RN SHETTY TRUST®

RNS INSTITUTE OF TECHNOLOGY

Autonomous Institution Affiliated to VTU, Recognized by GOK, Approved by AICTE
(NAAC 'A+ Grade' Accredited, NBA Accredited (UG - CSE, ECE, ISE, EIE and EEE)
Channasandra, Dr. Vishnuvardhan Road, Bengaluru - 560 098
Ph:(080)28611880,28611881 URL: www.rnsit.ac.in

Department of
Computer Science and Engineering
(Cyber Security)

BCSL404
Analysis and Design of Algorithms
Laboratory Manual
IV Semester

Faculty Name: _____

Designation: _____

Academic Year: _____

VISION OF THE COLLEGE

Building RNSIT into a World - Class Institution

MISSION OF THE COLLEGE

To impart high quality education in Engineering, Technology and Management with a difference, enabling students to excel in their career by

1. Attracting quality Students and preparing them with a strong foundation in fundamentals so as *to achieve distinctions in various walks of life* leading to outstanding contributions.

2. Imparting value based, need based, and choice based and skill based professional education to the aspiring youth and *carving them into disciplined, World class Professionals* with *social responsibility.*

3. Promoting excellence in Teaching, Research and Consultancy that galvanizes academic consciousness among Faculty and Students.

4. Exposing Students to emerging frontiers of knowledge in various domains and make them suitable for Industry, Entrepreneurship, Higher studies, and Research & Development.

5. Providing freedom of action and choice for all the Stake holders with better visibility.

VISION OF THE DEPARTMENT

To be a global leader in Cyber Security education, research, innovation, and nurturing young minds to safeguard the digital world

MISSION OF THE DEPARTMENT

1. Empower students with Cyber Security expertise through hands-on training and a strong ethical foundation
2. Prioritize collaborative learning environment, teamwork, and global certifications for real-time challenges.
3. Drive innovation in Cyber Security through cutting-edge research, fostering a culture of exploration through higher learning and entrepreneurship
4. Facilitate students to possess qualities of interpersonal, interdisciplinary, leadership, and societal responsibilities

PROGRAM EDUCATIONAL OBJECTIVES (PEOs)

CSE – Cyber Security Graduates within three-four years of graduation will be able to**:**

PEO1: Demonstrate applied skills and problem-solving capabilities in the field of cyber security, effectively addressing complex challenges such as threat detection, vulnerability assessment, and incident response.

PEO2: Develop strong communication skills, enabling them to convey technical concepts and findings effectively to diverse audiences, facilitating their employability in cyber security and related fields.

PEO3: Incorporate ethical, legal, and social considerations into their professional practice, recognizing the implications of their actions on individuals, organizations, and society.

PEO4: Actively engage in activities that foster the continuous development of their computing and cyber security skills, staying abreast of emerging technologies, best practices, and evolving threats to maintain their effectiveness as professionals in the field.

PROGRAM OUTCOMES (POs)

Engineering Graduates will be able to:

PO1: Engineering knowledge: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization for the solution of complex engineering problems

PO2: Problem analysis: Identify, formulate, research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences and engineering sciences.

PO3: Design/development of solutions: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for public health and safety, and cultural, societal, and environmental considerations.

PO4: Conduct investigations of complex problems: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

PO5: Modern tool usage: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools, including prediction and modeling to complex engineering activities, with an understanding of the limitations.

PO6: The engineer and society: Apply reasoning informed by the contextual knowledge to assess Societal, health, safety, legal and cultural

issues and the consequent responsibilities relevant to the professional engineering practice.

PO7: Environment and sustainability: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

PO8: Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

PO9: Individual and team work: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

PO10: Communication: Communicate effectively on complex engineering activities with the engineering community and with the society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

PO11: Project management and finance: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

PO12: Life-long learning: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

PROGRAM SPECIFIC OUTCOMES (PSOs)

Computer Science and Engineering – (Cyber Security) Graduates will have:

PSO1: Proficiency in applying principles of software development to deliver efficient software solutions.

PSO2: Competent in securing information, systems, and networks through the implementation of cybersecurity best practices.

ADA LABORATORY

Course objectives

- To design and implement various algorithms in C/C++ programming using suitable development tools to address different computational challenges.
- To apply diverse design strategies for effective problem-solving.
- To Measure and compare the performance of different algorithms to determine their efficiency and suitability for specific tasks.

Course Outcomes

After studying this course, students will be able to:

1	Apply asymptotic notational method to analyze the performance of the algorithms in terms of time complexity.
2	Demonstrate divide & conquer approaches and decrease & conquer approaches to solve computational problems.
3	Make use of transform & conquer and dynamic programming design approaches to solve the given real world or complex computational problems.
4	Apply greedy and input enhancement methods to solve graph & string based computational problems.
5	Illustrate backtracking, branch & bound and approximation methods.

