VISVESVARAYA TECHNOLOGICAL UNIVERSITY

“JnanaSangama”, Belgaum -590014, Karnataka.

LAB REPORT
on

Analysis and Design of Algorithms

Submitted by

NIKHIL SRIKANTH (1BM20CS096)

in partial fulfilment for the award of the degree of
BACHELOR OF ENGINEERING
in
COMPUTER SCIENCE AND ENGINEERING

B.M.S. COLLEGE OF ENGINEERING
(Autonomous Institution under VTU)
BENGALURU-560019
May-2022 to July-2022

B. M. S. College of Engineering,

Bull Temple Road, Bangalore 560019
(Affiliated To Visvesvaraya Technological University, Belgaum)

Department of Computer Science and Engineering

CERTIFICATE

This is to certify that the Lab work entitled "**Analysis and Design of Algorithms**" carried out by **NIKHIL SRIKANTH (1BM20CS096),** who is bonafide student of **B. M. S. College of Engineering.** It is in partial fulfillment for the award of **Bachelor of Engineering in Computer Science and Engineering** of the Visvesvaraya Technological University, Belgaum during the year 2022. The Lab report has been approved as it satisfies the academic requirements in respect of an **Analysis and Design of Algorithms - (19CS4PCADA)** work prescribed for the said degree.

Name of the Lab-In charge:
Namratha M
Assistant Professor
Department of CSE
BMSCE, Bengaluru

Dr. Jyothi S Nayak
Professor and Head
Department of CSE
BMSCE, Bengaluru

Index Sheet

Course Outcome

CO1	Ability to **analyze** time complexity of Recursive and Non-Recursive algorithms using asymptotic notations.
CO2	Ability to **design** efficient algorithms using various design techniques.
CO3	Ability to **apply** the knowledge of complexity classes P, NP, and NP-Complete and prove certain problems are NP-Complete
CO4	Ability to **conduct** practical experiments to solve problems using an appropriate designing method and find time efficiency.

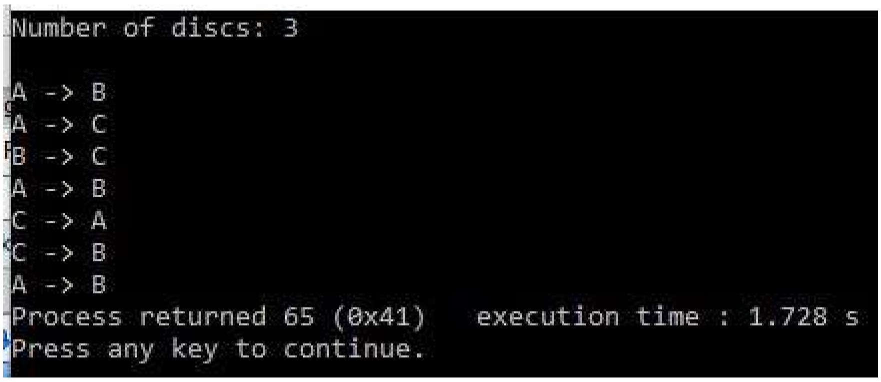
Experiment 1

Write a recursive program to Solve
A: *Towers-of-Hanoi problem*

Program:

```
#include<stdio.h>

void TOH(int,char,char,char);

void main()

{        int n;

         printf("Number of discs: ");

         scanf("%d",&n);

         TOH(n,'A','B','C');

}


void TOH(int n,char x,char y,char z)

{  if(n>0)

   {

         TOH(n-1,x,z,y);

         printf("\n%c -> %c",x,y);

         TOH(n-1,z,y,x);

   }

}
```

Output:



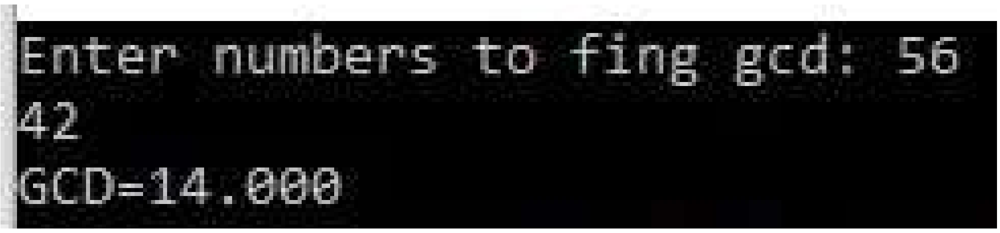
```
Number of discs: 3

A -> B
A -> C
B -> C
A -> B
C -> A
C -> B
A -> B
Process returned 65 (0x41)   execution time : 1.728 s
Press any key to continue.
```

B: *To find GCD*

Program:

```
#include<stdio.h>
int main()
{   double HCF;
    int n1,n2;
    printf("Enter numbers to fing gcd: ");
    scanf("%d %d", &n1,&n2);
    HCF=hcf(n1,n2);
    printf("GCD=%.3f", HCF);
}

hcf(int m,int n)
{
    if(n==0)
        return m;
    else
        return(hcf(n,m%n));
}
```

Output:



```
Enter numbers to fing gcd: 56
42
GCD=14.000
```

Experiment 2

Implement Recursive *Binary search* and *Linear search* and determine the time required to search an element. Repeat the experiment for different values of N and plot a graph of the time taken versus N.

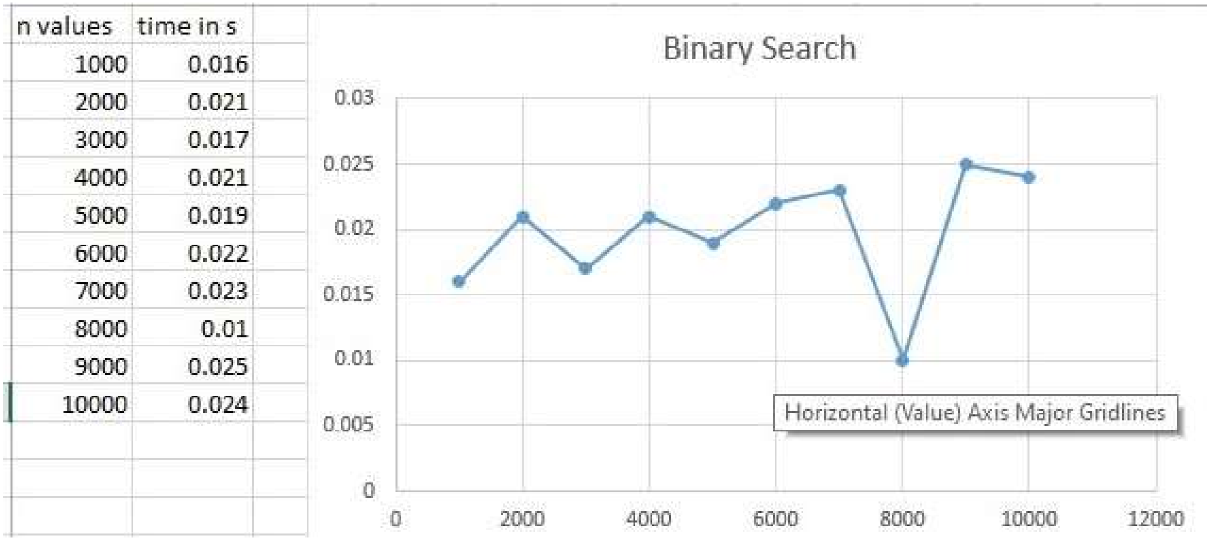
Binary Search:

Program:

```
#include<stdio.h>
#include<stdlib.h>
#include<time.h>

void delay()
{
   int i,j,temp;
   for(i=0;i<500000;i++)
      temp=30/333;
   return;
}
int binary(int l,int h,int arr[],int key)
{
   int m;
   delay();
   m=(l+h)/2;
   if(l>h)
      return -1;
   if(arr[m]==key)
      return (m+1);
   else if(key>arr[m])
```

```
        return (binary(m+1,h,arr,key));
    else
        return (binary(l,m-1,arr,key));
}

int main()
{
    clock_t start,end;
    int m,l,h,flag;
    int n, arr[10000],key,i;
    printf("Enter the value of n: ");
    scanf("%d",&n);
    for(i=0;i<n;i++)
        arr[i]=i;
    key=arr[n-1];
    l=0;
    h=n-1;
    i=0;
    start=clock();
    flag=binary(l,h,arr,key);
    if(flag==-1)
        printf("\nKey not found!");
    else
        printf("\nKey found at %d position",flag);
    end=clock();
    printf("\nTime taken: %f",(double)(end-start)/CLOCKS_PER_SEC);
}
```

Output:



n values	time in s
1000	0.016
2000	0.021
3000	0.017
4000	0.021
5000	0.019
6000	0.022
7000	0.023
8000	0.01
9000	0.025
10000	0.024

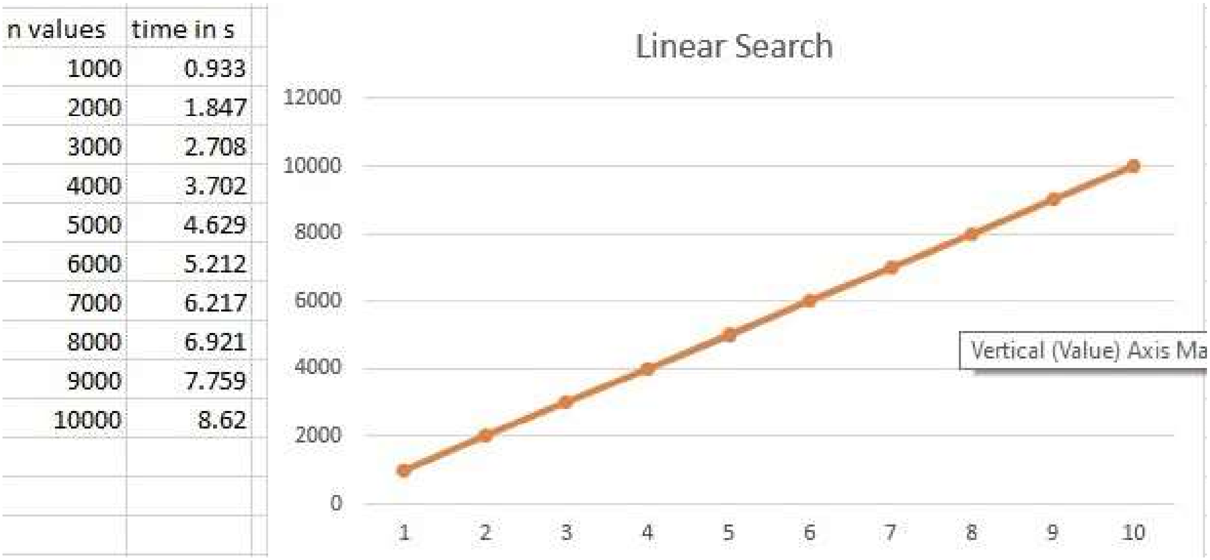
Linear Search:

