VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT

on

Analysis and Design of Algorithms

Submitted by

YATHARTH SINGH (1WA23CS051)

in partial fulfillment for the award of the degree of

BACHELOR OF ENGINEERING

in

COMPUTER SCIENCE AND ENGINEERING

B.M.S. COLLEGE OF ENGINEERING

(Autonomous Institution under VTU)

BENGALURU-560019

Feb-2025 to June-2025

B. M. S. College of Engineering,

Bull Temple Road, Bangalore 560019

(Affiliated To Visvesvaraya Technological University, Belgaum)

Department of Computer Science and Engineering

CERTIFICATE

This is to certify that the Lab work entitled "Analysis and Design of Algorithms – 23CS4PCADA" carried out by Yatharth Singh (1WA23CS051), who is a Bonafide student of B. M. S. College of Engineering. It is in partial fulfilment for the award of Bachelor of Engineering in Computer Science and Engineering of the Visvesvaraya Technological University, Belgaum during the year Feb 2025- June 2025. The Lab report has been approved as it satisfies the academic requirements in respect of a Analysis and Design of Algorithms – (23CS4PCADA) work prescribed for the said degree.

Vikranth B.M	Dr. Kavitha Sooda
Assistant Professor	Professor and Head
Department of CSE	Department of CSE
BMSCE, Bengaluru	BMSCE, Bengaluru

Index Sheet

9.	Implement Fractional Knapsack using Greedy technique.	17-18
10.	From a given vertex in a weighted connected graph, find shortest paths to other vertices using Dijkstra's algorithm.	18-20
11.	Implement "N-Queens Problem" using Backtracking.	20-22

Course Outcomes

C01	Analyze time complexity of recursive and non-recursive algorithms using asymptotic notations
C02	Apply various algorithm design techniques for the given problem
C03	Apply the knowledge of complexity classes P, NP, and NP-Complete and prove certain problems are NP-Complete
C04	Design efficient algorithms and conduct practical experiments to solve problems.

Program -1

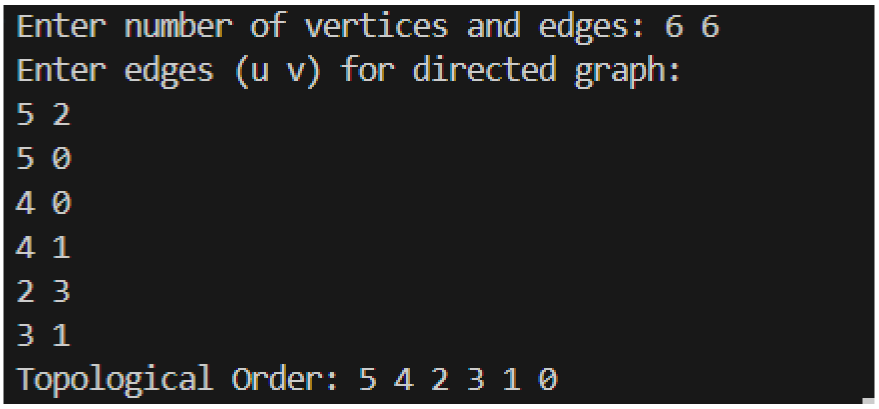
Question:

Write program to obtain the Topological ordering of vertices in a given digraph

Code:

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

#define MAX 100

int graph[MAX][MAX], visited[MAX], stack[MAX], top = -1;
int V;  // number of vertices

void dfs(int v) {
    visited[v] = 1;
    for (int i = 0; i < V; i++) {
        if (graph[v][i] && !visited[i])
            dfs(i);
    }
    stack[++top] = v;
}

void topologicalSort() {
    for (int i = 0; i < V; i++)
        if (!visited[i])
            dfs(i);

    printf("Topological Order: ");
    while (top != -1)
        printf("%d ", stack[top--]);
}

