/* 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>
#include<conio.h>
#define INFINITY 999
#define MAX 100
int parent([MAX],cost[MAX][MAX],t[MAX][2]);
int find(int v)
{
	while(parent[v])
	{
		v=parent[v];
	}
	return v;
}
void union1(int i,int j)
{
	parent[j]=i;
}
void kruskal(int n)
{
	int i,j,k,u,v,mincost,res1,res2,sum=0;
	for(k=1;k<n;k++)
	{
		mincost=INFINITY;
		for(i=1;i<n;i++)
		{
			for(j=1;j<=n;j++)
			{
				if(i==j)  continue;
				if(cost[i][j]<mincost)
				{
					u=find(i);
					v=find(j);
					if(u!=v)
					{
						res1=i;
						res2=j;
						mincost=cost[i][j];
					}
				}
			}
		}
		union1(res1,find(res2));
		t[k][1]=res1;
		t[k][2]=res2;
		sum=sum+mincost;
	}
	printf("\nCost of spanning tree is %d\n",sum);
```

```
        printf("\nEdges of spanning tree are\n");
        for(i=1;i<n;i++)
        printf("%d->%d\n",t[i][1],t[i][2]);
}
void main()
{
        int i,j,n;
        clrscr();
        printf("\nEnter the number of vertices :  ");
        scanf("%d",&n);
        for(i=1;i<=n;i++)
        parent[i]=0;
        printf("\nEnter the cost adjacency matrix 0-for self edge and 999-if no edge\n");
        for(i=1;i<=n;i++)
        for(j=1;j<=n;j++)
        scanf("%d",&cost[i][j]);
         kruskal(n);
        getch();
  }
```

/* 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<conio.h>
#define INFINITY 999
int prim(int cost[10][10],int source,int n)
{
            int i,j,sum=0,visited[10];
            int distance[10],vertex[10];
            int min,u,v;
       for(i=1;i<=n;i++)
      {
          vertex[i]=source;
          visited[i]=0;
          distance[i]=cost[source][i];
      }
      visited[source]=1;
      for(i=1;i<n;i++)
     {
          min=INFINITY;
           for(j=1;j<=n;j++)
          {
                              if(!visited[j]&&distance[j]<min)
                              {
                                        min=distance[j];
                                        u=j;
                               }
                    }
             visited[u]=1;
             sum=sum+distance[u];
             printf("\n%d->%d",vertex[u],u);
             for(v=1;v<=n;v++)
                      {
                              if(!visited[v]&&cost[u][v]<distance[v])
                              {
                                        distance[v]=cost[u][v];
                                        vertex[v]=u;
                              }
          }
    }
    return sum;
}
void main()
{
  int a[10][10],n,i,j,m,source;
  clrscr();
  printf("\n enter the  number of vertices:\n");
  scanf("%d",&n);
  printf("\nenter the cost matrix:\n 0-self loop and 999-no edge:\n");
```

```
  for(i=1;i<=n;i++)
  for(j=1;j<=n;j++)
  scanf("%d",&a[i][j]);
  printf("\n enter the source:\n");
  scanf("%d",&source);
  m=prim(a,source,n);
  printf("\n the cost of spanning tree=%d",m);
  getch();
}
```

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

```
#include<stdio.h>
#define INFINITY 999
int min(int i,int j)
{
        if(i<j)
                return i;
        else
                return j;
}
void floyd(int n,int p[10][10])
{
         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]=min(p[i][j],p[i][k]+p[k][j]);
}
int main()
{
        int i,j,n,a[10][10],d[10][10],source;
        double starttime,end time;
        printf("Enter the no.of nodes: ");
        scanf("%d",&n);
        printf("\nEnter the adjacency matrix\n");
        for(i=1;i<=n;i++)
                for(j=1;j<=n;j++)
                        scanf("%d",&a[i][j]);
        floyd(n,a);
        printf("\n\nThe distance matrix is \n");
        for(i=1;i<=n;i++)
        {
                for(j=1;j<=n;j++)
                        printf("%d\t",a[i][j]);
                printf("\n");
        }
        return 0;
}
```

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

```
#include<stdio.h>
#include<conio.h>
void warshall(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(p[i][k]==1 && p[k][j]==1)
				{
					p[i][j]=1;
				}
			}
		}
	}
}
void main()
{
	int a[10][10],i,j,n;
	clrscr();
	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(a,n);
	printf(" the resultant path matrix:\n");
	for(i=1;i<=n;i++)
	{
		for(j=1;j<=n;j++)
		{
			printf("%d\t",a[i][j]);
		}
		printf("\n");
	}
	getch();
}
```