Program:

```
#include<stdio.h>

#include<stdlib.h>

#include<time.h>


void delay()

{

   int i,j,temp;

   for(i=0;i<500000;i++)

      temp=30/333;

   return;

}


int linear(int arr[],int i,int key,int n)

{

   delay();
```

```
    if(i==n)
        return -1;
    else if(arr[i]==key)
        return (i+1);
    else
        return (linear(arr,(i+1),key,n));
}

int main()
{
    clock_t start,end;
    int flag;
    int n, arr[10000],key,i;
    printf("Enter the value of n: ");
    scanf("%d",&n);
    for(i=0;i<n;i++)
        arr[i]=i;
    key=arr[n-1];
    i=0;
    start=clock();
    flag=linear(arr,i,key,n);
    if(flag==-1)
        printf("\nKey not found!");
    else
        printf("\nKey found at %d position",flag);
    end=clock();
    printf("\nTime taken: %f",(double)(end-start)/CLOCKS_PER_SEC);   }
```

Output:



n values	time in s
1000	0.933
2000	1.847
3000	2.708
4000	3.702
5000	4.629
6000	5.212
7000	6.217
8000	6.921
9000	7.759
10000	8.62

Experiment 3

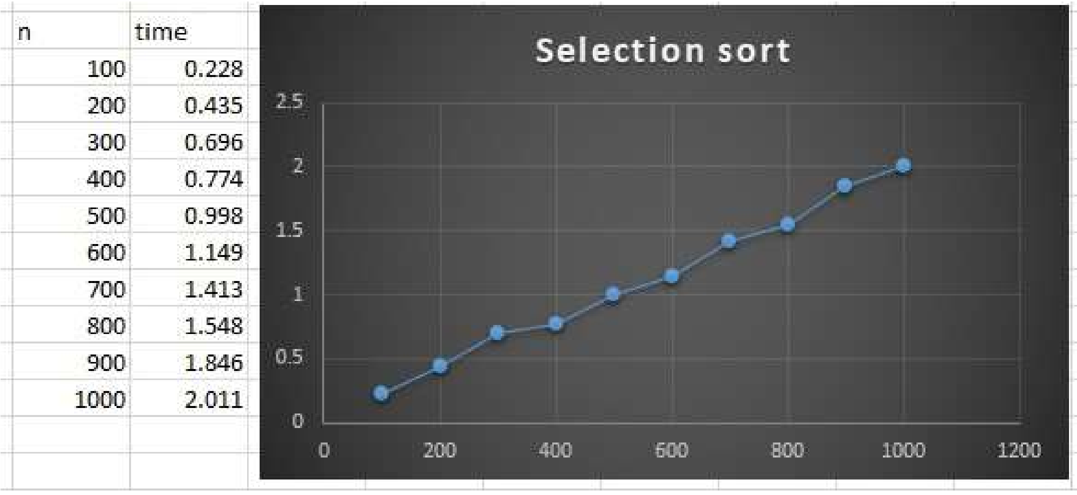
Sort a given set of N integer elements using ***Selection Sort*** technique and compute its time taken. Run the program for different values of N and record the time taken to sort.

Program:

```
#include <stdio.h>

#include<time.h>

#define MAX 20000

void delay()

{

   int i,temp;

   for(i=0;i<1000000;i++)

      temp=32/33233;

   return;

}

int main()

{

         int a[MAX], k, n, i, j, position, swap;

         printf("Enter number of elements:");

         scanf("%d", &n);

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

           {  a[i]= rand(); }

         clock_t start=clock();

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

         {delay();

         position=i;

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

         {
```

```
    if(a[position] > a[j])
        position=j;
    }
    if(position != i)
    {
    swap=a[i];
    a[i]=a[position];
    a[position]=swap;
    }
    }
    clock_t end=clock();
    printf("Sorted Array: ");
    for(i = 0; i < n; i++)
    printf("%d ", a[i]);
    printf("\nExecution time: %f",(double)(end-start)/CLOCKS_PER_SEC);
    return 0;
}
```

Output:



n	time
100	0.228
200	0.435
300	0.696
400	0.774
500	0.998
600	1.149
700	1.413
800	1.548
900	1.846
1000	2.011

Experiment 4

Write program to do the following:

a) Print all the nodes reachable from a given starting node in a digraph using *BFS method*.
b) Check whether a given graph is connected or not using *DFS method*.

a)

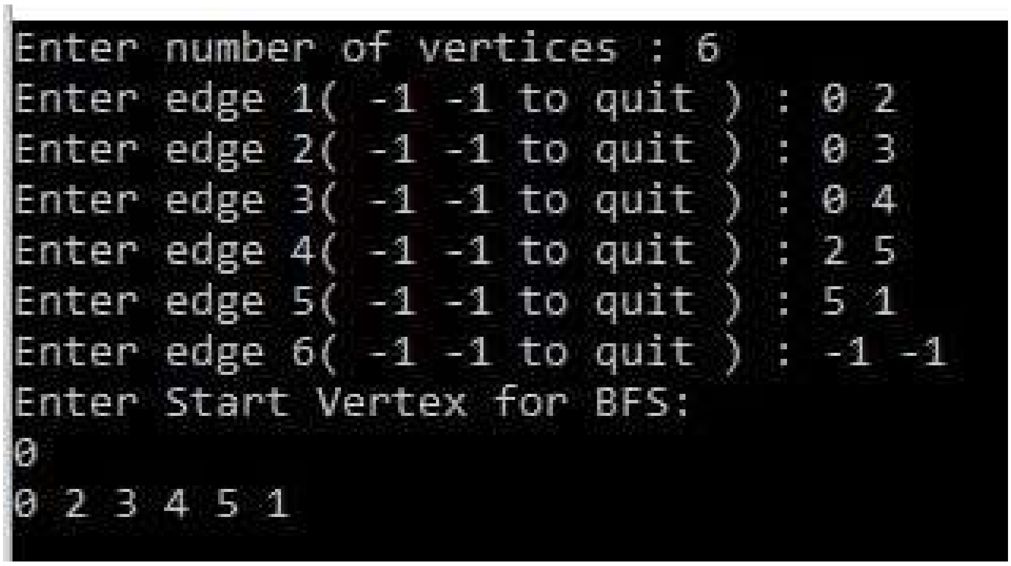
Program:

```
#include<stdio.h>
#include<stdlib.h>

#define MAX 100

#define initial 1
#define waiting 2
#define visited 3

int n;
int adj[MAX][MAX];
int state[MAX];
void create_graph();
void BF_Traversal();
void BFS(int v);

int queue[MAX], front = -1,rear = -1;
void insert_queue(int vertex);
int delete_queue();
int isEmpty_queue();
int main()
{
create_graph();
BF_Traversal();
return 0;
}

void BF_Traversal()
{
int v;
for(v=0; v<n; v++)
state[v] = initial;
printf("Enter Start Vertex for BFS: \n");
scanf("%d", &v);
BFS(v);
}

void BFS(int v)
{
int i;
insert_queue(v);
state[v] = waiting;
while(!isEmpty_queue())
{
v = delete_queue( );
printf("%d ",v);
state[v] = visited;
for(i=0; i<n; i++)
{
if(adj[v][i] == 1 && state[i] == initial)
{
insert_queue(i);
state[i] = waiting;
}
}
}
printf("\n");
}

void insert_queue(int vertex)
{
if(rear == MAX-1)
printf("Queue Overflow\n");
else
{
if(front == -1)
front = 0;
rear = rear+1;
queue[rear] = vertex ;
}
}

int isEmpty_queue()
{
if(front == -1 || front > rear)
```

```c
return 1;
else
return 0;
}

int delete_queue()
{
int delete_item;
if(front == -1 || front > rear)
{
printf("Queue Underflow\n");
exit(1);
}
delete_item = queue[front];
front = front+1;
return delete_item;
}

void create_graph()
{
int count,max_edge,origin,destin;

printf("Enter number of vertices : ");
scanf("%d",&n);
max_edge = n*(n-1);

for(count=1; count<=max_edge; count++)
{
```

```
printf("Enter edge %d( -1 -1 to quit ) : ",count);
scanf("%d %d",&origin,&destin);

if((origin == -1) && (destin == -1))
break;

if(origin>=n || destin>=n || origin<0 || destin<0)
{
printf("Invalid edge!\n");
count--;
}
else
{
adj[origin][destin] = 1;
} } }
```

Output:



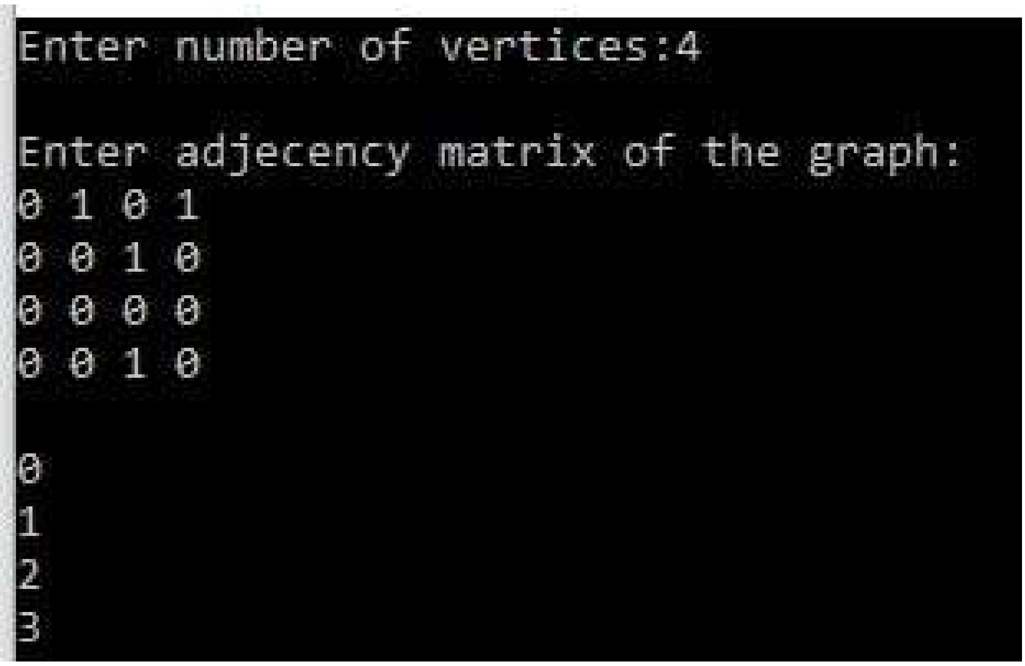
```
Enter number of vertices : 6
Enter edge 1( -1 -1 to quit ) : 0 2
Enter edge 2( -1 -1 to quit ) : 0 3
Enter edge 3( -1 -1 to quit ) : 0 4
Enter edge 4( -1 -1 to quit ) : 2 5
Enter edge 5( -1 -1 to quit ) : 5 1
Enter edge 6( -1 -1 to quit ) : -1 -1
Enter Start Vertex for BFS:
0
0 2 3 4 5 1
```

b)