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

    printf("Enter edges (u v) for directed graph:\n");
    for (int i = 0; i < E; i++) {
        scanf("%d %d", &u, &v);
        graph[u][v] = 1;
    }
    topologicalSort();
    return 0;
}
```

Output



```
Enter number of vertices and edges: 6 6
Enter edges (u v) for directed graph:
5 2
5 0
4 0
4 1
2 3
3 1
Topological Order: 5 4 2 3 1 0
```

Program - 2

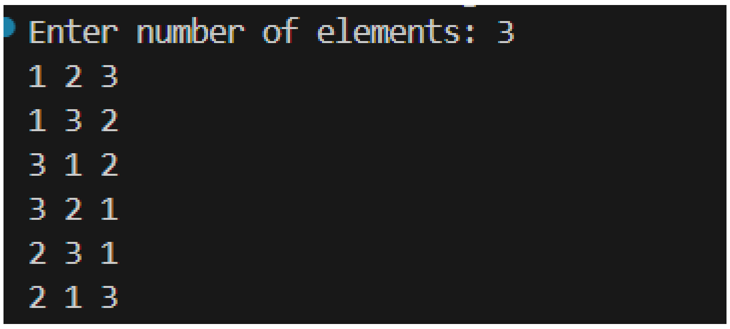
Question:

Implement Johnson Trotter algorithm to generate permutations

Code

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

#define LEFT -1
#define RIGHT 1

int mobile(int a[], int dir[], int n) {
    int max = 0, mob = 0;
    for (int i = 0; i < n; i++) {
        int next = i + dir[i];
        if (next >= 0 && next < n && a[i] > a[next] && a[i] > max)
{
            max = a[i];
            mob = i;
        }
    }
    return mob;
}

void print(int a[], int n) {
    for (int i = 0; i < n; i++) printf("%d ", a[i]);
    printf("\n");
}

void johnsonTrotter(int n) {
    int a[n], dir[n];
    for (int i = 0; i < n; i++) { a[i] = i + 1; dir[i] = LEFT; }

    print(a, n);

    while (1) {
        int m = mobile(a, dir, n);
        if (m == 0 && a[m] <= a[m + dir[m]]) break;

        int swapIdx = m + dir[m];
        int temp = a[m]; a[m] = a[swapIdx]; a[swapIdx] = temp;
        temp = dir[m]; dir[m] = dir[swapIdx]; dir[swapIdx] = temp;

        for (int i = 0; i < n; i++)
            if (a[i] > a[swapIdx]) dir[i] *= -1;

        print(a, n);
    }
}

int main() {
    int n;
    printf("Enter number of elements: ");
    scanf("%d", &n);
    johnsonTrotter(n);
    return 0;
}
```

Output



```
Enter number of elements: 3
1 2 3
1 3 2
3 1 2
3 2 1
2 3 1
2 1 3
```

