```
/*Design and implement C/C++ Program for N Queen's problem using Backtracking.*/
#include <stdio.h>
#define N 4

void printBoard(int board[N][N]) {
    for (int i = 0; i < N; i++, printf("\n"))
        for (int j = 0; j < N; j++)
            printf("%d ", board[i][j]);
}

int isSafe(int board[N][N], int row, int col) {
    for (int i = 0; i < col; i++)
        if (board[row][i]) return 0;
    for (int i = row, j = col; i >= 0 && j >= 0; i--, j--)
        if (board[i][j]) return 0;
    for (int i = row, j = col; i < N && j >= 0; i++, j--)
        if (board[i][j]) return 0;
    return 1;
}

int solveNQueens(int board[N][N], int col) {
    if (col >= N) return 1;
    for (int i = 0; i < N; i++) {
        if (isSafe(board, i, col)) {
            board[i][col] = 1;
            if (solveNQueens(board, col + 1)) return 1;
            board[i][col] = 0;
        }
    }
    return 0;
}

int main() {
    int board[N][N] = {0};
    if (solveNQueens(board, 0))
        printBoard(board);
    else
        printf("Solution does not exist\n");
    return 0;
}
```

```
/*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<conio.h>
#include<time.h>
#define MAX 100000

void simple_merge(int a[],int low,int mid,int high)
{
int i=low;
int j=mid+1;
int k=low;
int c[MAX];
while(i<=mid &&j<=high)
{
if(a[i]<a[j])
{
c[k]=a[i];
i++;
k++;
}
else
{
c[k]=a[j];
j++;
k++;
}

}
while(i<=mid)
c[k++]=a[i++];
while(j<=high)
{
c[k++]=a[j++];
}
for(i=low;i<=high;i++)
a[i]=c[i];
}
```

```
void merge_sort(int a[],int low,int high)
{
int mid;
if(low<high)
{
mid=(low+high)/2;
merge_sort(a,low,mid);
merge_sort(a,mid+1,high);
simple_merge(a,low,mid,high);
}
}
int main()
{
    int a[100000], i, n;
    printf("\nEnter the n value:");
    scanf("%d", &n);
    for(i = 0; i < n; i++)
    {
       a[i]=rand()%100;

    }

        clock_t start = clock();
    merge_sort(a,0,n-1);
   clock_t end = clock();
    double timeTaken = ((double)(end - start)) / CLOCKS_PER_SEC;


       printf("Time taken to sort %d elements: %f seconds\n", n, timeTaken);

    return 0;
}
```

```
/*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*/
#include<stdio.h>
#include<time.h>
int partition(int low,int high,int a[])
{
   int i,j,key,temp;
   i=0,j=high+1;
   key=a[low];
   while(i<=j)
   {
      do i++;while(key>=a[i]);
      do j--;while(key<a[j]);
      if(i<j)
      {
             temp=a[i];
             a[i]=a[j];
             a[j]=temp;
      }
   }
      temp=a[low];
      a[low]=a[j];
      a[j]=temp;
      return j;
}
void quick_sort(int low,int high,int a[])
{
    int mid;
    if(low<high)
    {
       mid=partition(low,high,a);
       quick_sort(low,mid-1,a);
       quick_sort(mid+1,high,a);
    }
}


int main()
{
   int a[10000], i, n;
    printf("\nEnter the n value:");
    scanf("%d", &n);
    for(i = 0; i < n; i++)
    {
       a[i]=rand()%100;

    }

        clock_t start = clock();
    quick_sort(0,n,a);
   clock_t end = clock();
    double timeTaken = ((double)(end - start)) / CLOCKS_PER_SEC;


       printf("Time taken to sort %d elements: %f seconds\n", n, timeTaken);

    return 0;
}
```

```
/*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>

void sort(int a[],int n)
{
 int min,i,j,temp;
 for(i=0;i<n-2;i++)
 {
  min=i;
  for(j=i+1;j<n-1;j++)
  {
   if(a[j]<a[min])
   {
    min=j;
    //count++;
   }
  }
  temp=a[i];
  a[i]=a[min];
  a[min]=temp;
 }
}

int main()
{
    int a[10000], i, n;
    printf("\nEnter the n value:");
    scanf("%d", &n);
    for(i = 0; i < n; i++)
    {
       a[i]=rand()%100;

    }

        clock_t start = clock();
    sort(a, n);
   clock_t end = clock();
```

```
    double timeTaken = ((double)(end - start)) / CLOCKS_PER_SEC;


       printf("Time taken to sort %d elements: %f seconds\n", n, timeTaken);

    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>
#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;
}
```