/* 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>
#include<conio.h>
#define INFINITY 999
void dijkstra(int cost[10][10],int n,int source,int distance[10])
{
	int visited[10],min,u;
	int i,j;
	for(i=1;i<=n;i++)
	{
		distance[i]=cost[source][i];
		visited[i]=0;
	}
	visited[source]=1;
	for(i=1;i<=n;i++)
	{
		min=INFINITY;
		for(j=1;j<=n;j++)
		if(visited[j]==0 && distance[j]<min)
		{
			min=distance[j];
			u=j;
		}
		visited[u]=1;
		for(j=1;j<=n;j++)
		if(visited[j]==0 && (distance[u]+cost[u][j])<distance[j])
		{
			distance[j]=distance[u]+cost[u][j];
		}
	}
}
void main()
{
	int n,cost[10][10],distance[10];
	int i,j,source,sum;
	clrscr();
	printf("\nEnter how many nodes :  ");
	scanf("%d",&n);
	printf("\nCost Matrix\nEnter 999 for no edge\n");
	for(i=1;i<=n;i++)
	for(j=1;j<=n;j++)
	scanf("%d",&cost[i][j]);
	printf("Enter the source node\n");
	scanf("%d",&source);
	dijkstra(cost,n,source,distance);
	for(i=1;i<=n;i++)
	printf("\n\nShortest Distance from  %d to %d is %d",source,i,distance[i]);
	getch();
}
```

/*5. Design and implement C/C++ Program to obtain the Topological ordering of vertices in a given digraph. */

```
#include<stdio.h>
#include<conio.h>
int temp[10],k=0;
void topo(int n,int indegree[10],int a[10][10])
{
	int i,j;
	for(i=1;i<=n;i++)
	{
		if(indegree[i]==0)
		{
			indegree[i]=-1;
			temp[++k]=i;
			for(j=1;j<=n;j++)
			{
				if(a[i][j]==1 && indegree[j]!=-1)
				indegree[j]--;
			}
			i=0;
		}
	}
}
void main()
{
	int i,j,n,indegree[10],a[10][10];
	clrscr();
	printf("Enter the number of vertices: ");
	scanf("%d",&n);
	for(i=1;i<=n;i++)
	indegree[i]=0;
	printf("Enter the adjacency matrix\n");
	for(i=1;i<=n;i++)
	for(j=1;j<=n;j++)
	{
		scanf("%d",&a[i][j]);
		if(a[i][j]==1)
		indegree[j]++;
	}
	topo(n,indegree,a);
	 if(k!=n)
	printf("\nTopological ordering is not possible\n");
	else
	{
		printf("The topological ordering is \n");
		for(i=1;i<=k;i++)
		printf("%d\t",temp[i]);
	}
	getch();
}
```

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

```
#include<stdio.h>
#include<conio.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]);
}
void main()
{
        int m,i,max_profit;
        clrscr();
        printf("\nEnter the number of objects: ");
        scanf("%d",&n);
        printf("\nEnter the knapsack capacity: ");
        scanf("%d",&m);
        printf("\nEnter profit followed by weight: ");
        for(i=1;i<=n;i++)
        scanf("%d%d",&p[i],&w[i]);
        max_profit=knap(1,m);
        printf("\nMax profit = %d",max_profit);
        getch();
}
```

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

```
#include <stdio.h>

// Structure to represent items
struct Item {
   int value;
   int weight;
};

// Function to compare items by their value-to-weight ratio
int compare(const void *a, const void *b) {
   double ratio_a = ((double)(((struct Item*)a)->value) / ((struct Item*)a)->weight);
   double ratio_b = ((double)(((struct Item*)b)->value) / ((struct Item*)b)->weight);
   if (ratio_a < ratio_b)
      return 1;
   else if (ratio_a > ratio_b)
      return -1;
   else
      return 0;
}

// Function to solve discrete knapsack problem using greedy approximation method
int discreteKnapsack(struct Item items[], int n, int capacity) {
   qsort(items, n, sizeof(items[0]), compare); // Sort items based on value-to-weight ratio
   int totalValue = 0;
   int currentWeight = 0;
   for (int i = 0; i < n; i++) {
      if (currentWeight + items[i].weight <= capacity) {
         totalValue += items[i].value;
         currentWeight += items[i].weight;
      } else {
         // Take a fraction of the item if it cannot be taken fully
         double remainingCapacity = capacity - currentWeight;
         totalValue += (int)((double)items[i].value / items[i].weight * remainingCapacity);
         break;
      }
   }
   return totalValue;
}

