c[0][1]=m3+m5;
c[1][0]=m2+m4;
c[1][1]=m1-m2+m3+m6;
cout<<"\nMatrix obtained after multiplication: \n\n";
for(i=0;i<2;i++)
{
    for(j=0;j<2;j++)
        cout<<c[i][j]<<" ";
    cout<<endl;
}
getch();
}
```

Output

```
C:\ Turbo C++ IDE
Enter the 4 elements of first matrix:
1
1
1
1
1
Enter the 4 elements of second matrix:
2
2
2
2
The first matrix is
1 1
1 1
The second matrix is
2 2
2 2
Matrix obtained after multiplication:
4 4
4 4
```



Program-6

Write a program for optimal merge patterns.

```
#include<iostream.h>
#include<conio.h>
void main( )
{
    clrscr( );
    int i,k,a[10],c[10],n,l;
    cout<<"Enter the no. of elements\t";
    cin>>n;
    cout<<"\nEnter the sorted elments for optimal merge pattern";
    for(i=0;i<n;i++)
    {
        cout<<"\t";
        cin>>a[i];
    }
    i=0;
    k=0;
    c[k]=a[i]+a[i+1];
    i=2;
    while(i<n)
    {
        k++;
        if(((c[k-1]+a[i])<=(a[i]+a[i+1])))
        {
            c[k]=c[k-1]+a[i];
        }
        else
```

```
{
c[k]=a[i]+a[i+1];
i=i+2;
while(i<n)
{
k++;
if((c[k-1]+a[i])<=(c[k-2]+a[i]))
{
c[k]=c[k-1]+a[i];
}
else
{
c[k]=c[k-2]+a[i];
}
i++;
}
i++;
}
k++;
c[k]=c[k-1]+c[k-2];
cout<<"\n\nThe optimal sum are as follows.....\n\n";
for(k=0;k<n-1;k++)
{
cout<<c[k]<<"\t";
}
l=0;
for(k=0;k<n-1;k++)
```



```
{  
l=l+c[k];  
}  
cout<<"\n\nThe external path length is = "<<l;  
getch( );  
}
```



Output

```
Turbo C++ IDE
Enter the no. of elements 5
Enter the sorted elements for optimal merge pattern 10
30
45
68
90

The optimal sum are as follows.....
40      85      153      243
The external path length is = 521
```

Program-7

Write a program for Huffman coding

```
#include<stdio.h>
#include<string.h>
#include<conio.h>
#include<stdlib.h>
#define MAX 10
struct link
{
    int freq;
    char ch[MAX];
    struct link* right;
    struct link* left;
};
typedef struct link node;
void sort(node *[], int);
node* create(char[], int);
void sright(node *[], int);
void Assign_Code(node*, int [], int);
void Delete_Tree(node *);

main()
{
    node* ptr, * head;
    int i, n, total = 0, u, c[15];
    char str[MAX];
    node* a[12];
    int freq;
    clrscr( );
    cout<<"Huffman Algorithm\n";
    cout<<"\nEnter the no. of letter to be coded:";
    cin>>n;
    for (i = 0; i < n; i++)
    {
        cout<<"Enter the letter &frequency:";
        cin>> str>>freq;
        a[i] = create(str, freq);
    }
}
```

```

    }
    while (n > 1)
    {
        sort(a, n);
        u = a[0]->freq + a[1]->freq;
        strcpy(str,a[0]->ch);
        strcat(str,a[1]->ch);
        ptr = create(str, u);
        ptr->right = a[1];
        ptr->left = a[0];
        a[0] = ptr;
        sright(a, n);
        n--;
    }
    Assign_Code(a[0], c, 0);
    getch();
    Delete_Tree(a[0]);
}

node* create(char a[], int x)
{
    node* ptr;
    ptr = (node *) malloc(sizeof(node));
    ptr->freq = x;
    strcpy( ptr->ch , a);
    ptr->right = ptr->left = NULL;
    return(ptr);
}

void sort(node* a[], int n)
{
    int i, j;
    node* temp;
    for (i = 0; i < n - 1; i++)
        for (j = i; j < n; j++)
            if (a[i]->freq > a[j]->freq)
            {
                temp = a[i];
                a[i] = a[j];
                a[j] = temp;
            }
}

```

