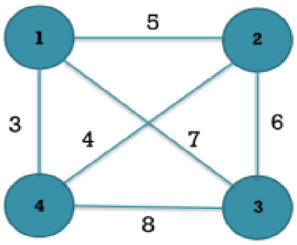
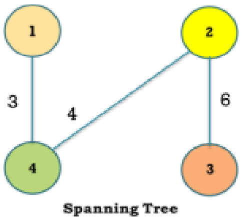
Program:1

```
/* Program to implement Kruskal’s Algorithm */

#include<stdio.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("\n\t Implementation of Kruskal's algorithm\n");
 printf("\nEnter the no. of vertices:");
 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("\n\tMinimum cost = %d\n",mincost);
}
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;
}
```

Input Graph:





```
Output 1:
Implementation of Kruskal's algorithm
Enter the no. of vertices: 4
Enter the cost adjacency matrix:
999 5 7 3
5 999 6 4
7 6 999 8
3 4 6 999
The edges of Minimum Cost Spanning Tree are
1 edge (1,4) =3
2 edge (2,4) =4
3 edge (2,3) =6
Minimum cost = 13
```

Program: 2 //*prims Algorithm*//

```
#include<stdio.h>
// #include<conio.h>
#define INF 999

int prim(int c[10][10],int n,int s)
{
        int v[10],i,j,sum=0,ver[10],d[10],min,u;
        for(i=1;i<=n;i++)
        {
                ver[i]=s;
                d[i]=c[s][i];
                v[i]=0;
        }
        v[s]=1;
        for(i=1;i<=n-1;i++)
        {
                min=INF;
                for(j=1;j<=n;j++)
                if(v[j]==0 && d[j]<min)
                {
                        min=d[j];
                        u=j;
                }
                v[u]=1;
                sum=sum+d[u];
                printf("\n%d -> %d sum=%d",ver[u],u,sum);
                for(j=1;j<=n;j++)
                if(v[j]==0 && c[u][j]<d[j])
                {
                        d[j]=c[u][j];
                        ver[j]=u;
                }
         }
        return sum;
}

void main()
{
        int c[10][10],i,j,res,s,n;
        clrscr();

        printf("\nEnter n value:");
        scanf("%d",&n);

        printf("\nEnter the graph data:\n");
        for(i=1;i<=n;i++)
        for(j=1;j<=n;j++)
        scanf("%d",&c[i][j]);

        printf("\nEnter the souce node:");
        scanf("%d",&s);

        res=prim(c,n,s);
        printf("\nCost=%d",res);
        getch();
}
```

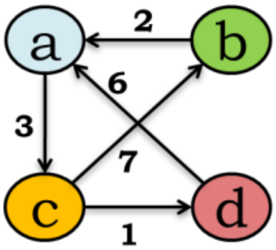
Input/output:

```
Enter n value:3

Enter the graph data:
0    10   1
10   0    6
1     6    0

Enter the source node:1
1 -> 3   sum=1
3 -> 2    sum=7

Cost=7
```

Program 3.a: Design and implement C/C++ Program to solve All-Pairs Shortest Paths problem using Floyd's algorithm.

```
#include<stdio.h>
void readf();
void amin();
int cost[20][20],a[20][20];
int i,j,k,n;

void readf()
 {
 printf("\n Enter the number of vertices :");
 scanf("%d",&n);
 printf("\n Enter the weighted matrix - 999 for infinity:");
    for(i=0;i<n;i++)
   {
    for(j=0;j<n;j++)
   {
        scanf("%d",&cost[i][j]);
        if(cost[i][j]==0 && (i!=j))
         cost[i][j]=999;
         a[i][j]=cost[i][j];
    }
   }
 }
 void amin()
 {
   for(k=0;k<n;k++)
   {
      for(i=0;i<n;i++)
     {
        for(j=0;j<n;j++)
        {
          if(a[i][j]>a[i][k]+a[k][j])
          {
            a[i][j]=a[i][k]+a[k][j];
          }
        }
     }
   }
   printf("\n The All pair shortest path is:");
   for(i=0;i<n;i++)
   {
   printf("\n");
    for(j=0;j<n;j++)
    {
     printf("%d\t",a[i][j]);
    }
  }
}

 void main()
 {
 readf();
 amin();
 }
```

