1. Design and implement C/C++ Program to find Minimum Cost Spanning Tree of a given connected undirected graph using Kruskal's algorithm.

```
#include<stdio.h>
    void main()
    {
            int mincost=0,n,i,j,ne,a = 0,b = 0,min,u = 0,v = 0;
            int cost[10][10],parent[10];
            printf("Enter the number of vertices\n");
            scanf("%d",&n);
            printf("Enter the cost 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;
                    }

            }
            ne=1;
            printf("Minimum cost spanning tree edges\n");
            while(ne<n)
            {
                    for(min=999,i=1;i<=n;i++)
                    {
                            for(j=1;j<=n;j++)
                                    if(cost[i][j]<min)

                                    {

                                            min=cost[i][j];
                                            a=u=i;
                                            b=v=j;
                                    }

                    }
                    while(parent[u]!=0)
                            u=parent[u];
                    while(parent[v]!=0)
                            v=parent[v];
                    if(v!=u)

                    {
                            printf("%d:(%d->%d)=%d\n",ne++,a,b,min);
                            mincost+=min;
                            parent[v]=u;

                    }

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

            printf("The minimum cost of spanning tree is =%d ",mincost);
    }
```

2. Design and implement C/C++ Program to find Minimum Cost Spanning Tree of a given connected undirected graph using Prim's algorithm.

```
#include<stdio.h>
int main()
{
 int mincost=0,n,i,j,ne,a=0,b=0,min,u=0,v=0;
 int cost[10][10],visited[10];
 printf("Enter the number of vertices \n");
 scanf("%d",&n);
 printf("Enter the cost 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;
  }
 }
 for(i=2;i<=n;i++)
 visited[i]=0;
 visited[1]=1;
 ne=1;
 while(ne<n)
 {
  for(min=999,i=1;i<=n;i++)
  {
   for(j=1;j<=n;j++)
   {
    if(cost[i][j]<min)
    {
     if(visited[i]==0)
      continue;
     else
     {
      min=cost[i][j];
      a=u=i;
      b=v=j;
     }
    }
   }
  }
 if(visited[u]==0||visited[v]==0)
 {
  printf("\n %d edge (%d,%d)=%d",ne++,a,b,min);
  mincost=mincost+min;
  visited[v]=1;
 }
 cost[a][b]=cost[b][a]=999;
}
printf("\n The minimum cost of spanning tree is =%d",mincost);
}
```

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

```
#include<stdio.h>
int min(int a, int b)

{

 return(a<b)? a: b ;
}

void floyds(int cost[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++)

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

}

void main()

{

 int n,i,j,cost[10][10];
 printf("enter the new vertices");
 scanf("%d",&n);
 printf("enter the cost adjacency matrix (enter 999 for infinity)");
 for(i=1;i<=n;i++)
  for(j=1;j<=n;j++)

  {

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

  }

floyds(cost,n);

printf("all pains shortest paths matrix \n ");

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

{

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

 {

  printf("%d \t" , cost[i][j]);

 }

 printf("\n");

 }

}
```

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

```
#include<stdio.h>

void warshal(int adj[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++)
    adj[i][j]= adj[i][j] || adj[i][k] && adj[k][j] ;
}
int main()
{
 int n,i,j,adj[10][10];
 printf("enter the new vertices");
 scanf("%d",&n);
 printf("enter the cost adjacency matrix (enter 999 for infinity)");
 for(i=1;i<=n;i++)
  for(j=1;j<=n;j++)
  {
   scanf("%d",&adj[i][j]);
  }
warshal(adj,n);
printf("all pains shortest paths matrix \n ");
for(i=1;i<=n;i++)
{
 for(j=1;j<=n;j++)
 {
  printf("%d \t" , adj[i][j]);
 }
 printf("\n");
 }
}
```

4. Design and implement C/C++ Program to find shortest paths from a given vertex in a weighted connected graph to other vertices using Dijkstra's algorithm.

```
#include<stdio.h>
void dijikstras(int cost[][100],int dist[],int n,int v)
{
        int i, u=0,w,count,min;
        int flag[100]={0};
        for(i=1;i<=n;i++)
        {
                flag[i]=0;
                dist[i]=cost[v][i];
        }
        flag[v]=1;
        dist[v]=0;
        count=2;
        while(count<n)
```

```
    {
        for (i=1; min=999; i<=n ; i++)
        {
            if ((dist[i]<min)&&(flag[i]==0))
                {
                    min=dist[i];
                    u=i;
                }
        }
        flag[u]=1;
        count++;
        for(w=1;w<=n;w++)
        {
            if((dist[u]+cost[u][w]<dist[w])&&flag[w]==0)
            {
                dist[w]=dist[u]+cost[u][w];
            }
        }
    }
}
int main()
{
int n,source,i,j;
int cost[100][100];
int dist[100];
printf("Enter the number of vertices \n");
scanf("%d",&n);
printf("Enter 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("Source\n");
        scanf("%d",&source);
        dijikstras(cost,dist,n,source);
        for(i=1;i<=n;i++)
        {
            if(source!=i)
                printf("%d->%d::%d\n",source,i,dist[i]);
        }
return 0;
}
```