CO mapping to PO/PSOs

CO / PO & PSO	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12	PSO1	PSO2

List of Programs

Sl. No	Name of Experiment	CO's
1	Design and implement C/C++ Program to find Minimum Cost Spanning Tree of a given connected undirected graph using Kruskal's algorithm.	CO4
2	Design and implement C/C++ Program to find Minimum Cost Spanning Tree of a given connected undirected graph using Prim's algorithm.	CO4
3	a. Design and implement C/C++ Program to solve All-Pairs Shortest Paths problem using Floyd's algorithm. b. Design and implement C/C++ Program to find the transitive closure using Warshal's algorithm.	CO3
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.	CO4
5	Design and implement C/C++ Program to obtain the Topological ordering of vertices in a given digraph.	CO2
6	Design and implement C/C++ Program to solve 0/1 Knapsack problem using Dynamic Programming method.	CO3
7	Design and implement C/C++ Program to solve discrete Knapsack and continuous Knapsack problems using greedy approximation method.	CO4
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.	CO5
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.	CO1
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.	CO2
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.	CO2
12	Design and implement C/C++ Program for N Queen's problem using Backtracking.	CO5

ANALYSIS AND DESIGN OF ALGORITHMS LABORATORY-BCSL404

INTERNAL EVALUATION SHEET

EVALUATION (MAX MARKS 40)			
TEST A	**REGULAR EVALUATION B**	**RECORD C**	**TOTAL MARKS A+B+C**
20	**10**	**20**	**50**

R1: REGULAR LAB EVALUATION WRITE UP RUBRIC (MAX MARKS 10)				
Sl. No.	**Parameters**	**Good**	**Average**	**Needs improvement**
a.	**Understanding of problem (3 marks)**	Clear understanding of problem statement while designing and implementing the program (3)	Problem statement is understood clearly but few mistakes while designing and implementing program (2)	Problem statement is not clearly understood while designing the program(1)
b.	**Writing program (4 marks)**	Program handles all possible conditions (4)	Average condition is defined and verified. (3)	Program does not handle possible conditions (1)
c.	**Result and documentation (3 marks)**	Meticulous documentation and all conditions are taken care (3)	Acceptable documentation shown (2)	Documentation does not take care all conditions (1)

R2: REGULAR LAB EVALUATION VIVA RUBRIC (MAX MARKS 10)					
Sl. No.	**Parameter**	**Excellent**	**Good**	**Average**	**Needs Improvement**
a.	**Conceptual understanding (10 marks)**	Answers 80% of the viva questions asked (10)	Answers 60% of the viva questions asked (7)	Answers 30% of the viva questions asked (4)	Unable to relate the concepts (1)

R3: REGULAR LAB PROGRAM EXECUTION RUBRIC (MAX MARKS 10)				
Sl. No.	**Parameters**	**Excellent**	**Good**	**Needs Improvement**
a.	**Design, implementation, and demonstration (5 marks)**	Program follows syntax and semantics of C programming language. Demonstrates the complete knowledge of the program written (5)	Program has few logical errors, moderately demonstrates all possible concepts implemented in programs (3)	Syntax and semanticsof C programming is not clear (1)
b.	**Result and documentation (5 marks)**	All test cases are successful, all errors are debugged with own practical knowledge and clear documentation according to the guidelines (5)	Moderately debugs the programs , few test case are unsuccessful and Partial documentation (3)	Test cases are not taken care , unable to debugthe errors and no proper documentation (1)

R4: RECORD EVALUATION RUBRIC (MAX MARKS 20)					
Sl. No.	**Parameter**	**Excellent**	**Good**	**Average**	**Needs Improvement**
a.	**Documentation (20 marks)**	Meticulous record writing including program, comments and test cases as per the guidelines mentioned (20)	Write up contains program and test cases, but comments are not included (18)	Write up contains only program(15)	Program written with few mistakes (10)

TEST /LAB INTERNALS MARKS (MAX MARKS 20)

TEST #	Write up 6	Execution 28	Viva 6	Sign	Total 40	Avg. 40	Final 20
TEST-1							
TEST-2						40	20

REGULAR LAB EVALUATION (MAX MARKS 10)

Lab program	Date of Execution	Additional programs	Write up (10)	Exen. (10)	Viva (10)	Total 30	Teacher Signature
1							
2							
3							
4							
5							
6							
7							
8							
9							
10							
11							
12							
Total Marks		360	30			10	

Final Marks obtained from **A. Test (20) +** **B. Regular Evaluation (10) +** **C. Record and Observation (20)**	50	**Lab in charge:** **HOD:**

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

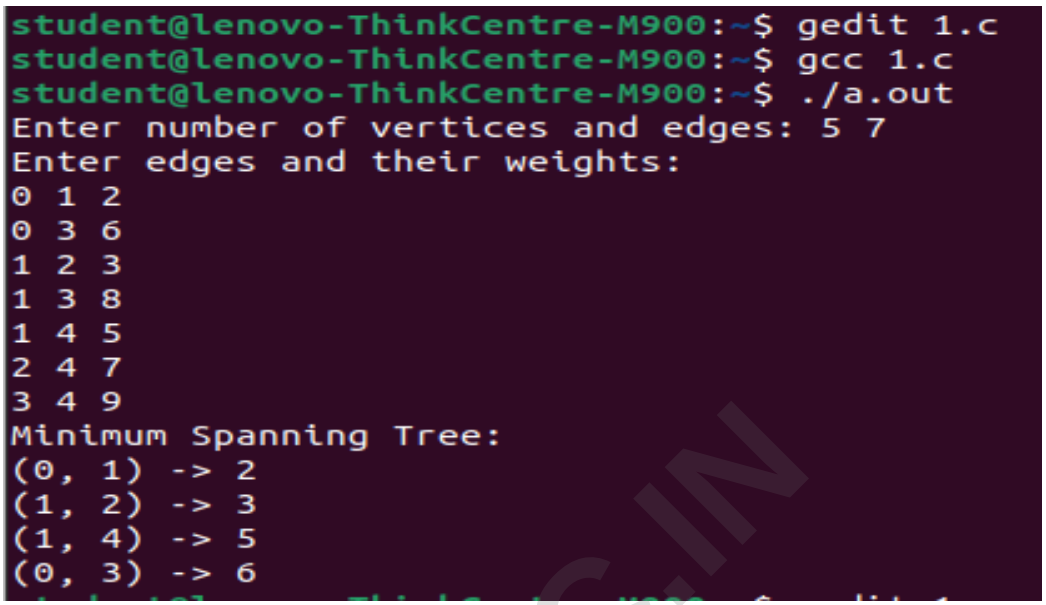
PROGRAM:

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

#define MAX_EDGES 1000

typedef struct Edge {
    int src, dest, weight;
} Edge;

typedef struct Graph {
    int V, E;
    Edge edges[MAX_EDGES];
} Graph;

typedef struct Subset {
    int parent, rank;
} Subset;

Graph* createGraph(int V, int E) {
    Graph* graph = (Graph*) malloc(sizeof(Graph));
    graph->V = V;
    graph->E = E;
    return graph;
}

int find(Subset subsets[], int i) {
    if (subsets[i].parent != i) {
        subsets[i].parent = find(subsets, subsets[i].parent);
    }
    return subsets[i].parent;
}

void Union(Subset subsets[], int x, int y) {
    int xroot = find(subsets, x);
    int yroot = find(subsets, y);

    if (subsets[xroot].rank < subsets[yroot].rank) {
        subsets[xroot].parent = yroot;
    } else if (subsets[xroot].rank > subsets[yroot].rank) {
        subsets[yroot].parent = xroot;
    } else {
        subsets[yroot].parent = xroot;
        subsets[xroot].rank++;
    }
}
```

```
int compare(const void* a, const void* b) {
    Edge* a_edge = (Edge*) a;
    Edge* b_edge = (Edge*) b;
    return a_edge->weight - b_edge->weight;
}

void kruskalMST(Graph* graph) {
    Edge mst[graph->V];
    int e = 0, i = 0;

    qsort(graph->edges, graph->E, sizeof(Edge), compare);

    Subset* subsets = (Subset*) malloc(graph->V * sizeof(Subset));
    for (int v = 0; v < graph->V; ++v) {
        subsets[v].parent = v;
        subsets[v].rank = 0;
    }

    while (e < graph->V - 1 && i < graph->E) {
        Edge next_edge = graph->edges[i++];

        int x = find(subsets, next_edge.src);
        int y = find(subsets, next_edge.dest);

        if (x != y) {
            mst[e++] = next_edge;
            Union(subsets, x, y);
        }
    }

    printf("Minimum Spanning Tree:\n");
    for (i = 0; i < e; ++i) {
        printf("(%d, %d) -> %d\n", mst[i].src, mst[i].dest, mst[i].weight);
    }
}

int main() {
    int V, E;
    printf("Enter number of vertices and edges: ");
    scanf("%d %d", &V, &E);

    Graph* graph = createGraph(V, E);

    printf("Enter edges and their weights:\n");
    for (int i = 0; i < E; ++i) {
        scanf("%d %d %d", &graph->edges[i].src, &graph->edges[i].dest,
&graph->edges[i].weight);
    }

    kruskalMST(graph);
```

```
    return 0;
}
```

OUTPUT:



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

PROGRAM:

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

#define V_MAX 100 // Maximum number of vertices

// Function to find the vertex with the minimum key value, from the set of vertices not yet included in the MST
int minKey(int key[], int mstSet[], int V) {
    int min = INT_MAX, min_index;

    for (int v = 0; v < V; v++)
        if (mstSet[v] == 0 && key[v] < min)
            min = key[v], min_index = v;

    return min_index;
}

// Function to print the constructed MST stored in parent[]
void printMST(int parent[], int n, int graph[V_MAX][V_MAX], int V) {
    printf("Edge   Weight\n");
    for (int i = 1; i < V; i++)
        printf("%d - %d    %d \n", parent[i], i, graph[i][parent[i]]);
}

// Function to construct and print MST for a graph represented using adjacency matrix representation
void primMST(int graph[][V_MAX], int V) {
    int parent[V_MAX]; // Array to store constructed MST
    int key[V_MAX]; // Key values used to pick minimum weight edge in cut
    int mstSet[V_MAX]; // To represent set of vertices not yet included in MST

    // Initialize all keys as INFINITE, mstSet[] as 0
    for (int i = 0; i < V; i++)
        key[i] = INT_MAX, mstSet[i] = 0;

    // Always include first 1st vertex in MST. Make key 0 so that this vertex is picked as the first vertex
    key[0] = 0;
    parent[0] = -1; // First node is always the root of MST

    // The MST will have V vertices
    for (int count = 0; count < V - 1; count++) {
        // Pick the minimum key vertex from the set of vertices not yet included in MST
        int u = minKey(key, mstSet, V);
```

```
        // Add the picked vertex to the MST set
        mstSet[u] = 1;

        // Update key value and parent index of the adjacent vertices of the
picked vertex
        // Consider only those vertices which are not yet included in the MST
        for (int v = 0; v < V; v++)
            if (graph[u][v] && mstSet[v] == 0 && graph[u][v] < key[v])
                parent[v] = u, key[v] = graph[u][v];
    }

    // Print the constructed MST
    printMST(parent, V, graph, V);
}

int main() {
    int V, E;
    printf("Enter the number of vertices and edges: ");
    scanf("%d %d", &V, &E);

    // Create the graph as an adjacency matrix
    int graph[V_MAX][V_MAX];
    for (int i = 0; i < V; i++) {
        for (int j = 0; j < V; j++) {
            graph[i][j] = 0; // Initialize the graph with 0s
        }
    }

    // Prompt the user to enter the source vertex, destination vertex, and
weight for each edge
    printf("Enter the source vertex, destination vertex, and weight for each
edge:\n");
    for (int i = 0; i < E; i++) {
        int source, dest, weight;
        scanf("%d %d %d", &source, &dest, &weight);
        graph[source][dest] = weight;
        graph[dest][source] = weight; // Since the graph is undirected
    }

    // Print the MST using Prim's algorithm
    primMST(graph, V);

    return 0;
}
```