Input:

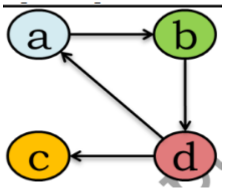


```
OUTPUT:
Enter the number of vertices:
4

Enter the weighted matrix - 999 for infinity :
 0    999   3     999
 2      0   999   999
999   7     0      1
 6     999  999   0

The All pair shortest path is:

0   10   3   4
2    0    5   6
7    7    0   1
6    16   9  0
```

Program 3.b: Design and implement C/C++ Program to find the transitive closure using Warshal's algorithm.

```
#include<stdio.h>
#include<math.h>
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] = p[i][j] || (p[i][k] && p[k][j]);
}
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);

 printf("Enter the edges: (u,v)\n");
              for (i = 1; i <= e; 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");
             }
 }
```

Input :



Output:

```
Enter the number of vertices: 4

Enter the number of edges: 4

Enter the edges: (u,v)
1   2
2   4
4   1
4   3
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
```

Program:

```
#include<stdio.h>
#define INF 999
void dijkstra(int c[10][10],int n,int s,int d[10])
{
        int v[10],min,u,i,j;
        for(i=1;i<=n;i++)
        {
                d[i]=c[s][i];
                v[i]=0;
        }
        v[s]=1;
        for(i=1;i<=n;i++)
        {
                min=INF;
                for(j=1;j<=n;j++)
                if(v[j]==0 && d[j]<min)
                {
                        min=d[j];
                        u=j;
                }
                v[u]=1;
                for(j=1;j<=n;j++)
                if(v[j]==0 && (d[u]+c[u][j])<d[j])
                d[j]=d[u]+c[u][j];
        }
}

int main()
{
        int c[10][10],d[10],i,j,s,sum,n;
        printf("\nEnter n value:");
        scanf("%d",&n);

        printf("\nEnter the graph data:\n");
        for(i=1;i<=n;i++)
        for(j=1;j<=n;j++)
        scanf("%d",&c[i][j]);

        printf("\nEnter the souce node:");
        scanf("%d",&s);

        dijkstra(c,n,s,d);

        for(i=1;i<=n;i++)
        printf("\nShortest distance from %d to %d is %d",s,i,d[i]);
        return 0;
}
```

Input/Output

```
Enter n value:6

Enter the graph data:
0 15 10 999 45 999
999 0 15 999 20 999
20 999 0 20 999 999
999 10 999 0 35 999
999 999 999 30 0 999
999 999 999 4 999 0

Enter the souce node:2

Shortest distance from 2 to 1 is 35
Shortest distance from 2 to 2 is 0
Shortest distance from 2 to 3 is 15
Shortest distance from 2 to 4 is 35
Shortest distance from 2 to 5 is 20
Shortest distance from 2 to 6 is 999
```

Program 5 To obtain the Topological ordering of vertices in a given digraph

```
#include<stdio.h>
int temp[10],k=0;
void sort(int a[][10],int id[],int n)
{
int i,j;
   for(i=1;i<=n;i++)
   {
     if(id[i]==0)
     {
       id[i]=-1;
       temp[++k]=i;
          for(j=1;j<=n;j++)
          {
            if(a[i][j]==1 && id[j]!=-1)
               id[j]--;
            }
         i=0;
      }
     }
    }

void main()
{
int a[10][10],id[10],n,i,j;
printf("\nEnter the n value:");
scanf("%d",&n);
for(i=1;i<=n;i++)
id[i]=0;

printf("\nEnter the graph data:\n");
for(i=1;i<=n;i++)
for(j=1;j<=n;j++)
{
scanf("%d",&a[i][j]);
if(a[i][j]==1)
id[j]++;
}
sort(a,id,n);

if(k!=n)
printf("\nTopological ordering not possible");
else
{
printf("\nTopological ordering is:");
for(i=1;i<=k;i++)
printf("%d ",temp[i]);
}
}
```