Program - 3

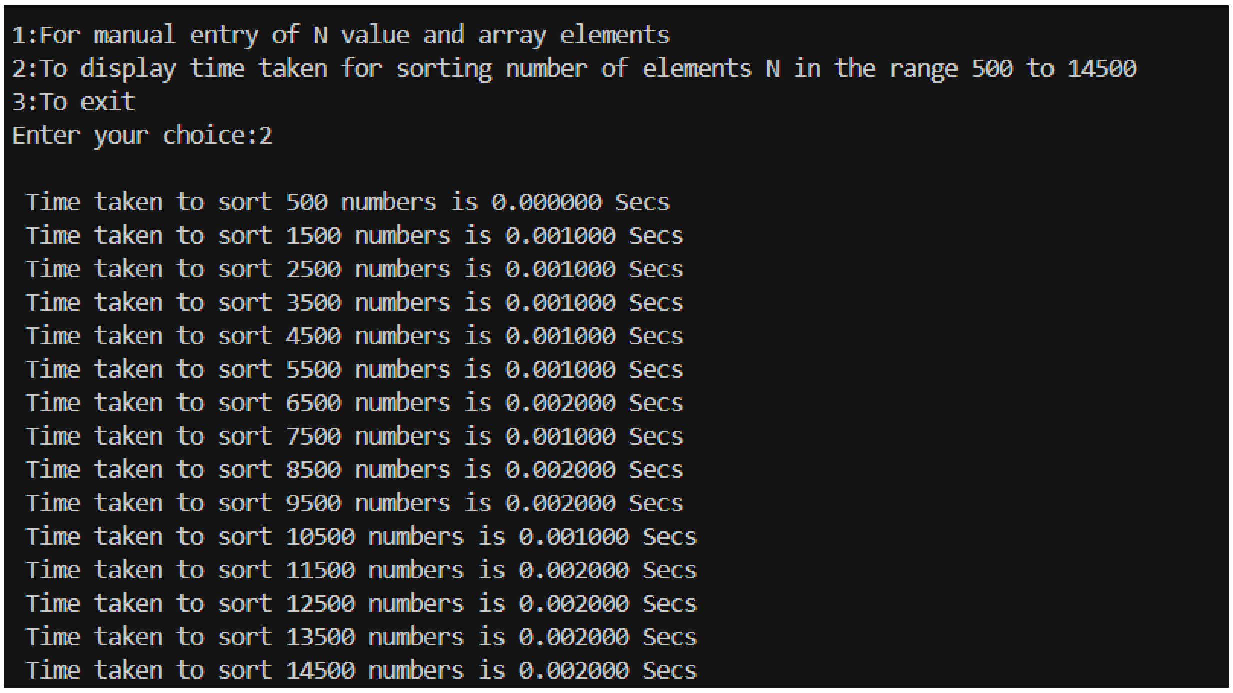
Question:

Sort a given set of N integer elements using Merge Sort technique and compute its time taken. Run the program for different values of N and record the time taken to sort.

Code

```
#include<stdio.h>
#include<time.h>
#include<stdlib.h> /* To recognise exit function when compiling
with gcc*/

void split(int[],int,int);
void combine(int[],int,int,int);
void main()
{
    int a[15000],n, i,j,ch, temp;
    clock_t start,end;
    while(1)
    {
        printf("\n1:For manual entry of N value and array
elements");
        printf("\n2:To display time taken for sorting number of
elements N in the range 500 to 14500");
        printf("\n3:To exit");
        printf("\nEnter your choice:");
        scanf("%d", &ch);
        switch(ch)
        {
            case 1: printf("\nEnter the number of elements: ");
                    scanf("%d",&n);
                    printf("\nEnter array elements: ");
                    for(i=0;i<n;i++)
                    {
                        scanf("%d",&a[i]);
                    }
                    start=clock();
                    split(a,0,n-1);
                    end=clock();
                    printf("\nSorted array is: ");
                    for(i=0;i<n;i++)
                    printf("%d\t",a[i]);
                    printf("\n Time taken to sort %d numbers is %f
Secs",n, (((double)(end-start))/CLOCKS_PER_SEC));
                    break;
            case 2:
                n=500;
                while(n<=14500) {
                    for(i=0;i<n;i++)
                    {
                        //a[i]=random(1000);
                        a[i]=n-i;
                    }
                    start=clock();
                    split(a,0,n-1);
                    //Dummy loop to create delay
                    for(j=0;j<500000;j++){ temp=38/600;}
                    end=clock();
                    printf("\n Time taken to sort %d numbers is %f
Secs",n, (((double)(end-start))/CLOCKS_PER_SEC));
                    n=n+1000;
                }
                break;
            case 3: exit(0);
        }
        getchar();
    }
}

void split(int a[],int low,int high)
{
    int mid;
    if(low<high)
    {
        mid=(low+high)/2;
        split(a,low,mid);
        split(a,mid+1,high);
        combine(a,low,mid,high);
    }
}
void combine(int a[],int low,int mid,int high)
{
    int c[15000],i,j,k;
    i=k=low;
    j=mid+1;
    while(i<=mid&&j<=high)
    {
        if(a[i]<a[j])
        {
            c[k]=a[i];
            ++k;
            ++i;
        }
        else
        {
            c[k]=a[j];
            ++k;
            ++j;
        }
    }
    if(i>mid)
    {
        while(j<=high)
        {
            c[k]=a[j];
            ++k;
            ++j;
        }
    }
    if(j>high)
    {
        while(i<=mid)
        {
            c[k]=a[i];
            ++k;
            ++i;
        }
    }
    for(i=low;i<=high;i++)
    {
        a[i]=c[i];
    }
}
```