OUTPUT:

```
student@lenovo-ThinkCentre-M900:~$ gedit 2.c
student@lenovo-ThinkCentre-M900:~$ gcc 2.c
student@lenovo-ThinkCentre-M900:~$ ./a.out
Enter the number of vertices and edges: 5
7
Enter the source vertex, destination vertex, and weight for each edge:
0 1 2
0 3 6
1 2 3
1 3 8
1 4 5
2 4 7
3 4 9
Edge   Weight
0 - 1    2
1 - 2    3
0 - 3    6
1 - 4    5
```

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

PROGRAM:

```
#include<stdio.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 Transitive closure:\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]);
        }

    }
```

OUTPUT:

```
student@lenovo-ThinkCentre-M900:~$ gcc 3a.c
student@lenovo-ThinkCentre-M900:~$ ./a.out

 Enter the number of vertices:4

 Enter the number of edges:
5

 Enter the end vertices of edge1 with its weight
1 3 3

 Enter the end vertices of edge2 with its weight
2 1 2

 Enter the end vertices of edge3 with its weight
3 2 7

 Enter the end vertices of edge4 with its weight
3 4 1

 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

 Transitive closure:
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

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

PROGRAM:

```
#include<stdio.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");
        }

    }
```

OUTPUT:

```
student@lenovo-ThinkCentre-M900:~$ gedit 3b.c
student@lenovo-ThinkCentre-M900:~$ gcc 3b.c
student@lenovo-ThinkCentre-M900:~$ ./a.out

 Enter the number of vertices:5

 Enter the number of edges:11
```

```
Enter the end vertices of edge 1:1 1

Enter the end vertices of edge 2:1 4

Enter the end vertices of edge 3:3 2

Enter the end vertices of edge 4:3 3

Enter the end vertices of edge 5:3 4

Enter the end vertices of edge 6:4 2

Enter the end vertices of edge 7:4 4

Enter the end vertices of edge 8:5 2

Enter the end vertices of edge 9:5 3

Enter the end vertices of edge 10:5 4

Enter the end vertices of edge 11:5 5

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

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

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

PROGRAM:

```
  #include <stdio.h>
#include <stdbool.h>
#include <limits.h>

#define MAX_VERTICES 10  // Maximum number of vertices
#define INF INT_MAX

// A function to find the vertex with the minimum distance value, from the
set of vertices not yet included in the shortest path tree
int minDistance(int dist[], bool sptSet[], int V) {
    int min = INF, min_index;

    for (int v = 0; v < V; v++)
        if (sptSet[v] == false && dist[v] <= min)
            min = dist[v], min_index = v;

    return min_index;
}

// A utility function to print the constructed distance array
void printSolution(int dist[], int V) {
    printf("Vertex \t\t Distance from Source\n");
    for (int i = 0; i < V; i++)
        printf("%d \t\t %d\n", i, dist[i]);
}

// Dijkstra's algorithm for adjacency matrix representation of the graph
void dijkstra(int graph[MAX_VERTICES][MAX_VERTICES], int src, int V) {
    int dist[MAX_VERTICES];  // The output array. dist[i] will hold the
shortest distance from src to i
    bool sptSet[MAX_VERTICES];  // sptSet[i] will be true if vertex i is
included in the shortest path tree

    // Initialize all distances as INFINITE and sptSet[] as false
    for (int i = 0; i < V; i++)
        dist[i] = INF, sptSet[i] = false;

    dist[src] = 0;

    // Find shortest path for all vertices
    for (int count = 0; count < V - 1; count++) {
        int u = minDistance(dist, sptSet, V);
        sptSet[u] = true;
        for (int v = 0; v < V; v++)
```

```
            if (!sptSet[v] && graph[u][v] && dist[u] != INF && dist[u] + graph[u][v]
< dist[v])
                dist[v] = dist[u] + graph[u][v];
    }

    printSolution(dist, V);
}

// Driver code
int main() {
    int V, E;
    printf("Enter the number of vertices: ");
    scanf("%d", &V);
    printf("Enter the number of edges: ");
    scanf("%d", &E);

    int graph[MAX_VERTICES][MAX_VERTICES] = {{0}};

    printf("Enter the source vertex, destination vertex, and weight for each
edge:\n");
    for (int i = 0; i < E; i++) {
        int source, dest, weight;
        scanf("%d %d %d", &source, &dest, &weight);
        graph[source][dest] = weight;
        graph[dest][source] = weight; // Assuming undirected graph
    }

    dijkstra(graph, 0, V);
    return 0;
}
```