Input/output:

```
Enter the n value:6
Enter the graph data:
0 0 1 1 0 0
0 0 0 1 1 0
0 0 0 1 0 1
0 0 0 0 0 1
0 0 0 0 0 1
0 0 0 0 0 0
Topological ordering is:1 2 3 4 5 6
```

Program 6 Design and implement C/C++ Program to solve 0/1 Knapsack problem using Dynamic Programming method.

```
#include<stdio.h>
int w[10],p[10],n;

int max(int a,int b)
{
        return a>b?a:b;
}
int knap(int i,int m)
{
        if(i==n) return w[i]>m?0:p[i];

        if(w[i]>m) return knap(i+1,m);

        return max(knap(i+1,m),knap(i+1,m-w[i])+p[i]);
}
int main()
{
        int m,i,max_profit;
        printf("\nEnter the no. of objects:");
        scanf("%d",&n);

        printf("\nEnter the knapsack capacity:");
        scanf("%d",&m);

        printf("\nEnter profit followed by weight:\n");
        for(i=1;i<=n;i++)
        scanf("%d %d",&p[i],&w[i]);

        max_profit=knap(1,m);
        printf("\nMax profit=%d",max_profit);
        return 0;
}
```

Input/Output:

```
Enter the no. of objects:4

Enter the knapsack capacity:6

Enter profit followed by weight:
78      2
45      3
92      4
71      5

Max profit=170
```

Program 7 Design and implement C/C++ Program to solve discrete Knapsack and continuous Knapsack problems using greedy approximation method.

```
#include <stdio.h>
void main()
{
 int cur_w,n;
 float tot_v;
 int p[10],w[10],W;
 int i, maxi;
 int used[10];

 printf("Enter the no. of items:\n");
 scanf("%d",&n);

 printf("Enter the weight and price of all items:\n");
 for(i=0;i<n;i++)
 {
 scanf("%d%d",&w[i],&p[i]);
 }

 printf("Enter the capacity of knapsack:\n");
 scanf("%d",&W);
 for (i = 0; i < n; ++i)
 used[i] = 0;
 cur_w = W;

 while (cur_w > 0)
 {
 maxi = -1;
 for (i = 0; i < n; ++i)

 if ((used[i] == 0) &&
 ((maxi == -1) || ((float)w[i]/p[i] > (float)w[maxi]/p[maxi])))
 maxi = i;
 used[maxi] = 1;
 cur_w -= p[maxi];
 tot_v += w[maxi];

 if (cur_w >= 0)
 printf("Added object %d (%d, %d) completely in the bag. Space left: %d.\n", maxi + 1, w[maxi],
p[maxi], cur_w);
 else
{
 printf("Added %d%% (%d, %d) of object %d in the bag.\n", (int)((1 + (float)cur_w/p[maxi]) * 100),
w[maxi], p[maxi], maxi + 1);

 tot_v -= w[maxi];

 tot_v += (1 + (float)cur_w/p[maxi]) * w[maxi];
 }
 }
 printf("Filled the bag with objects worth %.2f.\n", tot_v);
}
```

Output:

```
Enter the no. of items:
5

Enter the weight and price of all items:
10       3
15      3
10      2
12      5
8       1
Enter the capacity of knapsack:
10
Added object 5 (8, 1) completely in the bag. Space left: 9.
Added object 2 (15, 3) completely in the bag. Space left: 6.
Added object 3 (10, 2) completely in the bag. Space left: 4.
Added object 1 (10, 3) completely in the bag. Space left: 1.
Added 19% (12, 5) of object 4 in the bag.

Filled the bag with objects worth 45.40.
```