Output


```
1:For manual entry of N value and array elements
2:To display time taken for sorting number of elements N in the range 500 to 14500
3:To exit
Enter your choice:2

 Time taken to sort 500 numbers is 0.000000 Secs
 Time taken to sort 1500 numbers is 0.001000 Secs
 Time taken to sort 2500 numbers is 0.001000 Secs
 Time taken to sort 3500 numbers is 0.001000 Secs
 Time taken to sort 4500 numbers is 0.001000 Secs
 Time taken to sort 5500 numbers is 0.001000 Secs
 Time taken to sort 6500 numbers is 0.002000 Secs
 Time taken to sort 7500 numbers is 0.001000 Secs
 Time taken to sort 8500 numbers is 0.002000 Secs
 Time taken to sort 9500 numbers is 0.002000 Secs
 Time taken to sort 10500 numbers is 0.001000 Secs
 Time taken to sort 11500 numbers is 0.002000 Secs
 Time taken to sort 12500 numbers is 0.002000 Secs
 Time taken to sort 13500 numbers is 0.002000 Secs
 Time taken to sort 14500 numbers is 0.002000 Secs
```

Program - 4

Question:

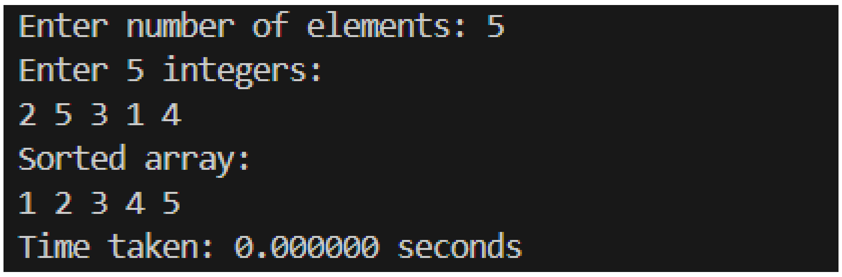
Sort a given set of N integer elements using Quick Sort technique and compute its time taken.

Code

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

void quickSort(int a[], int low, int high) {
    if (low < high) {
        int pivot = a[high], i = low - 1;
        for (int j = low; j < high; j++) {
            if (a[j] < pivot) {
                int t = a[++i]; a[i] = a[j]; a[j] = t;
            }
        }
        int t = a[i + 1]; a[i + 1] = a[high]; a[high] = t;
        int p = i + 1;

        quickSort(a, low, p - 1);
        quickSort(a, p + 1, high);
    }
}
```

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

    int a[n];
    printf("Enter %d integers:\n", n);
    for (int i = 0; i < n; i++)
        scanf("%d", &a[i]);

    clock_t start = clock();
    quickSort(a, 0, n - 1);
    clock_t end = clock();

    printf("Sorted array:\n");
    for (int i = 0; i < n; i++)
        printf("%d ", a[i]);
    printf("\n");

    double time_taken = (double)(end - start) / CLOCKS_PER_SEC;
    printf("Time taken: %f seconds\n", time_taken);

    return 0;
}
```

Code



```
Enter number of elements: 5
Enter 5 integers:
2 5 3 1 4
Sorted array:
1 2 3 4 5
Time taken: 0.000000 seconds
```