5. Design and implement C/C++ Program 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]);
   }

}
```

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;
}
```

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

```
#include <stdio.h>
#define MAX 50
int p[MAX], w[MAX], x[MAX];
double maxprofit;
int n, m, i;
void greedyKnapsack(int n, int w[], int p[], int m)
{
  double ratio[MAX];
  for (i = 0; i < n; i++)
  {
    ratio[i] = (double)p[i] / w[i];
  }
// Sort items based on the ratio in non-increasing order
  for (i = 0; i < n - 1; i++)
  {
    for (int j = i + 1; j < n; j++)
    {
      if (ratio[i] < ratio[j])
      {
        double temp = ratio[i];
```

```
                ratio[i] = ratio[j];
                ratio[j] = temp;

                int temp2 = w[i];
                w[i] = w[j];
                w[j] = temp2;

                temp2 = p[i];
                p[i] = p[j];
                p[j] = temp2;
            }
        }
    }
    int currentWeight = 0;
    maxprofit = 0.0;
// Fill the knapsack with items
    for (i = 0; i < n; i++)
    {
        if (currentWeight + w[i] <= m)
        {
            x[i] = 1; // Item i is selected
            currentWeight += w[i];
            maxprofit += p[i];
        }
        else
        {
// Fractional part of item i is selected
            x[i] = (m - currentWeight) / (double)w[i];
            maxprofit += x[i] * p[i];
            break;
        }
    }
    printf("Optimal solution for greedy method: %.1f\n", maxprofit);
    printf("Solution vector for greedy method: ");
    for (i = 0; i < n; i++)
        printf("%d\t", x[i]);
}
int main()
{
    printf("Enter the number of objects: ");
    scanf("%d", &n);
    printf("Enter the objects' weights: ");
    for (i = 0; i < n; i++)
        scanf("%d", &w[i]);
    printf("Enter the objects' profits: ");
    for (i = 0; i < n; i++)
        scanf("%d", &p[i]);
    printf("Enter the maximum capacity: ");
    scanf("%d", &m);
    greedyKnapsack(n, w, p, m);
    return 0;
}
```

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>

    int flag, count;
    int x[100], w[100], d, n;

    void sum(int s, int k, int r)
    {
        x[k] = 1;
            if (s + w[k] == d)
        {
             printf("\nsubset :%d", ++count);
             flag = 1;
             for (int i = 0; i <= k; i++)
                 if (x[i] == 1)
                 printf("%d ", w[i]);
            }
        else if (s + w[k] + w[k + 1] <= d)
            sum(s + w[k], k + 1, r - w[k]);
        if ((s + r - w[k]) >= d && (s + w[k + 1] <= d))
        {
               x[k] = 0;
               sum(s, k + 1, r - w[k]);
        }
    }

    int main()
    {
      int r = 0;
          flag = 0;
          printf("enter the total no of elements:");
          scanf("%d", &n);
          for (int i = 0; i < n; i++)
          scanf("%d", &w[i]);
          printf("enter the value of sum:");
          scanf("%d", &d);
          for (int i = 0; i < n; i++)
          x[i] = 0;
          for (int i = 0; i < n; i++)
           r += w[i];
          if (r < d)
          {
          printf("no subset is possible\n");
          flag = 1;
          }
          else
          sum(0, 0, r);
          if (flag == 0)
           printf("no more subset is possible\n");
          return 0;
       }
```

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<time.h>
#include<stdlib.h>
int main()
{
  int temp,min,j,i,n,a[100000],choice;
  clock_t t;
  printf("enter the number of elements :");
  scanf("%d",&n);
  for(i=0;i<n;i++)
  {
   a[i]=rand()%1000;
   printf("\n%d",a[i]);
  }
  t = clock();
  for(i=0;i<n-2;i++)
 {
 min = i;
 for(j=i+1;j<n-1;j++)
 {
  if(a[j] <a[min])
     min = j;
 }
 temp = a[i];
 a[i] = a[min];
 a[min] = temp;
}
  t = clock()-t;
  double time =((double)t)/CLOCKS_PER_SEC;
  printf("entered number after sorting\n");
  for (i=0;i<n;i++)
     printf("%d\n",a[i]);
  printf("sort function took %f sec to execute",time);
  return 0;
  }
```