Program 8 Design and implement C/C++ Program to find a subset of a given set S = {sl , s2,.....,sn} of n positive integers whose sum is equal to a given positive integer d.

```
 #include<stdio.h>
#define MAX 10
int s[MAX],x[MAX],d;
void sumofsub(int p,int k,int r)
{
        int i;
        x[k]=1;
        if((p+s[k])==d)
        {
                for(i=1;i<=k;i++)
                if(x[i]==1)
                printf("%d ",s[i]);
                printf("\n");
        }
        else
        if(p+s[k]+s[k+1]<=d)
        sumofsub(p+s[k],k+1,r-s[k]);

        if((p+r-s[k]>=d) && (p+s[k+1]<=d))
        {
                x[k]=0;
                sumofsub(p,k+1,r-s[k]);
        }
}
int  main()
{
        int i,n,sum=0;
        printf("\nEnter the n value:");
        scanf("%d",&n);

        printf("\nEnter the set in increasing order:");
        for(i=1;i<=n;i++)
        scanf("%d",&s[i]);

        printf("\nEnter the max subset value:");
        scanf("%d",&d);
        for(i=1;i<=n;i++)
        sum=sum+s[i];

        if(sum<d || s[1]>d)
        printf("\nNo subset possible");
        else
        sumofsub(0,1,sum);
        return 0;
}
```

Input/output:

```
Enter the n value:9
Enter the set in increasing order:1 2 3 4 5 6 7 8 9
Enter the max subset value:9
1  2  6
1  3  5
1  8
2  3  4
2  7
3  6
4  5
9
```

Program 9: Design and implement C/C++ Program to sort a given set of n integer elements using Selection Sort method and compute its time complexity. Run the program for varied values of n> 5000 and record the time taken to sort. Plot a graph of the time taken versus n. The elements can be read from a file or can be generated using the random number generator.

```
#include <stdio.h>
#include <stdlib.h>
#include<time.h>
void swap(long int*a,long int*b)
{
 int tmp=*a;
 *a=*b;
 *b=tmp;
}
void selectionsort (long int arr[],long int n)
{
 long int i,j,midx;
 for(i=0;i<n-1;i++)
 {
 midx=i;
 for(j=i+1;j<n;j++)
 if(arr[j]<arr[midx])
 midx=j;
 swap(&arr[midx],&arr[i]);
 }
}
void main()
{
 long int n=1000;
 int it=0;
 double tim1[10];
 printf("Input Size, Selection Sorting time \n");
      while(it++<5)
     {
            long int a[n];
             for(int i=0;i<n;i++)
            {
             long int no=rand()%n+1;
             a[i]=no;
            }
 //using clock t to store time
 clock_t start,end;
 start=clock();
 selectionsort(a,n);
 end=clock();
 tim1[it]=(double)(end-start)/1000;
 printf(" %ld = %ld ms\n",n,(long int)tim1[it]) n+=1000;
      }
 }
```

Output:

```
Input Size, Selection Sorting time
 1000 = 1 ms
 2000 = 5 ms
 3000 = 13 ms
 4000 = 23 ms
 5000 = 35 ms
```

Program 10: **Design and implement C/C++ Program to sort a given set of n integer elements using Quick Sort method and compute its time complexity. Run the program for varied values of n> 5000 and record the time taken to sort. Plot a graph of the time taken versus n. The elements can be read from a file or can be generated using the random number generator.**

```
#include <stdio.h>
#include <stdlib.h>
#include<time.h>
static int max= 5000;
 static int partition(long int arr[],int low,int high)
 {
 int pivot = arr[low];
 int i = low;
 int j= high+1;
 while(i<=j)
 {
 do
 {
 i++;
 }while(pivot>=arr[i] && i<=high);
 do
 {
 j--;
 }
while(pivot<arr[j])
if(i<j)
 {
 int temp = arr[i];
 arr[i] = arr[j];
 arr[j] = temp;
 }
 }
 int temp = arr[low];
 arr[low] = arr[j];
 arr[j] = temp;
 return j;
 }
 static void qs(long int arr[],int low,int high)
 {
 int mid;
 if(low<high)
 {
 mid = partition(arr, low, high);
 qs(arr,low,mid-1);
 qs(arr,mid+1,high);
 }
 }
 void main()
 {
 int n,i;
 long int a[5000], no;
 double tm;
 //using clock t to store time
 clock_t start,end;
 printf("\n Enter the number of elements:\n");
 scanf("%d",&n);
 for(i=0;i<n;i++)
 {
 no=rand()%n+1;
 a[i]=no;
 }
 start=clock();
 qs(a,0,n-1);
 end=clock();
 tm = (end - start);
 printf(" %d = %lf\n Nano Seconds",n,tm);
}
```