Program:

```
#include<stdio.h>
#include<stdlib.h>
void DFS(int);
int G[10][10],visited[10],n;

void DFS(int i)
{
    int j;
    printf("\n%d",i);
    visited[i]=1;
    for(j=0;j<n;j++)
       {if(!visited[j]&&G[i][j]==1)
            {DFS(j); }
       }  }
void main()
{
    int i,j;
    printf("Enter number of vertices:");
    scanf("%d",&n);
    printf("\nEnter adjecency matrix of the graph:");
    for(i=0;i<n;i++)
       for(j=0;j<n;j++)
        scanf("%d",&G[i][j]);
    for(i=0;i<n;i++)
        visited[i]=0;
    DFS(0);   }
```

Output:



```
Enter number of vertices:4

Enter adjecency matrix of the graph:
0 1 0 1
0 0 1 0
0 0 0 0
0 0 1 0

0
1
2
3
```

Experiment 5

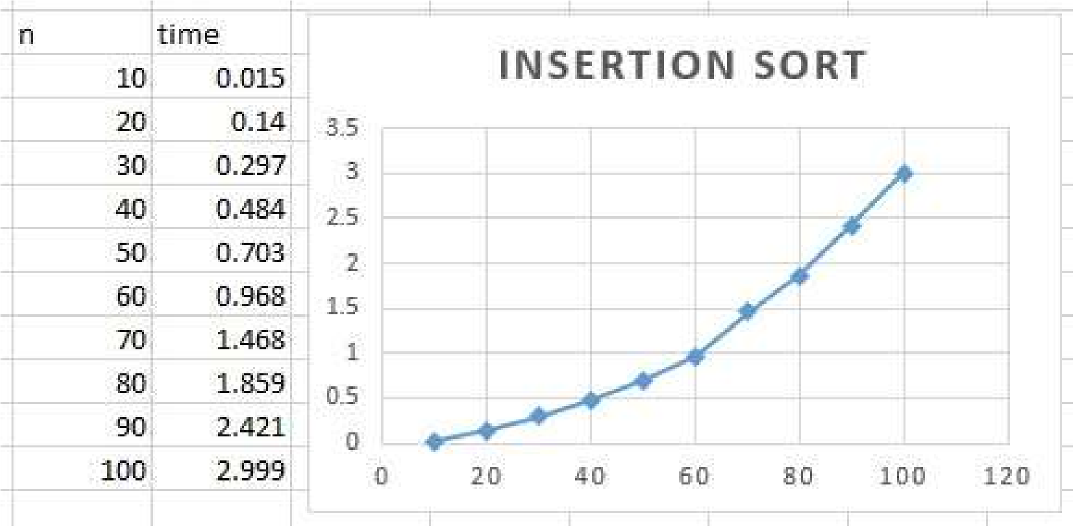
Sort a given set of N integer elements using *Insertion Sort* technique and compute its time taken.

Program:

```
#include <stdio.h>

#include<time.h>

#define MAX 200000

void delay()

{

   int i,j,temp;

   for(i=0;i<1000000;i++)

      temp=32/33233;

   return;

}

void insert(int a[], int n)

{

   int i, j, temp;

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

      temp = a[i];

      j = i - 1;

      while(j>=0 && temp <= a[j])

      {   delay();

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

         j = j-1;

      }

      a[j+1] = temp;

   } }
```

```
void print(int a[], int n)
{
    int i;
    for (i = 0; i < n; i++)
        printf("%d ", a[i]);
}

int main()
{  clock_t start,end;
     int a[MAX],n,i;
    printf("Enter the number of elements: ");
    scanf("%d",&n);
    for(i=0;i<n;i++)
        {   a[i]= rand(); }
    start=clock();
    insert(a, n);
    end=clock();
    printf("\nAfter sorting array elements are - ");
    print(a, n);
    printf("\nTime taken: %f", (double)(end-start)/CLOCKS_PER_SEC );
    printf("\n");
    return 0;
}
```

Output:



n	time
10	0.015
20	0.14
30	0.297
40	0.484
50	0.703
60	0.968
70	1.468
80	1.859
90	2.421
100	2.999

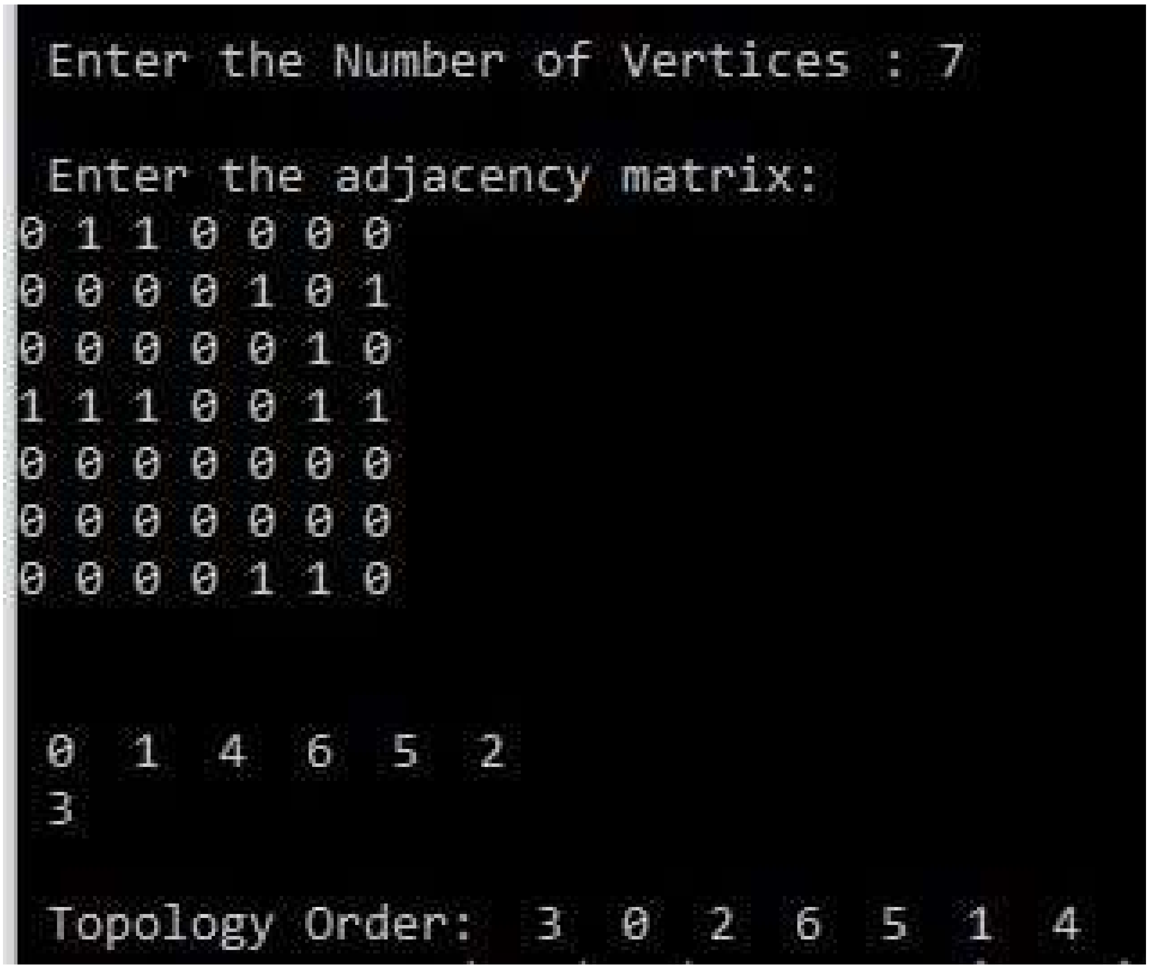
Experiment 6

Write a program to obtain the *Topological ordering of vertices* in a given digraph.

Program:

```
#include<stdio.h>


 void topo(int,int);
 int g[10][10],visited[10],deadend[10],d=-1,n;
 int sortedOrder[10],o,count=0;


 int main()
 {
    printf("\n Enter the Number of Vertices : ");
    scanf("%d",&n);
    o=n;
    printf("\n Enter the adjacency matrix:\n");
       for(int i=0;i<n;i++)
       { for(int j=0;j<n;j++)
                   scanf("%d",&g[i][j]);
       visited[i]=0;
       }
       printf("\n");
       for(int i=0;i<n;i++)
    {
      topo(i,0);
    }
    printf("\n\n Topology Order: ");
    for(int i=0;i<n;i++)
    {
        printf(" %d ",sortedOrder[i]);
    }
    return 0;
}

void topo(int k,int flag)
{
    if(flag==0 && visited[k]==0)
    {
        printf("\n %d ",k);
        sortedOrder[--o]=k;
    }
    else if(flag==0 && visited[k]!=0)
    printf("");
    else
    {
        printf(" %d ",k);
        deadend[++d]=k;
    }
    visited[k]=1;
    for(int j=0;j<n;j++)
        if(visited[j]==0 && g[k][j]==1)
        {
            topo(j,1);
        }
    if(d>=0){
    int temp=sortedOrder[o++];
    for(int k=d;k>=0;k--,--d)
    {
       sortedOrder[--o]=deadend[k];
    }
    sortedOrder[--o]=temp;
    }
 }
```

Output:



```
 Enter the Number of Vertices : 7

 Enter the adjacency matrix:
0 1 1 0 0 0 0
0 0 0 0 1 0 1
0 0 0 0 0 1 0
1 1 1 0 0 1 1
0 0 0 0 0 0 0
0 0 0 0 0 0 0
0 0 0 0 1 1 0


 0  1  4  6  5  2
 3

 Topology Order:  3  0  2  6  5  1  4
```

Experiment 7

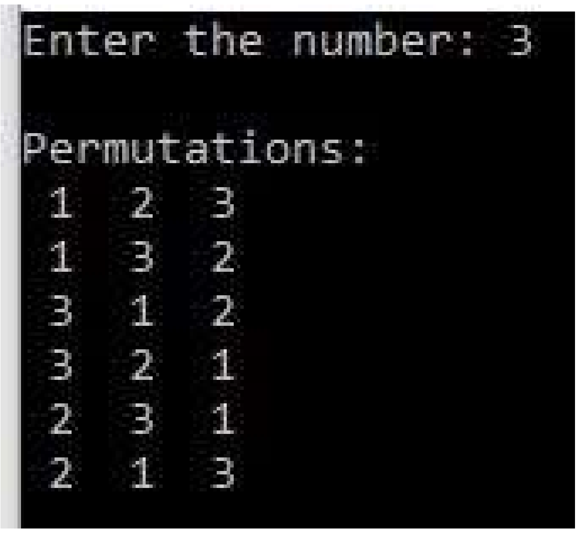
Implement *Johnson Trotter algorithm* to generate permutations.