Program - 5

Question:

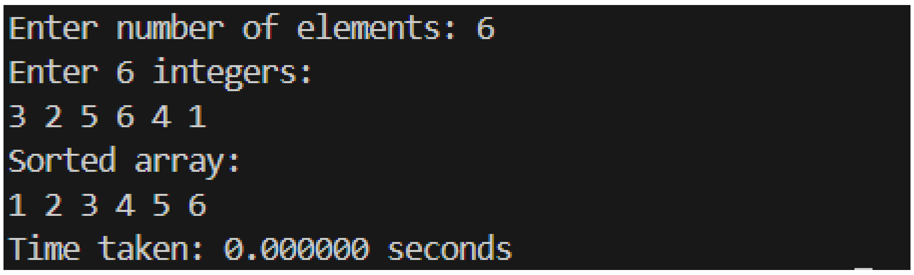
Sort a given set of N integer elements using Heap Sort technique and compute its time taken.

Code

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

```
void heapify(int a[], int n, int i) {
    int largest = i, left = 2*i + 1, right = 2*i + 2;

    if (left < n && a[left] > a[largest])
        largest = left;
    if (right < n && a[right] > a[largest])
        largest = right;

    if (largest != i) {
        int temp = a[i]; a[i] = a[largest]; a[largest] = temp;
        heapify(a, n, largest);
    }
}

void heapSort(int a[], int n) {
    for (int i = n/2 - 1; i >= 0; i--)
        heapify(a, n, i);

    for (int i = n - 1; i > 0; i--) {
        int temp = a[0]; a[0] = a[i]; a[i] = temp;
        heapify(a, i, 0);
    }
}

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

    int a[n];
    printf("Enter %d integers:\n", n);
    for (int i = 0; i < n; i++)
        scanf("%d", &a[i]);

    clock_t start = clock();
    heapSort(a, n);
    clock_t end = clock();

    printf("Sorted array:\n");
    for (int i = 0; i < n; i++)
        printf("%d ", a[i]);
    printf("\n");

    double time_taken = (double)(end - start) / CLOCKS_PER_SEC;
    printf("Time taken: %f seconds\n", time_taken);
    return 0;
}
```

Output



```
Enter number of elements: 6
Enter 6 integers:
3 2 5 6 4 1
Sorted array:
1 2 3 4 5 6
Time taken: 0.000000 seconds
```

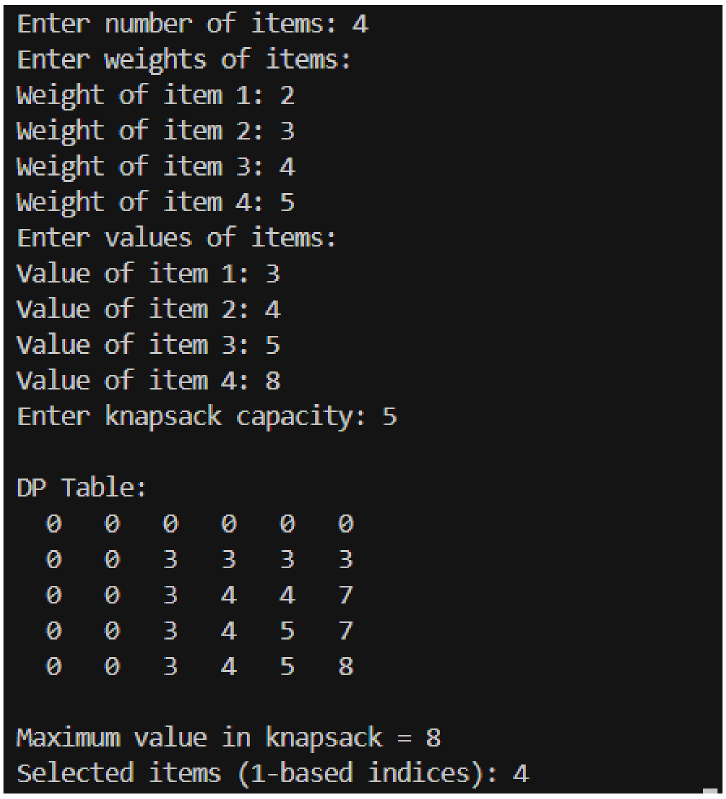
Program - 6

Question:

Implement 0/1 Knapsack problem using dynamic programming.