// Function to solve continuous knapsack problem using greedy approximation method
double continuousKnapsack(struct Item items[], int n, int capacity) {
   qsort(items, n, sizeof(items[0]), compare); // Sort items based on value-to-weight ratio
   double totalValue = 0.0;
   int currentWeight = 0;
   for (int i = 0; i < n; i++) {
      if (currentWeight + items[i].weight <= capacity) {
         totalValue += items[i].value;
```

```
            currentWeight += items[i].weight;
        } else {
            // Take a fraction of the item if it cannot be taken fully
            double remainingCapacity = capacity - currentWeight;
            totalValue += (double)items[i].value / items[i].weight * remainingCapacity;
            break;
        }
    }
    return totalValue;
}

int main() {
    // Example usage
    struct Item items[] = {{60, 10}, {100, 20}, {120, 30}};
    int n = sizeof(items) / sizeof(items[0]);
    int capacity = 50;

    // Solving discrete knapsack problem
    int discreteMaxValue = discreteKnapsack(items, n, capacity);
    printf("Maximum value (discrete knapsack): %d\n", discreteMaxValue);

    // Solving continuous knapsack problem
    double continuousMaxValue = continuousKnapsack(items, n, capacity);
    printf("Maximum value (continuous knapsack): %.2lf\n", continuousMaxValue);

    return 0;
}
```

```
#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;
}
```

output:-

```
/*Enter the number of items :4
Enter Weight and Profit for item[0] :
2
12
```

```
Enter Weight and Profit for item[1] :
1
10
Enter Weight and Profit for item[2] :
3
20
Enter Weight and Profit for item[3] :
2
15
Enter the capacity of knapsack :
5
Knapsack problems using Greedy Algorithm:

The maximum value is :38.333332
--------------------------------*/
```

/* 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>
#include<conio.h>
#define MAX 10
int s[MAX],x[MAX];
int 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]);
        }
}
void main()
{
        int i,n,sum=0;
        clrscr();
        printf("\nEnter max number : ");
        scanf("%d",&n);
        printf("\nEnter the set in increasing order : \n");
        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);
        getch();
}
```

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


void selsort(int a[], int n) {
  int i, j, small, pos, temp;
  for (j = 0; j < n - 1; j++) {
      small = a[j];
      pos = j;
      for (i = j + 1; i < n; i++) {
          if (a[i] < small) {
              small = a[i];
              pos = i;
          }
      }
      temp = a[j];
      a[j] = a[pos];
      a[pos] = temp;
  }
}


int main() {
  int *a, i, n;
  struct timespec start, end;
  double dura;


  printf("\nEnter the number of elements (n): ");
  scanf("%d", &n);
```

```
a = (int*)malloc(n * sizeof(int));
if (a == NULL) {
    printf("Memory allocation failed!\n");
    return 1;
}


srand(time(NULL));


printf("\nGenerated array: ");
for (i = 0; i < n; i++) {
    a[i] = rand() % 1000;
    printf("%d ", a[i]);
}
printf("\n");


clock_gettime(CLOCK_MONOTONIC, &start);
selsort(a, n);
clock_gettime(CLOCK_MONOTONIC, &end);


dura = (end.tv_sec - start.tv_sec) + (end.tv_nsec - start.tv_nsec) / 1e9;
printf("\nTime taken to sort: %lf seconds\n", dura);


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

```
  free(a);


  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<conio.h>
#include<time.h>

int partition(int a[],int low,int high)
{
	int i,j,temp,key;
	 key=a[low];
	 i=low;
	 j=high+1;

	while(i<=j)
	{
		do i++;  while(i<=high&&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 quicksort(int a[],int low,int high)
{
	int mid;
	if(low<high)
	{
		mid=partition(a,low,high);
		delay(100);
		quicksort(a,low,mid-1);
		quicksort(a,mid+1,high);
	}
}
```

```
void main()
{
	int a[1000];
	int n,i;
	clock_t start,end;
	clrscr();
	printf("enter the no of elements\n");
	scanf("%d",&n);
	printf("enter the array elements\n");
	for(i=0;i<n;i++)
		a[i]=rand()%100;
	for(i=0;i<n;i++)
		printf(“%d\n”,a[i]);
	start=clock();
	quicksort(a,0,n-1);
	end=clock();
	printf("the sorted elements are\n");
	for(i=0;i<n;i++)
		printf("%d\t",a[i]);
	printf("the time taken is %f",(double)(end-start)/(CLK_TCK));
	getch();
}
```

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

```
#include<stdio.h>
#include<stdlib.h>
#include<conio.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");
			 }
			else
```

```
                        {
                                r++;
                                c[r]=-1;
                        }
                }
                else
                        r--;
        }
}
void main()
{
        int n;
        clrscr();
        printf("\nEnter the number of queens : ");
        scanf("%d",&n);
        n_queens(n);
        getch();
}
```