Program:

```
#include <stdio.h>
#include <stdlib.h>
int flag = 0;
int swap(int *a,int *b)
{
     int t = *a;
    *a = *b;
    *b = t;
}

int search(int arr[],int num,int mobile)
{
    int g;
    for(g=0;g<num;g++)
    {
        if(arr[g] == mobile)
        {
             return g+1;
        }
        else
        {
         flag++;
        }
    }
    return -1;
}
int find_Moblie(int arr[],int d[],int num)
{
    int mobile = 0;
    int mobile_p = 0;
    int i;
    for(i=0;i<num;i++)
    {
        if((d[arr[i]-1] == 0) && i != 0)
        {
            if(arr[i]>arr[i-1] && arr[i]>mobile_p)
            {
                mobile = arr[i];
                mobile_p = mobile;
            }
            else
            {
                flag++ ;    }
        }
        else if((d[arr[i]-1] == 1) & i != num-1)
        {
            if(arr[i]>arr[i+1] && arr[i]>mobile_p)
            {
                mobile = arr[i];
                mobile_p = mobile;
            }
            else
            {
                flag++;
            }
        }
        else
            {
                flag++;
            }
    }
    if((mobile_p == 0) && (mobile == 0))
        return 0;
    else
        return mobile;
}
void permutations(int arr[],int d[],int num)
{
    int i;
    int mobile = find_Moblie(arr,d,num);
    int pos = search(arr,num,mobile);
    if(d[arr[pos-1]-1]==0)
        swap(&arr[pos-1],&arr[pos-2]);
    else
        swap(&arr[pos-1],&arr[pos]);
    for(int i=0;i<num;i++)
    {
        if(arr[i] > mobile)
        {
            if(d[arr[i]-1]==0)
            d[arr[i]-1] = 1;
         else
            d[arr[i]-1] = 0;
      }
   }
   for(i=0;i<num;i++)
   {
      printf(" %d ",arr[i]);
   }
}

int factorial(int k)
{
   int f = 1;
   int i = 0;
   for(i=1;i<k+1;i++)
   {
      f = f*i;
   }
   return f;
}
int main()
{
   int num = 0;
   int i,j,z=0;
   printf("Enter the number: ");
   scanf("%d",&num);
   int arr[num],d[num];
```

```
    z = factorial(num);
    printf("\nPermutations: \n");
    for(i=0;i<num;i++)
    {
        d[i] = 0;
        arr[i] = i+1;
        printf(" %d ",arr[i]);
    }
    printf("\n");
    for(j=1;j<z;j++)
    {
        permutations(arr,d,num);
        printf("\n");
    }
    return 0;
}
```

Output:



```
Enter the number: 3

Permutations:
 1  2  3
 1  3  2
 3  1  2
 3  2  1
 2  3  1
 2  1  3
```

Experiment 8

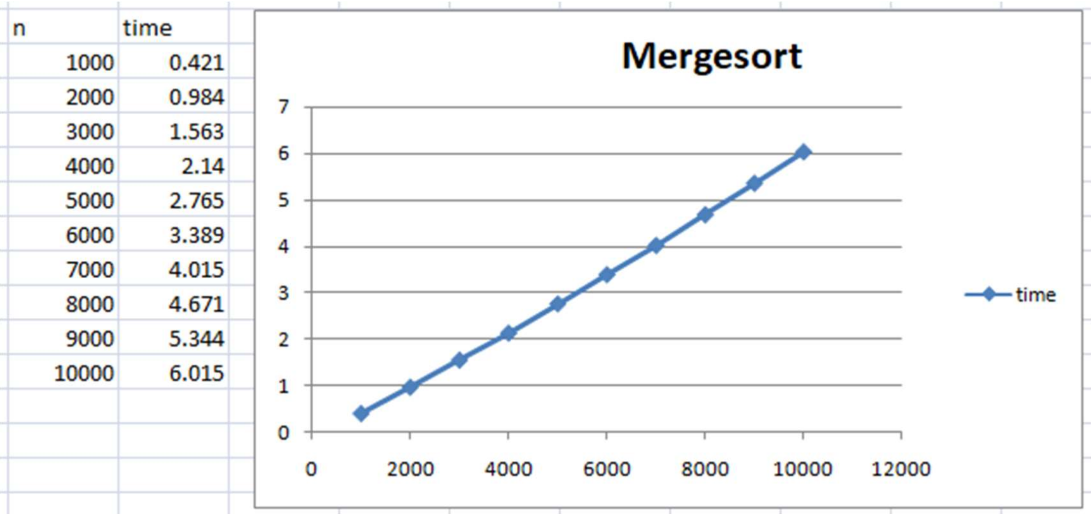
Sort a given set of N integer elements using *Merge Sort* technique and compute its time taken. Run the program for different values of N and record the time taken to sort.

Program:

```
#include<stdio.h>
#include<time.h>

void mergesort(int a[],int i,int j);
void merge(int a[],int i1,int j1,int i2,int j2);
void delay();
int main()
{
   int n,i;
   printf("Enter no of elements:");
   scanf("%d",&n);
   int a[n];
   for(i=0;i<n;i++)
      a[i] = rand();

   clock_t start = clock();
   mergesort(a,0,n-1);
   clock_t end = clock();

   printf("\nSorted array is :");
   for(i=0;i<n;i++)
      printf("%d ",a[i]);
```

```
    printf("\n\nThe total time taken is : %f",(double)(end-start)/CLOCKS_PER_SEC);
    return 0;
}

void mergesort(int a[],int i,int j)
{
int mid;
if(i<j)
{
mid=(i+j)/2;
mergesort(a,i,mid);
mergesort(a,mid+1,j);
merge(a,i,mid,mid+1,j);
}
}

void merge(int a[],int i1,int j1,int i2,int j2)
{
    int temp[10000];
    int i,j,k;
    i=i1;
    j=i2;
    k=0;
    while(i<=j1 && j<=j2)
    {   delay();
        if(a[i]<a[j])
        temp[k++]=a[i++];
        else
        temp[k++]=a[j++];
    }
    while(i<=j1)
        temp[k++]=a[i++];
    while(j<=j2)
        temp[k++]=a[j++];

    for(i=i1,j=0;i<=j2;i++,j++)
        a[i]=temp[j];
}

void delay()
{
    int  i, k;
    for(i=0;i<40000;i++)
        k= 33/333;
}
```

Output:



n	time
1000	0.421
2000	0.984
3000	1.563
4000	2.14
5000	2.765
6000	3.389
7000	4.015
8000	4.671
9000	5.344
10000	6.015

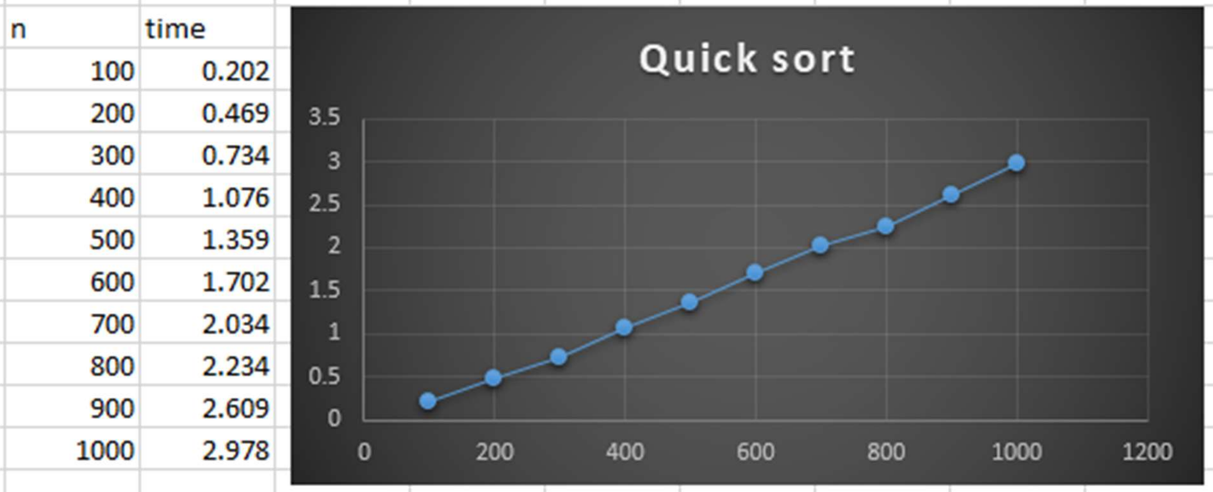
Experiment 9

Sort a given set of N integer elements using ***Quick Sort*** technique and compute its time taken.

Program:

```
#include<stdio.h>

#include<time.h>

#define MAX 15000


void delay()

{

   int i,temp;

   for(i=0;i<1000000;i++)

      temp=32/33233;

}


void quicksort(int number[MAX],int first,int last){

  int i, j, pivot, temp;

  if(first<last){

    pivot=first;

    i=first;

    j=last;

    while(i<j){

      delay();

      while(number[i]<=number[pivot]&&i<last)

      i++;

      while(number[j]>number[pivot])

      j--;
      if(i<j){
        temp=number[i];
        number[i]=number[j];
        number[j]=temp;
      }
    }
    temp=number[pivot];
    number[pivot]=number[j];
    number[j]=temp;
    quicksort(number,first,j-1);
    quicksort(number,j+1,last);
  }
}
int main(){
   clock_t start,end;
  int i, count, a[MAX];
  printf("No. of elements: ");
  scanf("%d",&count);

  for(i=0;i<count;i++)
   {
      a[i]=rand();
   }

  start=clock();
  quicksort(a,0,count-1);
  end=clock();
  printf("Order of Sorted elements: ");
  for(i=0;i<count;i++)
  printf(" %d",a[i]);
  printf("\nExecution time: %f",(double)(end-start)/CLOCKS_PER_SEC);
  return 0;
}
```

Output:



n	time
100	0.202
200	0.469
300	0.734
400	1.076
500	1.359
600	1.702
700	2.034
800	2.234
900	2.609
1000	2.978

Experiment 10

Sort a given set of N integer elements using *Heap Sort* technique and compute its time taken.