```

}
void sright(node* a[], int n)
{
    int i;
    for (i = 1; i < n - 1; i++)
        a[i] = a[i + 1];
}
void Assign_Code(node* tree, int c[], int n)
{
    int i;
    if ((tree->left == NULL) && (tree->right == NULL))
    {
        cout<<"\n"<<tree->ch<<" code:";
        for (i = 0; i < n; i++)
        {
            cout<<c[i];
        }
        cout<<"\n";
    }
    else
    {
        c[n] = 1;
        n++;
        Assign_Code(tree->left, c, n);
        c[n - 1] = 0;
        Assign_Code(tree->right, c, n);
    }
}
void Delete_Tree(node * root)
{
    if(root!=NULL)
    {
        Delete_Tree(root->left);
        Delete_Tree(root->right);
        free(root);
    }
}

```



Output

```
C:\ Turbo C++ IDE
Huffman Algorithm
Enter the no. of letter to be coded:5
Enter the letter &frequency:1 5
Enter the letter &frequency:2 6
Enter the letter &frequency:3 2
Enter the letter &frequency:4 9
Enter the letter &frequency:5 7

2 code:11
3 code:101
1 code:100
5 code:01
4 code:00
```

Program-8

Write a program for minimum spanning trees using kruskal's algorithm

```
#include<iostream.h>
#include<conio.h>
int cost[10][10],i,j,k,n,m,c,visit,visited[10],l,v,count,count1,vst,p;
void main( )
{ clrscr( );
  int dup1,dup2;
  cout<<"Enter no of vertices : ";
  cin>>n;
  cout<<"\nEnter no of edges : ";
  cin>>m;
  for(k=1;k<=m;k++)
  { cout <<"\nEnter edges a,b and cost c : ";
    cin>>i>>j>>c;
    cost[i][j]=c;
    cost[j][i]=c;
  }
  cout<<"\nProcedure to create minimum spanning tree using kruskal's
algorithm is: \n";
  for(i=1;i<=n;i++)
    for(j=1;j<=n;j++)
      if(cost[i][j]==0)
        cost[i][j]=31999;
  visit=1;
  while(visit<n)
  {
    v=31999;
    for(i=1;i<=n;i++)
      for(j=1;j<=n;j++)
        if(cost[i][j]!=31999 && cost[i][j]<v && cost[i][j]!=-1 )
        { int count =0;
          for(p=1;p<=n;p++)
            { if(visited[p]==i || visited[p]==j)
              count++;
            }
          if(count >= 2)
```

```
{
for(p=1;p<=n;p++)
    if(cost[i][p]!=31999 && p!=j)
        dup1=p;
for(p=1;p<=n;p++)
    if(cost[j][p]!=31999 && p!=i)
        dup2=p;
if(cost[dup1][dup2]==-1)
    continue;
}
l=i;
k=j;
v=cost[i][j];
}
cout <<"\n\nEdge from " <<l <<"-->"<<k;
cost[l][k]=-1;
cost[k][l]=-1;
visit++;
int count=0;
count1=0;
for(i=1;i<=n;i++)
{
    if(visited[i]==l)
        count++;
    if(visited[i]==k)
        count1++;
}
if(count==0)
    visited[++vst]=l;
if(count1==0)
    visited[++vst]=k;
}
getch( );
}
```

Output

```
c:\ Turbo C++ IDE
Enter no of vertices : 7
Enter no of edges : 11
Enter edges a,b and cost c : 1 2 3
Enter edges a,b and cost c : 1 4 5
Enter edges a,b and cost c : 1 3 4
Enter edges a,b and cost c : 1 6 7
Enter edges a,b and cost c : 2 7 9
Enter edges a,b and cost c : 7 3 10
Enter edges a,b and cost c : 3 5 8
Enter edges a,b and cost c : 5 6 11
Enter edges a,b and cost c : 4 6 10
Enter edges a,b and cost c : 4 5 9
Enter edges a,b and cost c : 1 3 4
Procedure to create minimum spanning tree using kruskal's algorithm is:

Edge from 1-->2
Edge from 1-->3
Edge from 1-->4
Edge from 1-->6
Edge from 3-->5
Edge from 2-->7
```

Program-9

Write a program for minimum spanning trees using prim's algorithm

```
#include<iostream.h>
#include<conio.h>
int weight[20][20],visited[20],d[20],p[20];
int v,e;
void creategraph( )
{
    int i,j,a,b,w;
    cout<<"\nEnter number of vertices";
    cin>>v;
    cout<<"\nEnter number of edges";
    cin>>e;
    for(i=1;i<=v;i++)
        for(j=1;j<=v;j++)
            weight[i][j]=0;
    for(i=1;i<=v;i++)
    { p[i]=visited[i]=0;
      d[i]=32767;
    }
    for(i=1;i<=e;i++)
    {
        cout<<"\nEnter edge a,b and weight w:";
        cin>>a>>b>>w;
        weight[a][b]=weight[b][a]=w;
    }
}
void prim( )
{
    int current,totalvisited,mincost,i;
    current=1;d[current]=0;
    totalvisited=1;
    visited[current]=1;
    while(totalvisited!=v)
    { for(i=1;i<=v;i++)
        {
            if(weight[current][i]!=0)
```

```

if(visited[i]==0)
    if(d[i]>weight[current][i])
    {
        d[i]=weight[current][i];
        p[i]=current;
    }
}
mincost=32767;
for(i=1;i<=v;i++)
{
    if(visited[i]==0)
    if(d[i]<mincost)
    {
        mincost=d[i];
        current=i;
    }
}
visited[current]=1;
totalvisited++;
}
mincost=0;
for(i=1;i<=v;i++)
    mincost+=d[i];
cout<<"\nMinimum cost="<<mincost;
cout<<"\nMinimum span tree is";
for(i=1;i<=v;i++)
    cout<<"\nVertex "<<i<<" is connected to"<<p[i];
}
void main( )
{ clrscr( );
  creategraph( );
  prim( );
  getch( );
}

```

Output

```
C:\ Turbo C++ IDE
Enter number of vertices6
Enter number of edges10
Enter edge a,b and weight w:1 2 6
Enter edge a,b and weight w:1 3 1
Enter edge a,b and weight w:1 4 5
Enter edge a,b and weight w:2 3 5
Enter edge a,b and weight w:2 5 3
Enter edge a,b and weight w:3 5 6
Enter edge a,b and weight w:3 6 4
Enter edge a,b and weight w:5 6 6
Enter edge a,b and weight w:4 6 2
Enter edge a,b and weight w:3 4 5
Minimum cost=15
Minimum span tree is
Vertex 1 is connected to0
Vertex 2 is connected to3
Vertex 3 is connected to1
Vertex 4 is connected to6
Vertex 5 is connected to2
Vertex 6 is connected to3
```

Program-10

Write a program for job sequencing problem with deadline

```
#include<conio.h>
#include<iostream.h>
int n,i,j,k,t;
int check(int s[],int p)
{ int ptr=0;
  for(int i=0;i<n;i++)
  { if(s[i]==p)
    ptr++;
  }
  if(ptr==0)
  return 1;
  else
  return 0;
}
int main()
{
  clrscr();
  cout<<"enter the no of jobs    : ";
  cin>>n;
  int slot[10],profit,p[10],d[10];
  for(i=0;i<n;i++)
  {
    cout<<"\nEnter the profit of job "<<i+1<<" :";
    cin>>p[i];
    cout<<"\nEnter the deadline of job "<<i+1<<" : ";
    cin>>d[i];
  }

  for(i=0;i<n;i++)
  for(j=i+1;j<n;j++)
  if(p[i]<p[j])
  { t=p[i];
    p[i]=p[j];
    p[j]=t;
    t=d[i];
```