OUTPUT:

```
student@lenovo-ThinkCentre-M900:~$ gedit 4.c
student@lenovo-ThinkCentre-M900:~$ gcc 4.c
student@lenovo-ThinkCentre-M900:~$ ./a.out
Enter the number of vertices: 5
Enter the number of edges: 7
Enter the source vertex, destination vertex, and weight for each edge:
0 1 2
0 3 6
1 2 3
1 3 8
1 4 5
2 4 7
3 4 9
Vertex          Distance from Source
0               0
1               2
2               5
3               6
4               7
```

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

PROGRAM:

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

#define MAX_VERTICES 100

// Structure to represent a graph
typedef struct {
    int V;
    int** adjMatrix;
} Graph;

// Function to create a new graph
Graph* createGraph(int V) {
    Graph* graph = (Graph*)malloc(sizeof(Graph));
    graph->V = V;
    graph->adjMatrix = (int**)calloc(V, sizeof(int*));
    for (int i = 0; i < V; i++) graph->adjMatrix[i] = (int*)calloc(V, sizeof(int));
    return graph;
}

// Function to add an edge to the graph
void addEdge(Graph* graph, int src, int dest) {
    graph->adjMatrix[src][dest] = 1;
}

// Function to perform topological sorting
void topologicalSort(Graph* graph) {
    int V = graph->V, inDegree[MAX_VERTICES] = {0},
queue[MAX_VERTICES], front = 0, rear = -1;

    for (int i = 0; i < V; i++)
        for (int j = 0; j < V; j++)
            if (graph->adjMatrix[i][j] == 1) inDegree[j]++;

    for (int i = 0; i < V; i++) if (inDegree[i] == 0) queue[++rear] = i;

    printf("Topological ordering of vertices: ");
    while (front <= rear) {
        int vertex = queue[front++];
        printf("%d ", vertex);
        for (int i = 0; i < V; i++) if (graph->adjMatrix[vertex][i] == 1 && --
inDegree[i] == 0) queue[++rear] = i;
    }
    printf("\n");
```

```
}

// Driver code
int main() {
    int V, E;
    printf("Enter the number of vertices: ");
    scanf("%d", &V);
    Graph* graph = createGraph(V);
    printf("Enter the number of edges: ");
    scanf("%d", &E);
    printf("Enter the edges (source vertex, destination vertex):\n");
    for (int i = 0, src, dest; i < E; i++) {
        scanf("%d %d", &src, &dest);
        addEdge(graph, src, dest);
    }
    topologicalSort(graph);
    return 0;
}
```

OUTPUT:

```
student@lenovo-ThinkCentre-M900:~$ gcc 5.c
student@lenovo-ThinkCentre-M900:~$ ./a.out
Enter the number of vertices: 7
Enter the number of edges: 8
Enter the edges (source vertex, destination vertex):
0 1
0 2
1 3
2 3
3 4
3 5
4 6
5 6
Topological ordering of vertices: 0 1 2 3 4 5 6
```

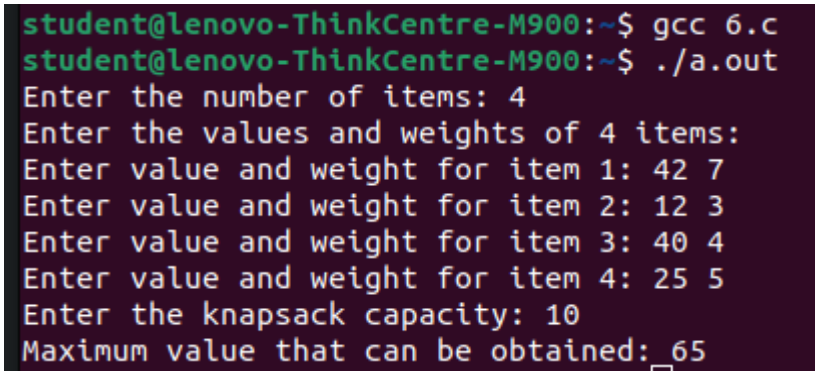
6.Design and implement C Program to solve 0/1 Knapsack problem using Dynamic Programming method.

PROGRAM:

```
#include <stdio.h>

// Function to find maximum of two integers
int max(int a, int b) {
    return (a > b) ? a : b;
}

// Function to solve 0/1 Knapsack problem
int knapsack(int W, int wt[], int val[], int n) {
    int i, w;
    int K[n + 1][W + 1];

    // Build table K[][] in bottom-up manner
    for (i = 0; i <= n; i++) {
        for (w = 0; w <= W; w++) {
            if (i == 0 || w == 0)
                K[i][w] = 0;
            else if (wt[i - 1] <= w)
                K[i][w] = max(val[i - 1] + K[i - 1][w - wt[i - 1]], K[i - 1][w]);
            else
                K[i][w] = K[i - 1][w];
        }
    }