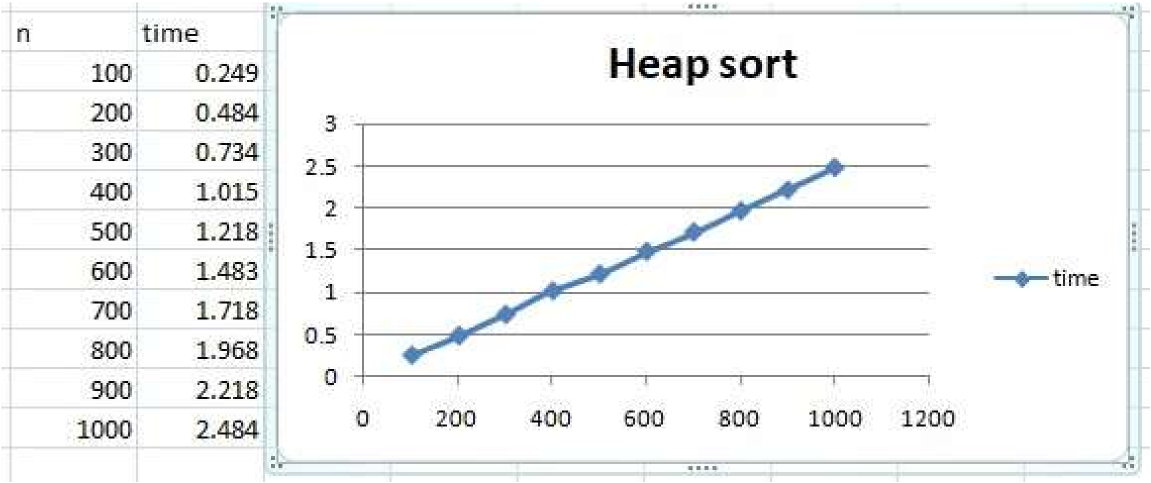
Program:

```
#include <stdio.h>

#include<time.h>

#define max 10000


void swap(int *a, int *b) {

  int temp = *a;

  *a = *b;

  *b = temp;

}


void delay()

{   int i,j,temp;

    for(i=0;i<2000000;i++)

        temp=32/33233;  }


void heapify(int arr[], int n, int i) {

  int largest = i;

  int left = 2 * i + 1;

  int right = 2 * i + 2;

  if (left < n && arr[left] > arr[largest])

    largest = left;

  if (right < n && arr[right] > arr[largest])
```

```
    largest = right;
  if (largest != i) {
    swap(&arr[i], &arr[largest]);
    heapify(arr, n, largest);
  }
}

void heapSort(int arr[], int n) {
  for (int i = n / 2 - 1; i >= 0; i--)
    heapify(arr, n, i);
  for (int i = n - 1; i >= 0; i--) {
     swap(&arr[0], &arr[i]);
    delay();
    heapify(arr, i, 0);
  }
}
void printArray(int arr[], int n)
{
  for (int i = 0; i < n; i++)
    printf("%d ", arr[i]);
  printf("\n");
}

int main()
{
  int arr[max],m,i;
  clock_t start,end;
  printf("Enter the number of elements: ");
```

```
 scanf("%d",&m);
 for(i=0;i<m;i++)
 {
    arr[i]=rand();
 }
 start=clock();
 heapSort(arr, m);
 end=clock();

 printf("\n\nSorted array is given in the following way \n");
 printArray(arr, m);
 printf("\nExecution time: %f",(double)(end-start)/CLOCKS_PER_SEC);

}
```

Output:



n	time
100	0.249
200	0.484
300	0.734
400	1.015
500	1.218
600	1.483
700	1.718
800	1.968
900	2.218
1000	2.484

Experiment 11

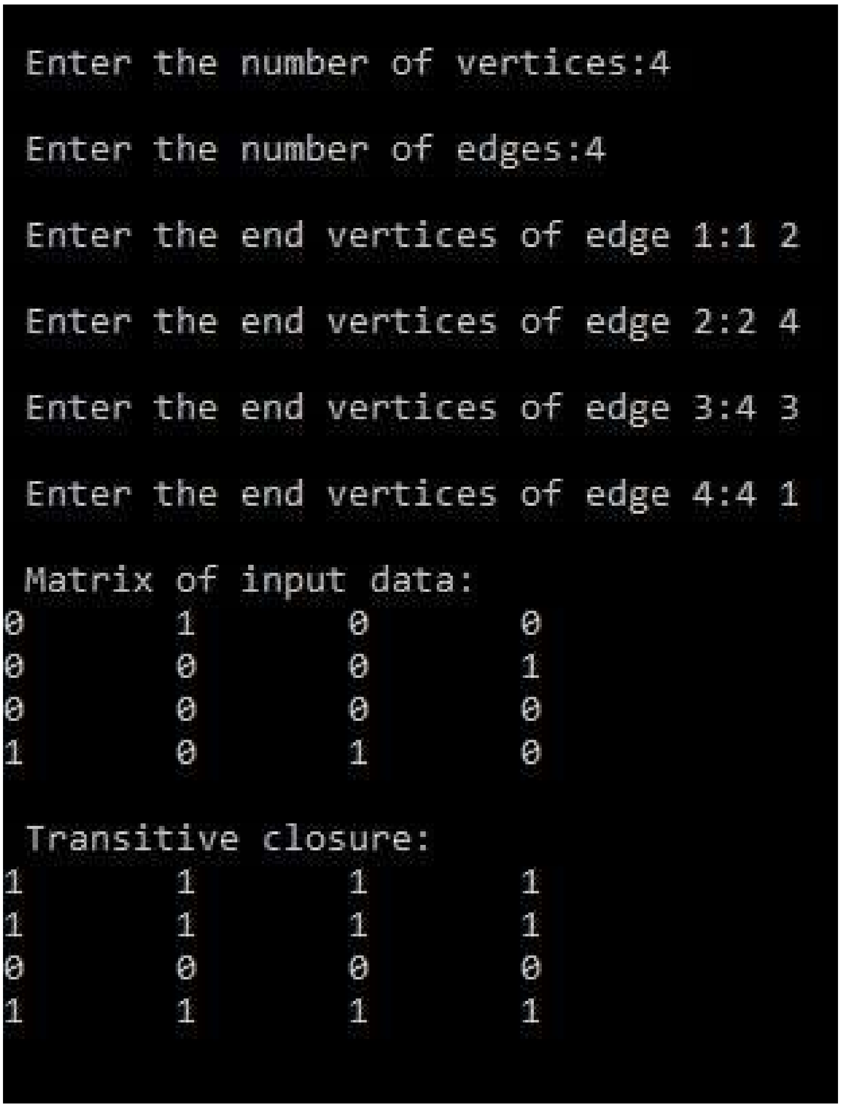
Implement *Warshall's algorithm* using dynamic programming

Program:

```
#include<stdio.h>

#include<conio.h>

#include<math.h>


int max(int,int);


void warshal(int p[10][10],int n) {

        int i,j,k;

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

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

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

            p[i][j]=max(p[i][j],p[i][k]&&p[k][j]);

}


int max(int a,int b) {

        if(a>b)

         return(a); else

         return(b);

}


void main() {

        int p[10][10]= {0},n,e,u,v,i,j;

        printf("\n Enter the number of vertices:");

        scanf("%d",&n);

        printf("\n Enter the number of edges:");
```

```
    scanf("%d",&e);
    for (i=1;i<=e;i++) {
        printf("\n Enter the end vertices of edge %d:",i);
        scanf("%d%d",&u,&v);
        p[u][v]=1;
    }
    printf("\n Matrix of input data: \n");
    for (i=1;i<=n;i++) {
        for (j=1;j<=n;j++)
          printf("%d\t",p[i][j]);
        printf("\n");
    }

    warshal(p,n);
    printf("\n Transitive closure: \n");
    for (i=1;i<=n;i++) {
        for (j=1;j<=n;j++)
          printf("%d\t",p[i][j]);
        printf("\n");
    }
    getch();
}
```

Output:



```
 Enter the number of vertices:4

 Enter the number of edges:4

 Enter the end vertices of edge 1:1 2

 Enter the end vertices of edge 2:2 4

 Enter the end vertices of edge 3:4 3

 Enter the end vertices of edge 4:4 1

 Matrix of input data:
0       1       0       0
0       0       0       1
0       0       0       0
1       0       1       0

 Transitive closure:
1       1       1       1
1       1       1       1
0       0       0       0
1       1       1       1
```

Experiment 12

Implement 0/1 *Knapsack problem* using dynamic programming.

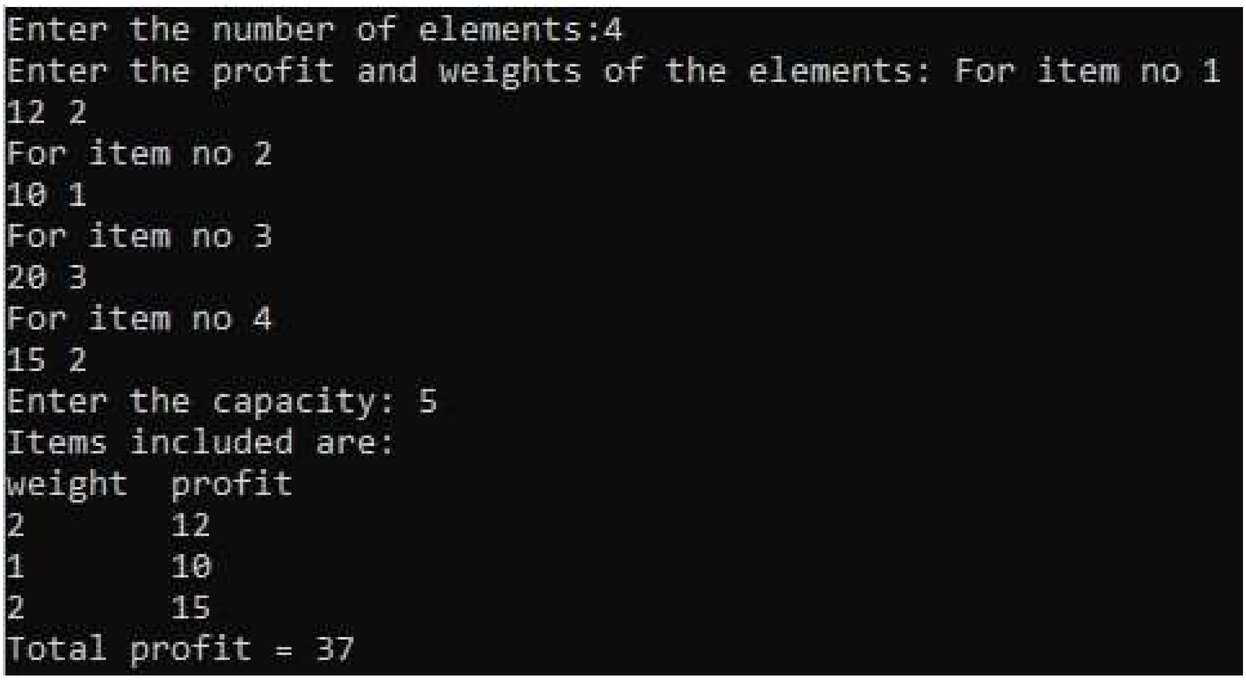
Program:

```
#include<stdio.h>

#include<conio.h>


int w[10],p[10],v[10][10],n,i,j,cap,x[10]= {0};


int max(int i,int j) {

    return ((i>j)?i:j);

}


int knap(int i,int j) {

    int value;

    if(v[i][j]<0) {

        if(j<w[i])

          value=knap(i-1,j); else

          value=max(knap(i-1,j),p[i]+knap(i-1,j-w[i]));

        v[i][j]=value;

    }

    return(v[i][j]);

}


void main() {

    int profit,count=0;

    printf("\nEnter the number of elements\n");

    scanf("%d",&n);
```

```
printf("Enter the profit and weights of the elements\n");
for (i=1;i<=n;i++) {
        printf("For item no %d\n",i);
        scanf("%d%d",&p[i],&w[i]);
}
printf("\nEnter the capacity \n");
scanf("%d",&cap);
for (i=0;i<=n;i++)
  for (j=0;j<=cap;j++)
   if((i==0)||(j==0))
    v[i][j]=0; else
    v[i][j]=-1;
profit=knap(n,cap);
i=n;
j=cap;
while(j!=0&&i!=0) {
        if(v[i][j]!=v[i-1][j]) {
                x[i]=1;
                j=j-w[i];
                i--;
        } else
           i--;
}
printf("Items included are\n");
printf("Sl.no\tweight\tprofit\n");
for (i=1;i<=n;i++)
  if(x[i])
   printf("%d\t%d\t%d\n",++count,w[i],p[i]);
```

```
        printf("Count:%d",count);

        printf("\nTotal profit = %d\n",profit);

        getch();

}
```

Output:



```
Enter the number of elements:4
Enter the profit and weights of the elements: For item no 1
12 2
For item no 2
10 1
For item no 3
20 3
For item no 4
15 2
Enter the capacity: 5
Items included are:
weight   profit
2        12
1        10
2        15
Total profit = 37
```

Experiment 13

Implement All Pair Shortest paths problem using *Floyd’s algorithm*.