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


#include<stdio.h>
int main()
{
     float weight[50],profit[50],ratio[50],Totalvalue,temp,capacity,amount;
     int n,i,j;
     printf("Enter the number of items :");
     scanf("%d",&n);
    for (i = 0; i < n; i++)
    {
        printf("Enter Weight and Profit for item[%d] :\n",i);
        scanf("%f %f", &weight[i], &profit[i]);
    }
    printf("Enter the capacity of knapsack :\n");
    scanf("%f",&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;
         }

     printf("Knapsack problems using Greedy Algorithm:\n");
     for (i = 0; i < n; i++)
     {
      if (weight[i] > capacity)
```

```
         break;
      else
     {
         Totalvalue = Totalvalue + profit[i];
         capacity = capacity - weight[i];
      }
    }
      if (i < n)
      Totalvalue = Totalvalue + (ratio[i]*capacity);
    printf("\nThe maximum value is
    :%f\n",Totalvalue); return 0;
}
```

```
/*Design and implement C/C++ Program to solve 0/1 Knapsack problem using Dynamic
Programming method.*/
#include<stdio.h>

int max(int a, int b) { return (a > b)? a : b; }

int knapSack(int W, int wt[], int p[], int n)
{
   int i, j;
   int V[n+1][W+1];

   for (i = 0; i <= n; i++)
   {
       for (j = 0; j <= W; j++)
       {
           if (i==0 || j==0)
               V[i][j] = 0;
           else if (wt[i] <= j)
                 V[i][j] = max( V[i-1][j],p[i] + V[i-1][j-wt[i]]);
           else
                 V[i][j] = V[i-1][j];
       }
   }

   return V[n][W];
}

int main()
{
    int i, n, p[20], wt[20], W;

    printf("Enter number of items:");
    scanf("%d", &n);

    printf("Enter value and weight of items:\n");
    for(i = 1;i <= n; ++i){
            scanf("%d%d", &p[i], &wt[i]);
    }

    printf("Enter size of knapsack:");
    scanf("%d", &W);

    printf("Max Profit=%d", knapSack(W, wt, p, n));
    return 0;
}
```

```
/*5-Design and implement C/C++ Program to obtain the Topological ordering of vertices in a
given digraph.*/
#include<stdio.h>
#include<conio.h>
void main()
{
        int a[10][10],t[10],indeg[10],n,SUM=0;
        int u,k=0,v;
        int i,j,stack[10],top=-1;
        printf("\n\n\t topological ordering \n\n");
        printf("enter the directed acyclic graph\n\n");
        printf("enter the no of vertex\t");
        scanf("%d",&n);
        printf("enter the adjacency matrix\n");
        for(i=0;i<n;i++)
        {
                for(j=0;j<n;j++)
                {
                        scanf("%d",&a[i][j]);
                }
        }
        for(i=0;i<n;i++)
                indeg[i]=0;
        for(j=0;j<n;j++)
        {
                SUM=0;
                for(i=0;i<n;i++)
                {
                        SUM+=a[i][j];
                }
                indeg[j]=SUM;
        }
        for(i=0;i<n;i++)
        {
                if(indeg[i]==0)
                {
                        stack[++top]=i;
                }
        }
        while(top!=-1)
        {
                u=stack[top--];
                t[k++]=u;
        for(v=0;v<n;v++)
        {
            if(a[u][v]==1)
            {
                indeg[v]--;
                if(indeg[v]==0)
                {
                    stack[++top]=v;
                }
            }
        }
    }
    printf("the topological sorting list\n");
    for(i=0;i<n;i++)
    {
        printf("%d\t",t[i]+1);
    }
}
```