```

d[i]=d[j];
    d[j]=t;
}

    for(i=0;i<n;i++)
    slot[i]=0;
    for(i=0;i<n;i++)
    for(j=d[i];j>0;j--)
        {if(check(slot,j)==1)
            {slot[i]=j;
            break;}}
    }

    cout<<"\n\nINDEX  PROFIT  DEADLINE  SLOT ALLOTTED ";
    for(i=0;i<n;i++)
    {
        if(slot[i]>0)
            cout<<"\n\n  "<<i+1<<"      "<<p[i]<<"      "<<d[i]<<"      ["<<slot[i]-
1<<" - "<<slot[i]<<"]";
        else
            cout<<"\n\n  "<<i+1<<"      "<<p[i]<<"      "<<d[i]<<"      REJECTED";
        }
        getch();
    }

```

Output

```

C:\ Turbo C++ IDE
enter the no of jobs      : 7
Enter the profit of job 1 :10
Enter the deadline of job 1 : 1
Enter the profit of job 2 :25
Enter the deadline of job 2 : 2
Enter the profit of job 3 :30
Enter the deadline of job 3 : 3
Enter the profit of job 4 :40
Enter the deadline of job 4 : 1
Enter the profit of job 5 :35
Enter the deadline of job 5 : 1
Enter the profit of job 6 :20
Enter the deadline of job 6 : 3
Enter the profit of job 7 :15
Enter the deadline of job 7 : 2

INDEX   PROFIT   DEADLINE   SLOT ALLOTTED
  1       40        1       [0 - 1]
  2       35        1       REJECTED
  3       30        3       [2 - 3]
  4       25        2       [1 - 2]
  5       20        3       REJECTED
  6       15        2       REJECTED
  7       10        1       REJECTED
  
```

Program-11

Write a program for single source shortest path algorithm

```
#include<iostream.h>
#include<conio.h>
int shortest(int ,int);
int cost[10][10],dist[20],i,j,n,k,m,S[20],v,totcost,path[20],p;
void main()
{
    clrscr();
    int c;
    cout <<"Enter no of vertices : ";
    cin >> n;
    cout <<"\nEnter no of edges : ";
    cin >> m;
    for(k=1;k<=m;k++)
    {
        cout <<"\nEnter edge a,b and weight w : ";
        cin >> i >> j >> c; cost[i][j]=c;
    }
    for(i=1;i<=n;i++)
        for(j=1;j<=n;j++)
            if(cost[i][j]==0)
                cost[i][j]=31999;
    cout <<"\nEnter initial vertex : ";
    cin >> v;
    cout << v << "\n\n";
    shortest(v,n);
}
int shortest(int v,int n)
{
    int min;
    for(i=1;i<=n;i++)
    {
        S[i]=0;
        dist[i]=cost[v][i];
    }
    path[++p]=v;
    S[v]=1;
```

```
dist[v]=0;
for(i=2;i<=n-1;i++)
{
    k=-1;
    min=31999;
    for(j=1;j<=n;j++)
    {
        if(dist[j]<min && S[j]!=1)
        {
            min=dist[j]; k=j;
        }
    }
    if(cost[v][k]<=dist[k])
        p=1;
    path[++p]=k;
    for(j=1;j<=p;j++)
        cout<<path[j]; cout <<"\n\n";
    S[k]=1;
    for(j=1;j<=n;j++)
        if(cost[k][j]!=31999 && dist[j]>=dist[k]+cost[k][j] && S[j]!=1)
            dist[j]=dist[k]+cost[k][j];
}
getch();
}
```

Output

```
c:\ Turbo C++ IDE
Enter no of vertices : 6
Enter no of edges : 11
Enter edge a,b and weight w : 1 2 50
Enter edge a,b and weight w : 2 3 10
Enter edge a,b and weight w : 1 3 45
Enter edge a,b and weight w : 1 4 10
Enter edge a,b and weight w : 4 1 20
Enter edge a,b and weight w : 2 4 15
Enter edge a,b and weight w : 5 2 20
Enter edge a,b and weight w : 4 5 15
Enter edge a,b and weight w : 6 5 3
Enter edge a,b and weight w : 5 3 35
Enter edge a,b and weight w : 3 5 30
Enter initial vertex : 1
1
14
145
1452
13
```

Program-12

Write a program for Floyd – warshall algorithm