Code

```
#include <stdio.h>

#define MAX_ITEMS 100
#define MAX_WEIGHT 100

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

void knapsack(int n, int W, int wt[], int val[]) {
    int dp[MAX_ITEMS][MAX_WEIGHT];

    // Fill DP table
    for (int i = 0; i <= n; i++) {
        for (int w = 0; w <= W; w++) {
            if (i == 0 || w == 0)
                dp[i][w] = 0;
            else if (wt[i - 1] <= w)
                dp[i][w] = max(val[i - 1] + dp[i - 1][w - wt[i -
1]], dp[i - 1][w]);
            else
                dp[i][w] = dp[i - 1][w];
        }
    }
    printf("\nDP Table:\n");
    for (int i = 0; i <= n; i++) {
        for (int w = 0; w <= W; w++) {
            printf("%3d ", dp[i][w]);
        }
        printf("\n");
    }
    printf("\nMaximum value in knapsack = %d\n", dp[n][W]);
    int w = W;
    printf("Selected items (1-based indices): ");
    for (int i = n; i > 0 && w > 0; i--) {
        if (dp[i][w] != dp[i - 1][w]) {
            printf("%d ", i);
            w -= wt[i - 1];
        }
    }
    printf("\n");
}

int main() {
    int n, W, val[MAX_ITEMS], wt[MAX_ITEMS];

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

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

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

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

    knapsack(n, W, wt, val);
    return 0;
}
```
Output

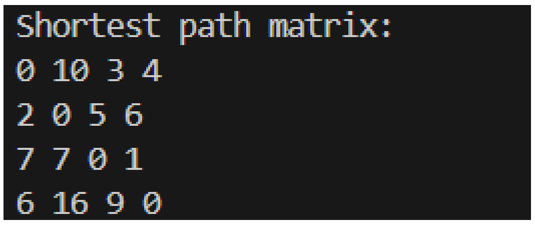


```
Enter number of items: 4
Enter weights of items:
Weight of item 1: 2
Weight of item 2: 3
Weight of item 3: 4
Weight of item 4: 5
Enter values of items:
Value of item 1: 3
Value of item 2: 4
Value of item 3: 5
Value of item 4: 8
Enter knapsack capacity: 5

DP Table:
  0   0   0   0   0   0
  0   0   3   3   3   3
  0   0   3   4   4   7
  0   0   3   4   5   7
  0   0   3   4   5   8

Maximum value in knapsack = 8
Selected items (1-based indices): 4
```

Program - 7
Question:
Implement All Pair Shortest paths problem using Floyd's algorithm.

Code
```
#include <stdio.h>
#define INF 99999
#define V 4

void floydWarshall(int graph[][V]) {
    int dist[V][V], i, j, k;


    for (i = 0; i < V; i++)
        for (j = 0; j < V; j++)
            dist[i][j] = graph[i][j];


    for (k = 0; k < V; k++) {
        for (i = 0; i < V; i++) {
            for (j = 0; j < V; j++) {
                if (dist[i][k] + dist[k][j] < dist[i][j])
                    dist[i][j] = dist[i][k] + dist[k][j];
            }
        }
    }


    printf("Shortest path matrix:\n");
    for (i = 0; i < V; i++) {
        for (j = 0; j < V; j++) {
            if (dist[i][j] == INF)
                printf("INF ");
            else
                printf("%d ", dist[i][j]);
        }
        printf("\n");
    }
}

int main() {

    int graph[V][V] = {
        {0, INF, 3, INF},
        {2, 0, INF, INF},
        {INF, 7, 0, 1},
        {6, INF, INF, 0}
    };

    floydWarshall(graph);

    return 0;
}
```