Output:

```
Enter the number of elements:
1000
 1000 = 128.000000
Nano Second
```

Program 11: **Design and implement C/C++ Program to sort a given set of n integer elements using Merge Sort method and compute its time complexity. Run the program for varied values of n> 5000, and record the time taken to sort. Plot a graph of the time taken versus n. The elements can be read from a file or can be generated using the random number generator.**

```
#include<stdio.h>
#include<time.h>
#include <stdlib.h>
#define max 5000
int array[max];
void merge(int low, int mid, int high)
{
int temp[max];
int i = low;
int j = mid +1;
int k = low ;
while((i <= mid) && (j <=high))
{
if(array[i] <= array[j])
temp[k++] = array[i++] ;
else
temp[k++] = array[j++] ;
}
while( i <= mid )
temp[k++]=array[i++];
while( j <= high )
temp[k++]=array[j++];
for(i= low; i <= high ; i++)
array[i]=temp[i];
}
void merge_sort(int low, int high)
{
int mid;
if( low != high )
{
mid = (low+high)/2;
merge_sort(low , mid);
merge_sort(mid+1, high);
merge(low, mid, high);
}
}
void main()
{
int i,n, no;
double tm;
 clock_t start,end;
printf("Enter the number of elements : ");
scanf("%d",&n);
for(i=0;i<n;i++)
{
no=rand()%n+1;
 array[i]=no;
}
printf("Unsorted list is :\n");
for( i = 0 ; i<n ; i++)
printf("%d ", array[i]);
 start=clock();
merge_sort(0, n-1);
printf("\nSorted list is :\n");
for( i = 0 ; i<n ; i++)
printf("%d ", array[i]);
printf("\n");
end=clock();
 tm = (end - start);
 printf(" %d = %lf Nano Seconds \n",n,tm);
printf("\n");
}/*End of main()*
```

Output

```
Enter the number of elements : 20

Unsorted list is :
4 7 18 16 14 16 7 13 10 2 3 8 11 20 4 7 1 7 13 17

Sorted list is :
1 2 3 4 4 7 7 7 7 8 10 11 13 13 14 16 16 17 18 20

 20 = 26.000000 Nano Seconds
```

Program 12:N-Queens program

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#define MAX 50

int can_place(int c[],int r)
{
        int i;
        for(i=0;i<r;i++)
        if(c[i]==c[r] || (abs(c[i]-c[r])==abs(i-r)))
        return 0;
        return 1;
}

void display(int c[],int n)
{
        int i,j;
        char cb[10][10];
        for(i=0;i<n;i++)
        for(j=0;j<n;j++)
        cb[i][j]='-';
        for(i=0;i<n;i++)
        cb[i][c[i]]='Q';
        for(i=0;i<n;i++)
        {
                for(j=0;j<n;j++)
                printf("%c",cb[i][j]);
                printf("\n");
        }
}

void n_queens(int n)
{
        int r;
        int c[MAX];
        c[0]=-1;
        r=0;
        while(r>=0)
        {       c[r]++;
                while(c[r]<n && !can_place(c,r))
                c[r]++;
                if(c[r]<n)
                {       if(r==n-1)
                        {       display(c,n);
                                printf("\n\n");
                        }
                        else
                        {       r++;
                                c[r]=-1;
                        }
                }
                else
                r--;
        }
}
```

```
void main()
{
        int n;
        clrscr();
        printf("\nEnter the no. of queens:");
        scanf("%d",&n);
        n_queens(n);
        getch();
}
```

Input/Output:

```
Enter the no. of queens:4
- Q - -
- - - Q
Q - - -
- - Q -

- - Q -
Q - - -
- - - Q
- Q - -
```