```
#include <iostream.h>
#include <conio.h>
void floyds(int b[][7])
{ int i, j, k;
  for (k = 0; k < 7; k++)
  {
    for (i = 0; i < 7; i++)
    {
      for (j = 0; j < 7; j++)
      {
        if ((b[i][k] * b[k][j] != 0) && (i != j))
        {
          if ((b[i][k] + b[k][j] < b[i][j]) || (b[i][j] == 0))
          {
            b[i][j] = b[i][k] + b[k][j];
          }
        }
      }
    }
  }
  for (i = 0; i < 7; i++)
  {
    cout<<"\nMinimum Cost With Respect to Node:"<<i<<endl;
    for (j = 0; j < 7; j++)
    {
      cout<<b[i][j]<<"\t";
    }
  }
}
void main()
{
  int b[7][7];
  cout<<"ENTER VALUES OF ADJACENCY MATRIX\n\n";
  for (int i = 0; i < 7; i++)
  {
    cout<<"enter values for "<<(i+1)<<" row"<<endl;
```



```
for (int j = 0; j < 7; j++)  
{  
    cin>>b[i][j];  
}  
}  
floyds(b);  
getch();  
}
```

Output

```

C:\ Turbo C++ IDE
ENTER VALUES OF ADJACENCY MATRIX

enter values for 1 row
0 3 6 0 0 0 0
enter values for 2 row
3 0 2 4 0 0 0
enter values for 3 row
6 2 0 1 4 2 0
enter values for 4 row
0 4 1 0 2 0 4
enter values for 5 row
0 0 4 2 0 2 1
enter values for 6 row
0 0 2 0 2 0 1
enter values for 7 row
0 0 0 4 1 1 0

Minimum Cost With Respect to Node:0
0 3 5 6 7 8
Minimum Cost With Respect to Node:1
3 0 2 3 4 5
Minimum Cost With Respect to Node:2
5 2 0 1 2 3
Minimum Cost With Respect to Node:3
6 3 1 0 2 3
Minimum Cost With Respect to Node:4
8 5 3 2 0 1
Minimum Cost With Respect to Node:5
7 4 2 3 2 0 1
Minimum Cost With Respect to Node:6
8 5 3 3 1 0

```

Program-13

Write a program for Hamiltonian cycle problem

```
#include<iostream.h>
#include<conio.h>
#define V 5
void printSolution(int path[]);
int isSafe(int v, int graph[V][V], int path[], int pos)
{
    if (graph [ path[pos-1] ][ v ] == 0)
        return 0;
    for (int i = 0; i < pos; i++)
        if (path[i] == v)
            return 0;
    return 1;
}
int hamCycleUtil(int graph[V][V], int path[], int pos)
{
    if (pos == V)
    {
        if ( graph[ path[pos-1] ][ path[0] ] == 1 )
            return 1;
        else
            return 0;
    }
    for (int v = 1; v < V; v++)
    {
        if (isSafe(v, graph, path, pos))
        {
            path[pos] = v;
            if (hamCycleUtil (graph, path, pos+1) == 1)
                return 1;
            path[pos] = -1;
        }
    }
    return 0;
}
int hamCycle (int graph[V][V])
```

```
{
    int *path = new int[V];
    for (int i = 0; i < V; i++)
        path[i] = -1;
    path[0] = 0;
    if ( hamCycleUtil(graph, path, 1) == 0 )
    {
        cout<<"\nSolution does not exist";
        return 0;
    }
    printSolution(path);
    return 1;
}

void printSolution(int path[])
{
    cout<<"\n\nSolution Exists: Following is one Hamiltonian Cycle \n";
    for (int i = 0; i < V; i++)
        cout<<path[i];
    cout<<path[0];
    cout<<"\n";
}

int main()
{
    clrscr();
    int graph1[5][5],i,j;
    cout<<"Enter matrix for first graph\n";
    for(i=0;i<5;i++)
        for(j=0;j<5;j++)
            cin>>graph1[i][j];
    hamCycle(graph1);
    getch();
    return 0;
}
```

Output

```
C:\ Turbo C++ IDE
Enter matrix for first graph
1 0 1 1 0
0 0 1 1 0
0 1 0 0 1
1 1 1 0 1
1 0 1 1 0