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<time.h>
#include<stdlib.h>
void swap (int a[], int i, int j)
{
int temp;
temp=a[i];
a[i]=a[j];
a[j]=temp;
}
int partition(int a[],int l,int r)
{
int i,j;
int p;
p=a[l];
i=l;
j=r+1;
do
{
do{   i++;}while(a[i]<p);
do{   j--;}while(a[j]>p);
swap(a,i,j);
}while(i<j);
swap(a,i,j);
swap(a,l,j);
return j;
}
void quicksort(int a[], int l, int r)
{
int s;
if(l<r)
{
s=partition(a,l,r);
quicksort(a,l,s-1);
quicksort(a,s+1,r);
}
}
int main()
{
int temp,min,j,i,n,a[100000],choice;
clock_t t;
printf("enter the numbers of elements");
scanf("%d",&n);
printf("1.Read from file 2.Random numbers");
 scanf("%d", &choice);
switch(choice)
{
case 1:
```

```
printf("File numbers\n");
FILE*file=fopen("data.txt","r");
int i=0;
while(! feof(file) && i<n)
{
fscanf(file, "%d",&a[i]);
printf("%d\n",a[i]);
i++;
}
fclose(file);
break;
case2:printf("Random number generator");
for(i=0;i<n;i++)
{
a[i]=rand()%1000;
printf("%d\n", a[i]);
}
break;
}
t=clock();
quicksort(a,0,n-1);
t=clock()-t;
double time_taken=((double)t)/CLOCKS_PER_SEC;
printf("entered numbers are after sorting");
for(i=0;i<n;i++)
printf("%d\n",a[i]);
printf("sort function took %f seconds to execute \n",time_taken);
return 0;
}
```

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>
int merge(int b[],int c[],int a[],int p,int q,int n)
{
 int i,j,k;
 i=j=k=0;
 while(i<p && j<q)
 {
  if(b[i]<=c[j])
  {
   a[k]=b[i];
   i++;
  }
  else
  {
   a[k]=c[j];
```

```
      j++;
    }
      k++;
  }
  if(i==p)
  {
    while(j<q)
    {
      a[k]=c[j];
      k++;
      j++;
    }
  }
  else
  {
    while(i<p && k<n)
      a[k++]=b[i++];
  }
 }
 int mergesort(int a[],int n)
 {
   int b[n/2];
   int c[n-n/2];
   int i,j;
   if(n>1)
   {
     for(i=0;i<n/2;i++)
       b[i]=a[i];
     for(i=n/2,j=0;i<n;i++,j++)
       c[j]=a[i];
     mergesort(b,n/2);
     mergesort(c,n-n/2);
     merge(b,c,a,n/2,n-n/2,n);
   }
}
int main()
{
  int temp,min,j,i,n,a[100000],choice;
  clock_t t;
  printf("enter the number of elements :");
  scanf("%d",&n);
  printf("1. Read from file             2. Random numbers");
  scanf("%d",&choice);
  switch(choice)
  {
     case 1:
          printf("file numbers\n");
          FILE *file = fopen("num.txt","r");
          int i=0;
          while(!feof(file) && i<n)
          {
             //printf("%d ",i+1);
             fscanf(file,"%d",&a[i]);
      printf("%d\n",a[i]);
      i++;
    }
    fclose(file);
    break;
  case 2:
    printf("Random number generator");
    for(i=0;i<n;i++)
    {
      a[i] = rand()%1000;
      printf("%d\n",a[i]);
    }
    break;
}
t = clock();
mergesort(a,n);
t = clock()-t;
double time =((double)t)/CLOCKS_PER_SEC;
printf("entered number after sorting\n");
for (i=0;i<n;i++)
  printf("%d\n",a[i]);
printf("sort function took %f sec to execute",time);
return 0;
}
```

12. Design and implement C/C++ Program for N Queen's problem using Backtracking.

```
#include <stdio.h>
        #include <stdlib.h>
        int x[10];
        int place(int k,int i)
        {
                int j;
                for(j=1;j<=k-1;j++)
                if(x[j]==i || abs(x[j]-i)==abs(j-k))
                return 0;
                return 1;
        }
        void display(int n)
        {
                int k,i,j;
                char cb[n][n];
                for(k=1;k<=n;k++)
                cb[k][x[k]]='Q';
                for(i=1;i<=n;i++)
                {
                        for(j=1;j<=n;j++)
                        {
                        if(j!=x[i])
                        cb[i][j]='-';
                        }
                }
```

```
	for(i=1;i<=n;i++)
	{
		for(j=1;j<=n;j++)
		printf("%c\t",cb[i][j]);
		printf("\n");
	}
	printf("\n\n");
}
void NQueens(int k,int n)
{
	int i;
	for(i=1;i<=n;i++)
	if(place(k,i))
	{
		x[k]=i;
		if(k==n)
		{
		printf("Solution\n");
		display(n);
		}
		else
		NQueens(k+1,n);
	}
}
int main(void)
{
	int n,k=1;
	printf("Enter the dimensions of the chessboard\n");
	scanf("%d",&n);
	if(n==2 || n==3)
	{
		printf("No solution\n");
		exit(0);
	}
	NQueens(k,n);
	return 0;
}
```