Output



```
Shortest path matrix:
0 10 3 4
2 0 5 6
7 7 0 1
6 16 9 0
```

Program - 8

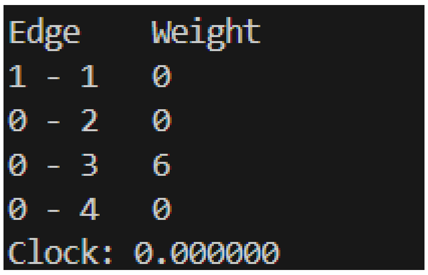
Question

(i) Find Minimum Cost Spanning Tree of a given undirected graph using Prim’s algorithm.

Code

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

int minKey(int key[], int mstSet[], int n)
{
    int min = INT_MAX, minIndex;
    for (int i = 0; i < n; i++)
        if (!mstSet[i] && key[i] < min)
        {
            min = key[i];
            minIndex = i;
        }
    return minIndex;
}

int printMST(int n, int parent[], int graph[n][n])
{
    printf("Edge \tWeight\n");
    for (int i = 1; i < n; i++)
        printf("%d - %d \t%d \n", parent[i], i,
graph[parent[i]][i]);
}

void prim(int n, int graph[n][n], int parent[n])
{
    int key[n];
    int mstSet[n];
    for (int i = 0; i < n; i++)
    {
        key[i] = INT_MAX;  //infinity
        mstSet[i] = -1;
    }

    key[0] = 0;
    for (int count = 0; count < n - 1; count++) {
        int u = minKey(key, mstSet, n);
        mstSet[u] = 1;

        for (int i = 0; i < n; i++)
        {
            if (graph[u][i] && !mstSet[i] && graph[u][i] < key[i])
            {
                parent[i] = u, key[i] = graph[u][i];
            }
        }
    }
}

int main()
{
    int graph[5][5] = { { 0, 2, 0, 6, 0 },
                        { 2, 0, 3, 8, 5 },
                        { 0, 3, 0, 0, 7 },
                        { 6, 8, 0, 0, 9 },
                        { 0, 5, 7, 9, 0 } };
    int mst[5] = {-1};
    clock_t start, end;

    start = clock();
    prim(5, graph, mst);
    end = clock();

    printMST(5, mst, graph);
    printf("Clock: %lf", (end - start)/CLK_TCK);
    return 0;
}
```

Output