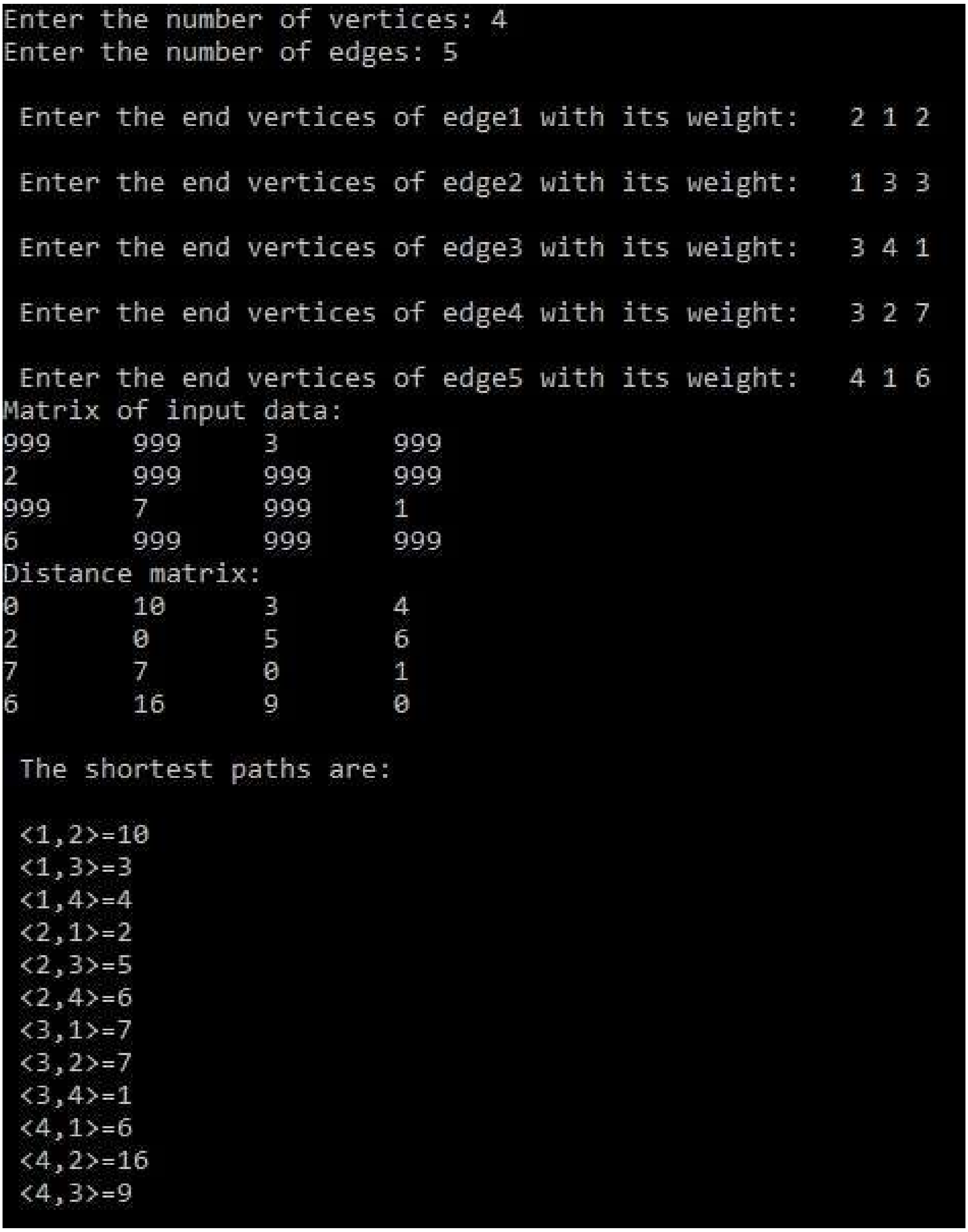
Program:

```
#include<stdio.h>

#include<conio.h>

int min(int,int);


void floyds(int p[10][10],int n) {

            int i,j,k;

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

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

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

                if(i==j)

                 p[i][j]=0; else

                 p[i][j]=min(p[i][j],p[i][k]+p[k][j]);

}


int min(int a,int b) {

            if(a<b)

              return(a); else

              return(b);

}


void main() {

            int p[10][10],w,n,e,u,v,i,j;

            printf("\n Enter the number of vertices:");

            scanf("%d",&n);
```

```
printf("\n Enter the number of edges:\n");
scanf("%d",&e);
for (i=1;i<=n;i++) {
        for (j=1;j<=n;j++)
          p[i][j]=999;
}
for (i=1;i<=e;i++) {
        printf("\n Enter the end vertices of edge%d with its weight \n",i);
        scanf("%d%d%d",&u,&v,&w);
        p[u][v]=w;
}
printf("\n Matrix of input data:\n");
for (i=1;i<=n;i++) {
        for (j=1;j<=n;j++)
          printf("%d \t",p[i][j]);
        printf("\n");
}
floyds(p,n);
printf("\n Distance matrix:\n");
for (i=1;i<=n;i++) {
        for (j=1;j<=n;j++)
          printf("%d \t",p[i][j]);
        printf("\n");
}
printf("\n The shortest paths are:\n");
for (i=1;i<=n;i++)
  for (j=1;j<=n;j++) {
        if(i!=j)
```

```
                printf("\n <%d,%d>=%d",i,j,p[i][j]);
        }
        getch();
}
```

Output:



```
Enter the number of vertices: 4
Enter the number of edges: 5

 Enter the end vertices of edge1 with its weight:   2 1 2

 Enter the end vertices of edge2 with its weight:   1 3 3

 Enter the end vertices of edge3 with its weight:   3 4 1

 Enter the end vertices of edge4 with its weight:   3 2 7

 Enter the end vertices of edge5 with its weight:   4 1 6
Matrix of input data:
999     999     3       999
2       999     999     999
999     7       999     1
6       999     999     999
Distance matrix:
0       10      3       4
2       0       5       6
7       7       0       1
6       16      9       0

 The shortest paths are:

 <1,2>=10
 <1,3>=3
 <1,4>=4
 <2,1>=2
 <2,3>=5
 <2,4>=6
 <3,1>=7
 <3,2>=7
 <3,4>=1
 <4,1>=6
 <4,2>=16
 <4,3>=9
```

Experiment 14

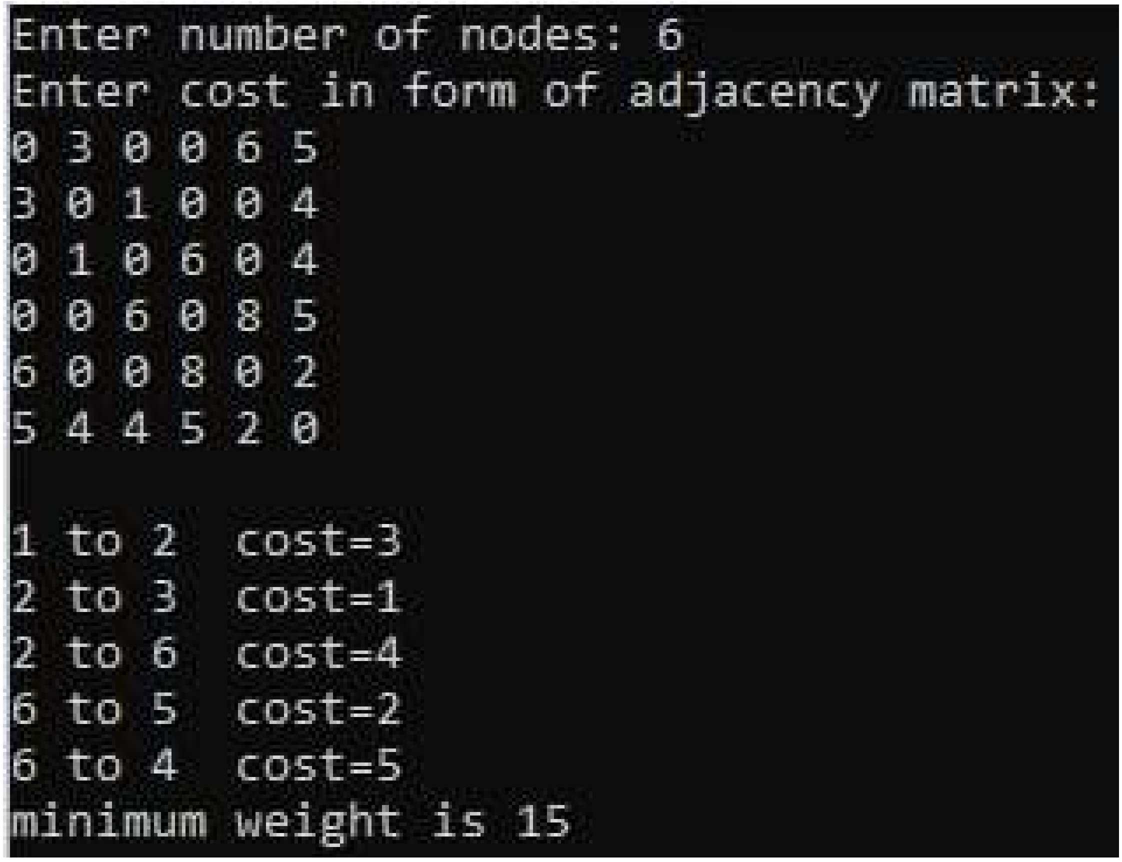
Find Minimum Cost Spanning Tree of a given undirected graph using *Prim's algorithm*.