```
#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;
}
```

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

#include<stdio.h>
#define INF 999
int a[10][10];
int min(int a,int b)
{
    return(a<b)?a:b;
}
void floyd(int n)
{
     int i,j,num;
     int k;
    for(k=1; k<=n; k++)
        for(i=1; i<=n; i++)
            for(j=1; j<=n; j++)
                a[i][j]=min(a[i][j],a[i][k]+a[k][j]);

    printf("\nShortest path matrix\n");
    for(i=1; i<=n; i++)
    {
        for(j=1; j<=n; j++)
            printf("%d ",a[i][j]);
        printf("\n");
    }
}
void main()
{
    int n,i,j;
    printf("\nEnter the 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",&a[i][j]);
    floyd(n);


}
```

```
/*b. Design and implement C/C++ Program to find the transitive closure using Warshal's
algorithm.*/
#include<stdio.h>
#include<stdlib.h>
int a[10][10];
void warshall(int n)
{
 int i,j,num;
 int k;
 for(k=1;k<=n;k++)
 {
  for(i=1;i<=n;i++)
  {
   for(j=1;j<=n;j++)
   a[i][j]=a[i][j]||(a[i][k]&&a[k][j]);
  }
 }
 printf("\nthe transitive closure matrix is:\n");
 for(i=1;i<=n;i++)
 {
  for(j=1;j<=n;j++)
  {
   printf("%d",a[i][j]);
  }
  printf("\n");
 }
}
void main()
{
 int i,j,n;
 printf("enter the no of vertices\n");
 scanf("%d",&n);
 printf("enter the adjacency matrix\n");
 for(i=1;i<=n;i++)
 for(j=1;j<=n;j++)
 {
  scanf("%d",&a[i][j]);
 }
 warshall(n);

}
```

```
/*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>

#include<stdlib.h>
int u,v,n,i,j,ne=1;
int visited[10]= {0},min,mincost=0,cost[10][10];
void main() {
        printf("\n Enter the number of nodes:");
        scanf("%d",&n);
        printf("\n Enter the 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;
        }
        visited[1]=1;
        printf("\n");
        while(ne<n) {
                for (i=1,min=999;i<=n;i++)
                   for (j=1;j<=n;j++)
                    if(cost[i][j]<min)
                     if(visited[i]!=0) {
                        min=cost[i][j];
                        u=i;
                        v=j;
                }
                if(visited[u]==0 || visited[v]==0) {
                        printf("\n Edge %d:(%d %d) cost:%d",ne++,u,v,min);
                        mincost+=min;
                        visited[v]=1;
                }
                cost[u][v]=cost[v][u]=999;
        }
        printf("\n Minimun cost=%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>

#include<stdlib.h>
int u,v,n,i,j,ne=1,k,a,b;
int parent[10],min,mincost=0,cost[10][10];
int find(int);
int union1(int,int);
void main()
{
        printf("\n Enter the number of nodes:");
        scanf("%d",&n);
        printf("\n Enter the 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("MST\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=fipar(u);
                v=fipar(v);
                if(union1(u,v))
     {
                        printf("\n %d:edge(%d %d) cost:%d",ne++,a,b,min);
                        mincost+=min;

                }
                cost[a][b]=cost[b][a]=999;
        }
        printf("\n Minimun cost=%d",mincost);
}
int fipar(int i)
{
    while(parent[i])
        i=parent[i];
        return i;
}
int union1(int i,int j)
{
   if(i!=j)
{
    parent[j]=i;
    return 1;
}
return 0;
}
```