    // K[n][W] contains the maximum value that can be put in a knapsack of capacity W
    return K[n][W];
}

int main() {
    int val[100], wt[100]; // Arrays to store values and weights
    int W, n; // Knapsack capacity and number of items
    printf("Enter the number of items: ");
    scanf("%d", &n);

    printf("Enter the values and weights of %d items:\n", n);
    for (int i = 0; i < n; i++) {
        printf("Enter value and weight for item %d: ", i + 1);
        scanf("%d %d", &val[i], &wt[i]);
    }

    printf("Enter the knapsack capacity: ");
    scanf("%d", &W);

    printf("Maximum value that can be obtained: %d\n", knapsack(W, wt, val, n));
```

```
    return 0;
}
```

OUTPUT:



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

PROGRAM:

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

// Structure to represent items
struct Item {
    int value;
    int weight;
    double ratio; // Value-to-weight ratio for sorting
};

// Comparison function for sorting items based on ratio in descending order
int compare(const void *a, const void *b) {
    struct Item *item1 = (struct Item *)a;
    struct Item *item2 = (struct Item *)b;
    double ratio1 = item1->ratio;
    double ratio2 = item2->ratio;
    if (ratio1 > ratio2) return -1;
    else if (ratio1 < ratio2) return 1;
    else return 0;
}

// Function to solve discrete Knapsack problem
void discreteKnapsack(struct Item items[], int n, int capacity) {
    int i, j;
    int dp[n + 1][capacity + 1];

    // Initialize the DP table
    for (i = 0; i <= n; i++) {
        for (j = 0; j <= capacity; j++) {
            if (i == 0 || j == 0)
                dp[i][j] = 0;
            else if (items[i - 1].weight <= j)
                dp[i][j] = (items[i - 1].value + dp[i - 1][j - items[i - 1].weight] > dp[i -
1][j]) ?
                            (items[i - 1].value + dp[i - 1][j - items[i - 1].weight]) :
                            dp[i - 1][j];
            else
                dp[i][j] = dp[i - 1][j];
        }
    }

    printf("Total value obtained for discrete knapsack: %d\n", dp[n][capacity]);
}

// Function to solve continuous Knapsack problem
void continuousKnapsack(struct Item items[], int n, int capacity) {
```

```
    int i;
    double totalValue = 0.0;
    int remainingCapacity = capacity;

    for (i = 0; i < n; i++) {
        if (remainingCapacity >= items[i].weight) {
            totalValue += items[i].value;
            remainingCapacity -= items[i].weight;
        } else {
            totalValue += (double)remainingCapacity / items[i].weight *
items[i].value;
            break;
        }
    }

    printf("Total value obtained for continuous knapsack: %.2lf\n",
totalValue);
}

int main() {
    int n, capacity, i;
    printf("Enter the number of items: ");
    scanf("%d", &n);

    struct Item items[n];

    printf("Enter the capacity of the knapsack: ");
    scanf("%d", &capacity);

    printf("Enter the value and weight of each item:\n");
    for (i = 0; i < n; i++) {
        scanf("%d %d", &items[i].value, &items[i].weight);
        items[i].ratio = (double)items[i].value / items[i].weight;
    }

    // Sort items based on value-to-weight ratio
    qsort(items, n, sizeof(struct Item), compare);

    discreteKnapsack(items, n, capacity);
    continuousKnapsack(items, n, capacity);

    return 0;
}
```

OUTPUT:

```
student@lenovo-ThinkCentre-M900:~$ ./a.out
Enter the number of items: 4
Enter the capacity of the knapsack: 10
Enter the value and weight of each item:
42 7
12 3
40 4
25 5
Total value obtained for discrete knapsack: 65
Total value obtained for continuous knapsack: 76.00
```

8.Design and implement 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.

PROGRAM:

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

#define MAX_SIZE 100

// Function to find subset with given sum
void subsetSum(int set[], int subset[], int n, int subSize, int total, int
nodeCount, int sum) {
    if (total == sum) {
        // Print the subset
        printf("Subset found: { ");
        for (int i = 0; i < subSize; i++) {
            printf("%d ", subset[i]);
        }
        printf("}\n");
        return;
    } else {
        // Check the sum of the remaining elements
        for (int i = nodeCount; i < n; i++) {
            subset[subSize] = set[i];
            subsetSum(set, subset, n, subSize + 1, total + set[i], i + 1, sum);
        }
    }
}

int main() {
```

```
    int set[MAX_SIZE];
    int subset[MAX_SIZE];
    int n, sum;

    // Input the number of elements in the set
    printf("Enter the number of elements in the set: ");
    scanf("%d", &n);

    // Input the elements of the set
    printf("Enter the elements of the set:\n");
    for (int i = 0; i < n; i++) {
        scanf("%d", &set[i]);
    }

    // Input the target sum
    printf("Enter the sum to find subset for: ");
    scanf("%d", &sum);

    printf("Subsets with sum %d:\n", sum);
    subsetSum(set, subset, n, 0, 0, 0, sum);

    return 0;
}
```

OUTPUT:

```
student@lenovo-ThinkCentre-M900:~$ gcc program8.c
student@lenovo-ThinkCentre-M900:~$ ./a.out
Enter the number of elements in the set: 5
Enter the elements of the set:
2
4
6
8
10
Enter the sum to find subset for: 10
Subsets with sum 10:
Subset found: { 2 8 }
Subset found: { 4 6 }
Subset found: { 10 }
```

9.Design and implement 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.

PROGRAM:

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

// Function to perform Selection Sort
void selectionSort(int arr[], int n) {
    int i, j, minIndex, temp;
    for (i = 0; i < n - 1; i++) {
        minIndex = i;
        for (j = i + 1; j < n; j++) {
            if (arr[j] < arr[minIndex]) {
                minIndex = j;
            }
        }
        // Swap the found minimum element with the first element
        temp = arr[minIndex];
        arr[minIndex] = arr[i];
        arr[i] = temp;
    }
}

int main() {
    int n, i;
    clock_t start, end;
    double cpu_time_used;

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

    if (n < 5000) {
        printf("Please enter a value of n greater than 5000.\n");
        return 1;
    }

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

    // Generate n random numbers
    srand(time(NULL));
```

```
    // printf("Randomly generated array: ");
    for (i = 0; i < n; i++) {
        arr[i] = rand() % 10000; // Generating random numbers between 0 to
9999
        // printf("%d ", arr[i]);
    }
    // printf("\n");

    // Record the starting time
    start = clock();

    // Perform Selection Sort
    selectionSort(arr, n);

    // Record the ending time
    end = clock();

    // Calculate the time taken
    cpu_time_used = ((double)(end - start)) / CLOCKS_PER_SEC;

    // printf("Sorted array: ");
    // for (i = 0; i < n; i++) {
    //     printf("%d ", arr[i]);
    // }
    // printf("\n");

    printf("Time taken for sorting: %lf seconds\n", cpu_time_used);

    free(arr); // Free dynamically allocated memory
    return 0;
}
```

OUTPUT:

```
student@lenovo-ThinkCentre-M900:~$ gcc program9.c
student@lenovo-ThinkCentre-M900:~$ ./a.out
Enter the number of elements: 5000
Time taken to sort 5000 elements: 0.028919 seconds
student@lenovo-ThinkCentre-M900:~$ ./a.out
Enter the number of elements: 10000
Time taken to sort 10000 elements: 0.112973 seconds
student@lenovo-ThinkCentre-M900:~$ ./a.out
Enter the number of elements: 15000
Time taken to sort 15000 elements: 0.250916 seconds
student@lenovo-ThinkCentre-M900:~$ ./a.out
Enter the number of elements: 20000
Time taken to sort 20000 elements: 0.447036 seconds
student@lenovo-ThinkCentre-M900:~$ ./a.out
Enter the number of elements: 25000
Time taken to sort 25000 elements: 0.693559 seconds
```

10.Design and implement 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.

PROGRAM:

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

// Function to partition the array and return the pivot index
int partition(int arr[], int low, int high) {
    int pivot = arr[high]; // Pivot element
    int i = (low - 1);     // Index of smaller element

    for (int j = low; j <= high - 1; j++) {
        // If current element is smaller than or equal to pivot
        if (arr[j] <= pivot) {
            i++; // Increment index of smaller element
            // Swap arr[i] and arr[j]
            int temp = arr[i];
            arr[i] = arr[j];
            arr[j] = temp;
        }
    }
    // Swap arr[i + 1] and arr[high] (or pivot)
    int temp = arr[i + 1];
    arr[i + 1] = arr[high];
    arr[high] = temp;
    return (i + 1);
}

// Function to implement Quick Sort
void quickSort(int arr[], int low, int high) {
    if (low < high) {
        // pi is partitioning index, arr[pi] is now at right place
        int pi = partition(arr, low, high);

        // Separately sort elements before partition and after partition
        quickSort(arr, low, pi - 1);
        quickSort(arr, pi + 1, high);
    }
}

int main() {
    int n, i;
    clock_t start, end;
    double cpu_time_used;
```

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

    if (n < 5000) {
        printf("Please enter a value of n greater than 5000.\n");
        return 1;
    }

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

    // Generate n random numbers
    srand(time(NULL));
    // printf("Randomly generated array: ");
    for (i = 0; i < n; i++) {
        arr[i] = rand() % 10000; // Generating random numbers between 0 to
9999
        // printf("%d ", arr[i]);
    }
    // printf("\n");

    // Record the starting time
    start = clock();

    // Perform Quick Sort
    quickSort(arr, 0, n - 1);

    // Record the ending time
    end = clock();

    // Calculate the time taken
    cpu_time_used = ((double)(end - start)) / CLOCKS_PER_SEC;

    // printf("Sorted array: ");
    // for (i = 0; i < n; i++) {
    //     printf("%d ", arr[i]);
    // }
    // printf("\n");

    printf("Time taken for sorting: %lf seconds\n", cpu_time_used);

    free(arr); // Free dynamically allocated memory
    return 0;
}
```

OUTPUT:

```
student@lenovo-ThinkCentre-M900:~$ gcc 10.c
student@lenovo-ThinkCentre-M900:~$ ./a.out
Enter the number of elements: 5000
Time taken to sort 5000 elements: 0.000557 seconds
student@lenovo-ThinkCentre-M900:~$ gcc 10.c
student@lenovo-ThinkCentre-M900:~$ ./a.out
Enter the number of elements: 10000
Time taken to sort 10000 elements: 0.001171 seconds
student@lenovo-ThinkCentre-M900:~$ gcc 10.c
student@lenovo-ThinkCentre-M900:~$ ./a.out
Enter the number of elements: 15000
Time taken to sort 15000 elements: 0.001912 seconds
student@lenovo-ThinkCentre-M900:~$ gcc 10.c
student@lenovo-ThinkCentre-M900:~$ ./a.out
Enter the number of elements: 20000
Time taken to sort 20000 elements: 0.002697 seconds
student@lenovo-ThinkCentre-M900:~$ gcc 10.c
student@lenovo-ThinkCentre-M900:~$ ./a.out
Enter the number of elements: 25000
Time taken to sort 25000 elements: 0.003862 seconds
```

11.Design and implement 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.