Program:

```
#include<stdio.h>

int main()
{
  int cost[10][10],visited[10]={0},i,j,n,no_e=1,min,a,b,min_cost=0;
  printf("Enter number of nodes ");
  scanf("%d",&n);
  printf("Enter cost in form of adjacency matrix\n");
  for(i=1;i<=n;i++)
  {
    for(j=1;j<=n;j++)
    {
      scanf("%d",&cost[i][j]);
      if(cost[i][j]==0)
       cost[i][j]=1000;
    }
  }

 visited[1]=1; // visited first node
  while(no_e<n)
  {
    min=1000;
    for(i=1;i<=n;i++)
```

```
        {
            for(j=1;j<=n;j++)
            {
                if(cost[i][j]<min)
                {
                    if(visited[i]!=0)
                    {
                        min=cost[i][j];
                        a=i;
                        b=j;
                    }
                }
            }
        }
        if(visited[b]==0)
        {
            printf("\n%d to %d  cost=%d",a,b,min);
            min_cost=min_cost+min;
            no_e++;
        }
        visited[b]=1;
        cost[a][b]=cost[b][a]=1000;
    }
    printf("\nminimum weight is %d",min_cost);
    return 0;
}
```

Output:



```
Enter number of nodes: 6
Enter cost in form of adjacency matrix:
0 3 0 0 6 5
3 0 1 0 0 4
0 1 0 6 0 4
0 0 6 0 8 5
6 0 0 8 0 2
5 4 4 5 2 0

1 to 2  cost=3
2 to 3  cost=1
2 to 6  cost=4
6 to 5  cost=2
6 to 4  cost=5
minimum weight is 15
```

Experiment 15

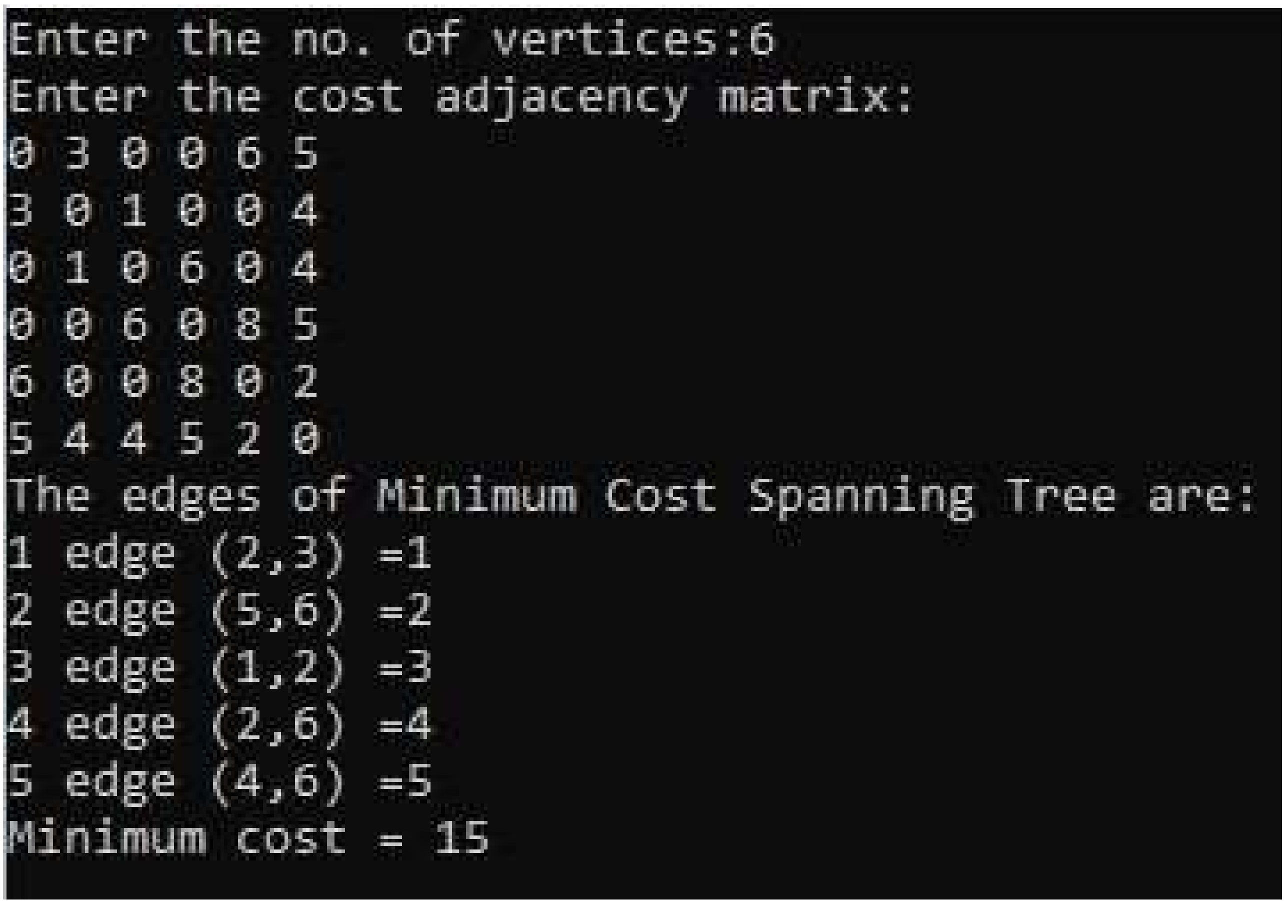
Find Minimum Cost Spanning Tree of a given undirected graph using *Kruskal’s algorithm*.

Program:

```
#include <stdio.h>

#include <conio.h>

#include <stdlib.h>


int i, j, k, a, b, u, v, n, ne = 1;

int min, mincost = 0, cost[9][9], parent[9];


int find(int);

int uni(int, int);


void main()

{

  printf("Enter the no. of vertices:\n");

  scanf("%d", &n);

  printf("\nEnter the cost adjacency matrix:\n");

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

  {

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

    {

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

      if (cost[i][j] == 0)

        cost[i][j] = 999;

    }
}
printf("The edges of Minimum Cost Spanning Tree are\n");
while (ne < n)
{
 for (i = 1, min = 999; i <= n; i++)
 {
  for (j = 1; j <= n; j++)
  {
   if (cost[i][j] < min)
   {
    min = cost[i][j];
    a = u = i;
    b = v = j;
   }
  }
 }

 u = find(u);
 v = find(v);

 if (uni(u, v))
 {
  printf("%d edge (%d,%d) =%d\n", ne++, a, b, min);
  mincost += min;
 }

 cost[a][b] = cost[b][a] = 999;
}
```

```
  printf("\nMinimum cost = %d\n", mincost);
  getch();
}

int find(int i)
{
  while (parent[i])
    i = parent[i];
  return i;
}

int uni(int i, int j)
{
  if (i != j)
  {
    parent[j] = i;
    return 1;
  }

  return 0;
}
```

Output:



```
Enter the no. of vertices:6
Enter the cost adjacency matrix:
0 3 0 0 6 5
3 0 1 0 0 4
0 1 0 6 0 4
0 0 6 0 8 5
6 0 0 8 0 2
5 4 4 5 2 0
The edges of Minimum Cost Spanning Tree are:
1 edge (2,3) =1
2 edge (5,6) =2
3 edge (1,2) =3
4 edge (2,6) =4
5 edge (4,6) =5
Minimum cost = 15
```

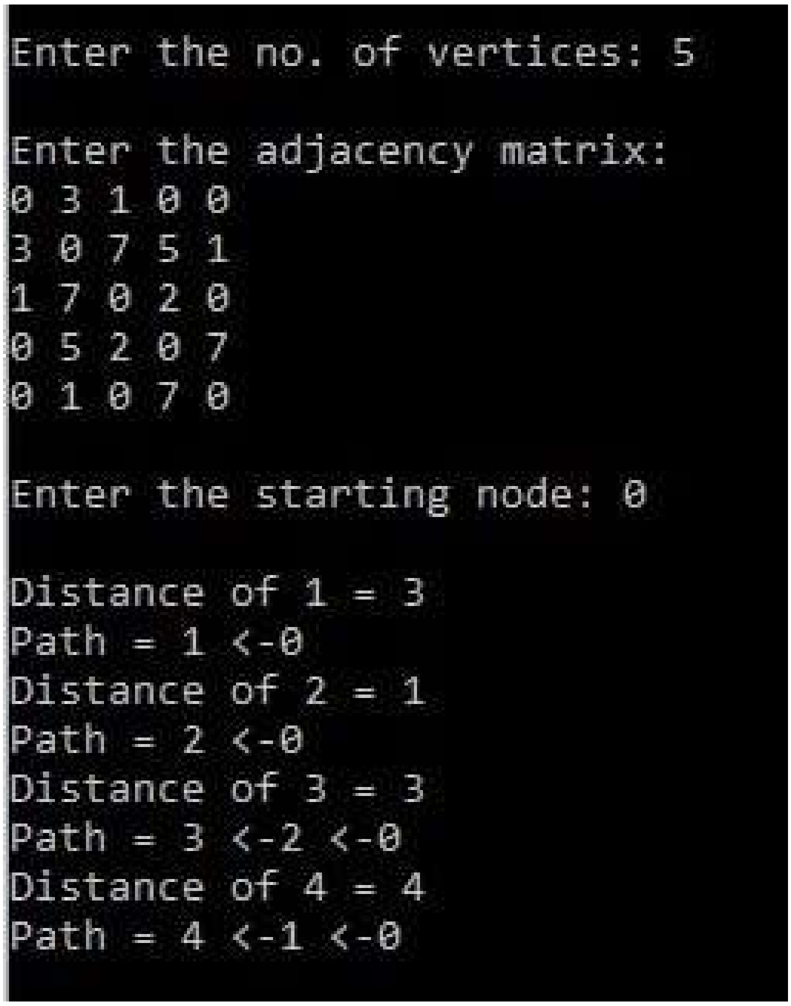
Experiment 16

From a given vertex in a weighted connected graph, find shortest paths to other vertices using *Dijkstra's algorithm*