Solution Exists: Following is one Hamiltonian Cycle
0 2 1 3 4 0
```

Program-14

Write a program for traveling salesman problem

```
#include<iostream.h>
#include<conio.h>
#define MAX 100
#define INFINITY 999
int tsp_dp (int c[][MAX], int tour[], int start, int n);
int main()
{
    clrscr();
    int n;
    int i, j;
    int c[MAX][MAX];
    int tour[MAX];
    int cost;
    cout<<"\nHow many cities to traverse? ";
    cin>>n;
    cout<<"\nEnter the cost matrix: (maximum : 999)\n";
    for (i=0; i<n; i++)
        for (j=0; j<n; j++)
            cin>>c[i][j];
    for (i=0; i<n; i++)
        tour[i] = i;
    cost = tsp_dp (c, tour, 0, n);
    cout<<"\n\nMinimum cost: "<<cost<<"\n\nTour: ";
    for (i=0; i<n; i++)
        cout<<tour[i]+1;
    cout<<"\n";
    getch();
}

int tsp_dp (int c[][MAX], int tour[], int start, int n)
{
    int i, j, k;
    int temp[MAX];
    int mintour[MAX];
    int mincost;
    int ccost;
```

```
if (start == n - 2)
return c[tour[n-2]][tour[n-1]] + c[tour[n-1]][0];
mincost = INFINITY;
for (i = start+1; i<n; i++)
{ for (j=0; j<n; j++)
temp[j] = tour[j];
temp[start+1] = tour[i];
temp[i] = tour[start+1];
if (c[tour[start]][tour[i]] +(ccost = tsp_dp (c, temp, start+1, n)) < mincost)
{
mincost = c[tour[start]][tour[i]] + ccost;
for (k=0; k<n; k++)
mintour[k] = temp[k];
}
}
for (i=0; i<n; i++)
tour[i] = mintour[i];
return mincost;
}
```



Output

```
C:\ Turbo C++ IDE
How many cities to traverse? 3
Enter the cost matrix: (maximum : 999)
150 100 240
300 180 245
145 310 290

Minimum cost: 490
Tour: 1231
```

Program-15

Write a program for fractional knapsack

```
# include<iostream.h>
# include<conio.h>
void knapsack(int n, float weight[], float profit[], float capacity)
{
    float x[20], tp= 0;
    int i, j, u;
    u=capacity;
    for (i=0;i<n;i++)
        x[i]=0.0;
    for (i=0;i<n;i++)
    {
        if(weight[i]>u)
            break;
        else
        {
            x[i]=1.0;
            tp= tp+profit[i];
            u=u-weight[i];
        }
    }
    if(i<n)
        x[i]=u/weight[i];
    tp= tp + (x[i]*profit[i]);
    cout<<"\nThe share taken from each segment is :\n";
    for(i=0;i<n;i++)
        cout<<x[i]<<endl;
    cout<<"\nMaximum profit is: "<< tp;
}
void main()
{
    float weight[20], profit[20], capacity;
    int n, i ,j;
    float ratio[20], temp;
    clrscr();
    cout<<"\nEnter the no. of objects: ";
```

```
cin>>n;
cout<<"\nEnter the wts and profits of each object: \n\n";
for (i=0; i<n; i++)
{
cin>>weight[i]>>profit[i];
}
cout<<"\nEnter the capacity of knapsack: ";
cin>>capacity;
for (i=0; i<n; i++)
{
ratio[i]=profit[i]/weight[i];
}
for(i=0; i<n; i++)
{
for(j=i+1;j< n; j++)
{
if(ratio[i]<ratio[j])
{
temp= ratio[j];
ratio[j]= ratio[i];
ratio[i]= temp;
temp= weight[j];
weight[j]= weight[i];
weight[i]= temp;
temp= profit[j];
profit[j]= profit[i];
profit[i]= temp;
}
}
}
knapsack(n, weight, profit, capacity);
getch();
}
```

Output

```
Turbo C++ IDE

Enter the no. of objects: 5
Enter the wts and profits of each object:
10 4
25 6
20 3
15 5
12 2

Enter the capacity of knapsack: 35
The share taken from each segment is :
1
1
0.4
0
0

Maximum profit is: 11.4
```