PROGRAM:

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

// Merge two subarrays of arr[].
// First subarray is arr[l..m]
// Second subarray is arr[m+1..r]
void merge(int arr[], int l, int m, int r) {
    int i, j, k;
    int n1 = m - l + 1;
    int n2 = r - m;

    // Create temporary arrays
    int L[n1], R[n2];

    // Copy data to temporary arrays L[] and R[]
    for (i = 0; i < n1; i++)
        L[i] = arr[l + i];
    for (j = 0; j < n2; j++)
        R[j] = arr[m + 1 + j];

    // Merge the temporary arrays back into arr[l..r]
    i = 0;
    j = 0;
    k = l;
    while (i < n1 && j < n2) {
        if (L[i] <= R[j]) {
            arr[k] = L[i];
            i++;
        } else {
            arr[k] = R[j];
            j++;
        }
        k++;
    }

    // Copy the remaining elements of L[], if any
    while (i < n1) {
        arr[k] = L[i];
        i++;
        k++;
    }
```

```
    // Copy the remaining elements of R[], if any
    while (j < n2) {
        arr[k] = R[j];
        j++;
        k++;
    }
}

// Merge Sort function
void mergeSort(int arr[], int l, int r) {
    if (l < r) {
        // Same as (l+r)/2, but avoids overflow for large l and r
        int m = l + (r - l) / 2;

        // Sort first and second halves
        mergeSort(arr, l, m);
        mergeSort(arr, m + 1, r);

        // Merge the sorted halves
        merge(arr, l, m, r);
    }
}

int main() {
    int n, i;
    clock_t start, end;
    double cpu_time_used;

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

    if (n < 5000) {
        printf("Please enter a value of n greater than 5000.\n");
        return 1;
    }

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

    // Generate n random numbers
    srand(time(NULL));
    // printf("Randomly generated array: ");
    for (i = 0; i < n; i++) {
        arr[i] = rand() % 10000; // Generating random numbers between 0 to
9999
        // printf("%d ", arr[i]);
```

```
    }
    // printf("\n");

    // Record the starting time
    start = clock();

    // Perform Merge Sort
    mergeSort(arr, 0, n - 1);

    // Record the ending time
    end = clock();

    // Calculate the time taken
    cpu_time_used = ((double)(end - start)) / CLOCKS_PER_SEC;

    // printf("Sorted array: ");
    // for (i = 0; i < n; i++) {
    //     printf("%d ", arr[i]);
    // }
    // printf("\n");

    printf("Time taken for sorting: %lf seconds\n", cpu_time_used);

    free(arr); // Free dynamically allocated memory
    return 0;
}
```

OUTPUT:

```
student@lenovo-ThinkCentre-M900:~$ gcc 11.c
student@lenovo-ThinkCentre-M900:~$ ./a.out
Enter the number of elements: 5000
Time taken to sort 5000 elements: 0.000691 seconds
student@lenovo-ThinkCentre-M900:~$ gcc 11.c
student@lenovo-ThinkCentre-M900:~$ ./a.out
Enter the number of elements: 10000
Time taken to sort 10000 elements: 0.001521 seconds
student@lenovo-ThinkCentre-M900:~$ gcc 11.c
student@lenovo-ThinkCentre-M900:~$ ./a.out
Enter the number of elements: 15000
Time taken to sort 15000 elements: 0.002262 seconds
student@lenovo-ThinkCentre-M900:~$ gcc 11.c
student@lenovo-ThinkCentre-M900:~$ ./a.out
Enter the number of elements: 20000
Time taken to sort 20000 elements: 0.003134 seconds
student@lenovo-ThinkCentre-M900:~$ gcc 11.c
student@lenovo-ThinkCentre-M900:~$ ./a.out
Enter the number of elements: 25000
Time taken to sort 25000 elements: 0.003956 seconds
```

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

PROGRAM:

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

int main()
{
int n,i,j;
void queen(int row,int n);

printf(" - N Queens Problem Using Backtracking -");
printf("\n\nEnter number of Queens:");
scanf("%d",&n);
queen(1,n);
return 0;
}

//function for printing the solution
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) //for nxn board
  {
   if(board[i]==j)
    printf("\tQ"); //queen at i,j position
   else
    printf("\t-"); //empty slot
  }
}
}

/*funtion to check conflicts
If no conflict for desired postion returns 1 otherwise returns 0*/
int place(int row,int column)
{
int i;
for(i=1;i<=row-1;++i)
{
```

```
  //checking column and digonal conflicts
  if(board[i]==column)
   return 0;
  else
   if(abs(board[i]-column)==abs(i-row))
    return 0;
}

return 1; //no conflicts
}

//function to check for proper positioning of queen
void queen(int row,int n)
{
int column;
for(column=1;column<=n;++column)
{
  if(place(row,column))
  {
   board[row]=column; //no conflicts so place queen
   if(row==n) //dead end
    print(n); //printing the board configuration
   else //try queen with next position
    queen(row+1,n);
  }
}
}
```

OUTPUT:

```
student@lenovo-ThinkCentre-M900:~$ gedit 12.c
student@lenovo-ThinkCentre-M900:~$ gcc 12.c
student@lenovo-ThinkCentre-M900:~$ ./a.out
 - N Queens Problem Using Backtracking -

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       -       -student@lenov
```