Program:

```
#include<stdio.h>

#include<conio.h>

#define INFINITY 9999

#define MAX 10

void dijkstra(int G[MAX][MAX], int n, int startnode);


void main(){

        int G[MAX][MAX], i, j, n, u;

        printf("\nEnter the no. of vertices: ");

        scanf("%d", &n);

        printf("\nEnter the adjacency matrix:\n");

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

                for(j=0;j < n;j++)

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

        printf("\nEnter the starting node: ");

        scanf("%d", &u);

        dijkstra(G,n,u);

        getch();

}


void dijkstra(int G[MAX][MAX], int n, int startnode)

{

        int cost[MAX][MAX], distance[MAX], pred[MAX];
int visited[MAX], count, mindistance, nextnode, i,j;
for(i=0;i < n;i++)
        for(j=0;j < n;j++)
                if(G[i][j]==0)
                        cost[i][j]=INFINITY;
                else
                        cost[i][j]=G[i][j];

for(i=0;i< n;i++)
{
        distance[i]=cost[startnode][i];
        pred[i]=startnode;
        visited[i]=0;
}
distance[startnode]=0;
visited[startnode]=1;
count=1;
while(count < n-1){
        mindistance=INFINITY;
        for(i=0;i < n;i++)
                if(distance[i] < mindistance&&!visited[i])
                {
                        mindistance=distance[i];
                        nextnode=i;
                }
        visited[nextnode]=1;
        for(i=0;i < n;i++)
                if(!visited[i])
                if(mindistance+cost[nextnode][i] < distance[i])
                {
                    distance[i]=mindistance+cost[nextnode][i];
                    pred[i]=nextnode;
                }
            count++;
    }
    for(i=0;i < n;i++)
        if(i!=startnode)
        {
            printf("\n\nDistance of %d = %d", i, distance[i]);
            printf("\nPath = %d ", i);
            j=i;
            int countn=1;
            do
            {
                j=pred[j];
                printf("<-%d", j);
                countn++;
            }
            while(j!=startnode);
    printf("\nCount=%d",countn);
        }
}
```

Output:



```
Enter the no. of vertices: 5

Enter the adjacency matrix:
0 3 1 0 0
3 0 7 5 1
1 7 0 2 0
0 5 2 0 7
0 1 0 7 0

Enter the starting node: 0

Distance of 1 = 3
Path = 1 <-0
Distance of 2 = 1
Path = 2 <-0
Distance of 3 = 3
Path = 3 <-2 <-0
Distance of 4 = 4
Path = 4 <-1 <-0
```

Experiment 17

Implement *"Sum of Subsets"* using Backtracking. Problem:
Find a subset of a given set S = {s1,s2,……,sn} of n positive integers whose sum is equal to a given positive integer d. For example, if S = {1,2,5,6,8} and d = 9 there are two solutions {1,2,6} and {1,8}. A suitable message is to be displayed if the given problem instance doesn't have a solution.

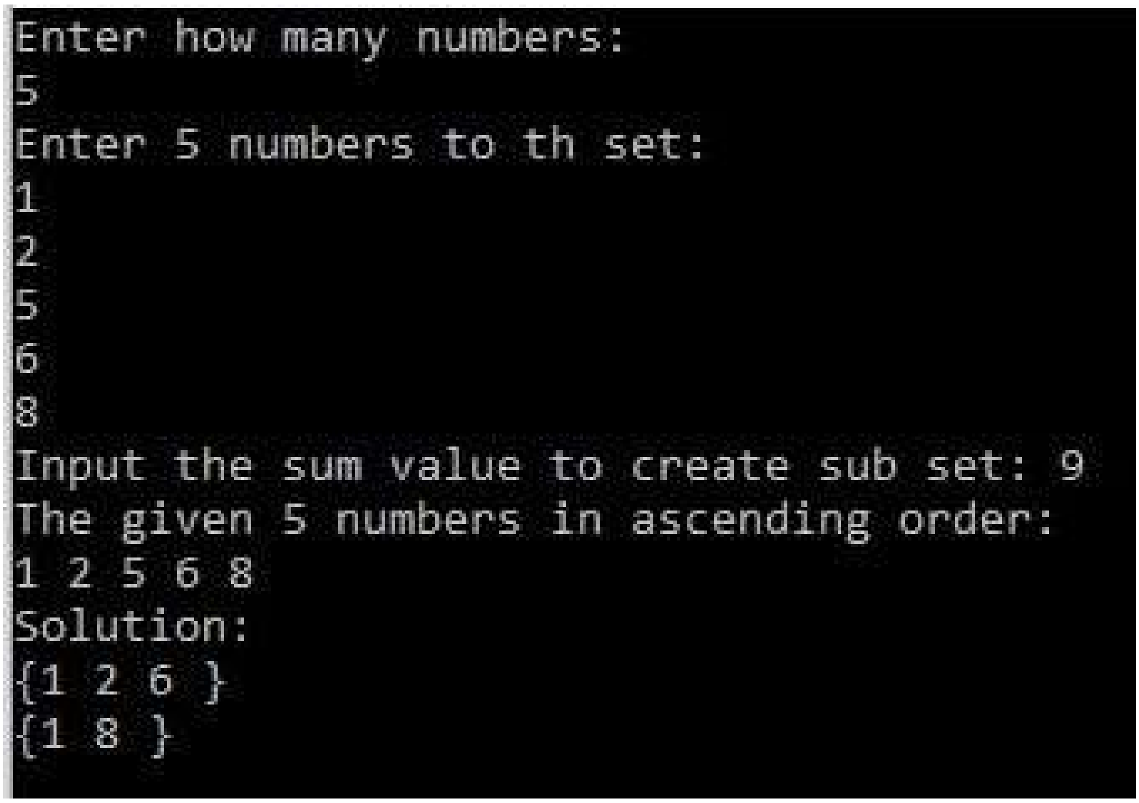
Program:

```
#include<stdio.h>

#include<conio.h>

#define TRUE 1

#define FALSE 0

int inc[50],w[50],sum,n;

int promising(int i,int wt,int total) {

        return(((wt+total)>=sum)&&((wt==sum)||(wt+w[i+1]<=sum)));

}


void main() {

        int i,j,n,temp,total=0;

        printf("Enter how many numbers:\n");

        scanf("%d",&n);

        printf("Enter %d numbers to th set:\n",n);

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

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

                total+=w[i];

        }

        printf("Input the sum value to create sub set: ");

        scanf("%d",&sum);
```

```
    for (i=0;i<=n;i++)
      for (j=0;j<n-1;j++)
       if(w[j]>w[j+1]) {
        temp=w[j];
        w[j]=w[j+1];
        w[j+1]=temp;
    }
    printf("The given %d numbers in ascending order:\n",n);
    for (i=0;i<n;i++)
      printf("%d ",w[i]);
    if((total<sum))
      printf("\n Subset construction is not possible"); else {
        for (i=0;i<n;i++)
           inc[i]=0;
        printf("\nSolution:\n");
        sumset(-1,0,total);
    }
    getch();
}
void sumset(int i,int wt,int total) {
    int j;
    if(promising(i,wt,total)) {
        if(wt==sum) {
            printf("{");
            for (j=0;j<=i;j++)
                if(inc[j])
                 printf("%d ",w[j]);
            printf("}\n");
```

```
                } else {
                        inc[i+1]=TRUE;
                        sumset(i+1,wt+w[i+1],total-w[i+1]);
                        inc[i+1]=FALSE;
                        sumset(i+1,wt,total-w[i+1]);
                }
        }
}
```

Output:



```
Enter how many numbers:
5
Enter 5 numbers to th set:
1
2
5
6
8
Input the sum value to create sub set: 9
The given 5 numbers in ascending order:
1 2 5 6 8
Solution:
{1 2 6 }
{1 8 }
```

Experiment 18

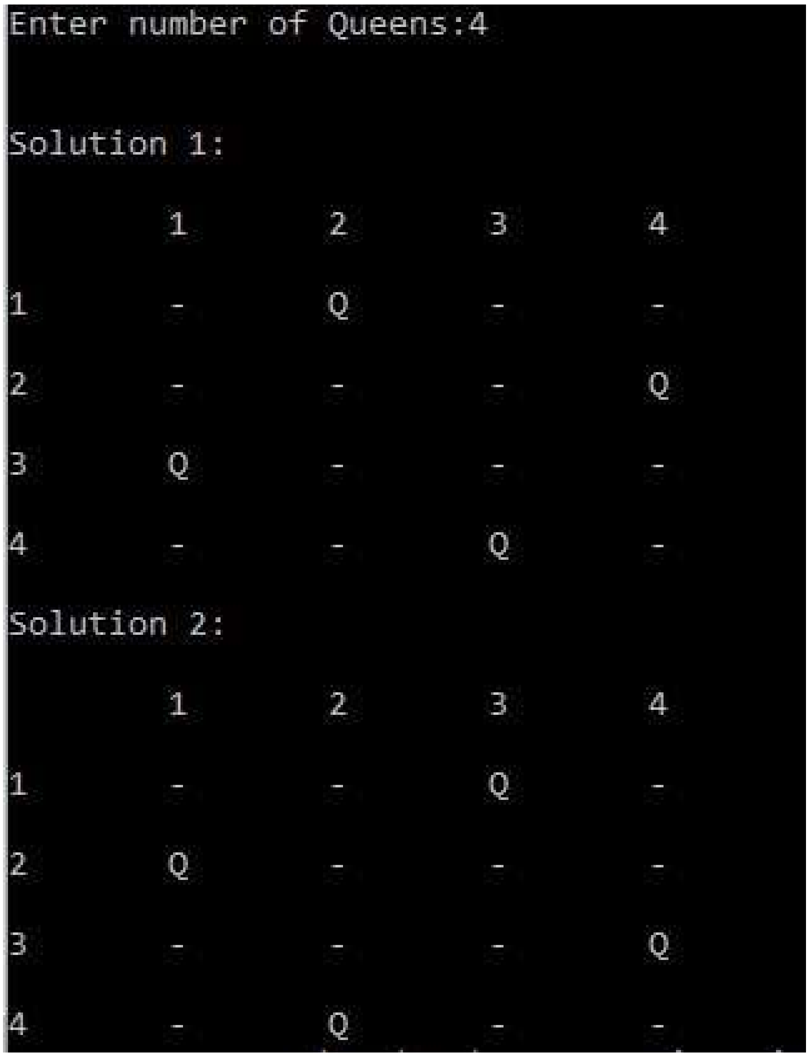
Implement *"N-Queens Problem"* using Backtracking.

Program:

```
#include<stdio.h>
#include<math.h>
int board[20],count;

int main()
{  int n,i,j;
void queen(int row,int n);
printf("Enter number of Queens:");
scanf("%d",&n);
queen(1,n);
return 0;
}

void print(int n)
{
int i,j;
printf("\n\nSolution %d:\n\n",++count);
for(i=1;i<=n;++i)
  printf("\t%d",i);
for(i=1;i<=n;++i)
{
  printf("\n\n%d",i);
  for(j=1;j<=n;++j)
  {
   if(board[i]==j)
    printf("\tQ");
   else
    printf("\t-");
  }
}
}

int place(int row,int column)
{
int i;
for(i=1;i<=row-1;++i)
{
  if(board[i]==column)
   return 0;
  else
   if(abs(board[i]-column)==abs(i-row))
    return 0;
}
return 1;
}

void queen(int row,int n)
{
int column;
for(column=1;column<=n;++column)
{
  if(place(row,column))
```

```
 {
  board[row]=column;
  if(row==n)
   print(n);
  else
   queen(row+1,n);
 }
}
}
```

Output:



```
Enter number of Queens:4

Solution 1:

        1       2       3       4

1       -       Q       -       -

2       -       -       -       Q

3       Q       -       -       -

4       -       -       Q       -
Solution 2:

        1       2       3       4

1       -       -       Q       -

2       Q       -       -       -

3       -       -       -       Q

4       -       Q       -       -
```