```
Edge    Weight
1 - 1   0
0 - 2   0
0 - 3   6
0 - 4   0
Clock: 0.000000
```

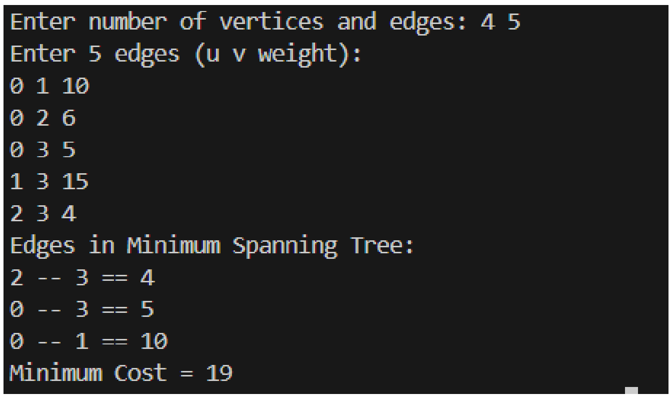
(ii) Find Minimum Cost Spanning Tree of a given undirected graph using Kruskal's algorithm.

Code

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

#define MAX 100

typedef struct {
    int u, v, weight;
} Edge;

int parent[MAX];
```

```
// Find set of an element i (with path compression)
int find(int i) {
    while (i != parent[i])
        i = parent[i];
    return i;
}

// Union of two sets
void unionSets(int i, int j) {
    int a = find(i);
    int b = find(j);
    parent[a] = b;
}

// Compare function for sorting edges by weight
int compare(const void *a, const void *b) {
    return ((Edge *)a)->weight - ((Edge *)b)->weight;
}

void kruskal(Edge edges[], int n, int e) {
    Edge result[MAX];
    int total_cost = 0, count = 0;

    // Sort edges by weight
    qsort(edges, e, sizeof(Edge), compare);

    // Initialize parent array
    for (int i = 0; i < n; i++)
        parent[i] = i;

    for (int i = 0; i < e && count < n - 1; i++) {
        int u = edges[i].u;
        int v = edges[i].v;
        if (find(u) != find(v)) {
            result[count++] = edges[i];
            total_cost += edges[i].weight;
            unionSets(u, v);
        }
    }

    printf("Edges in Minimum Spanning Tree:\n");
    for (int i = 0; i < count; i++)
        printf("%d -- %d == %d\n", result[i].u, result[i].v,
result[i].weight);

    printf("Minimum Cost = %d\n", total_cost);
}

int main() {
    int n, e;
    Edge edges[MAX];

    printf("Enter number of vertices and edges: ");
    scanf("%d %d", &n, &e);

    printf("Enter %d edges (u v weight):\n", e);
    for (int i = 0; i < e; i++) {
        scanf("%d %d %d", &edges[i].u, &edges[i].v,
&edges[i].weight);
    }

    kruskal(edges, n, e);
    return 0;
}
```

Output



```
Enter number of vertices and edges: 4 5
Enter 5 edges (u v weight):
0 1 10
0 2 6
0 3 5
1 3 15
2 3 4
Edges in Minimum Spanning Tree:
2 -- 3 == 4
0 -- 3 == 5
0 -- 1 == 10
Minimum Cost = 19
```

Program - 9

Question:

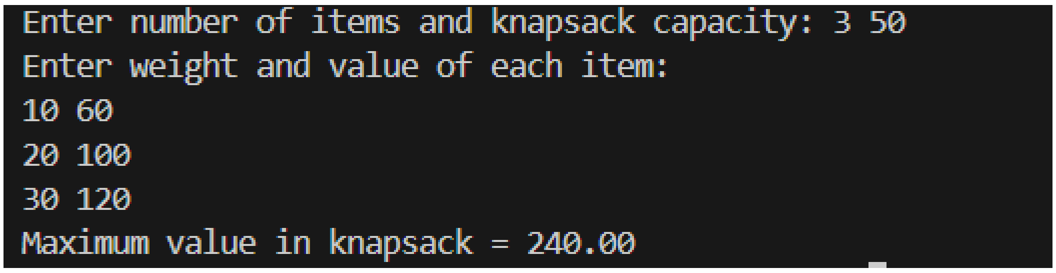
Implement Fractional Knapsack using Greedy technique.

Code

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

typedef struct {
    int weight, value;
    float ratio;
} Item;
```

```c
int cmp(const void* a, const void* b) {
    float r1 = ((Item*)a)->ratio;
    float r2 = ((Item*)b)->ratio;
    return (r2 > r1) - (r2 < r1);
}

int main() {
    int n;
    float capacity, total = 0.0;
    printf("Enter number of items and knapsack capacity: ");
    scanf("%d %f", &n, &capacity);

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

    qsort(items, n, sizeof(Item), cmp);

    for (int i = 0; i < n && capacity > 0; i++) {
        if (items[i].weight <= capacity) {
            capacity -= items[i].weight;
            total += items[i].value;
        } else {
            total += items[i].ratio * capacity;
            capacity = 0;
        }
    }

    printf("Maximum value in knapsack = %.2f\n", total);
    return 0;
}
```

Output



```
Enter number of items and knapsack capacity: 3 50
Enter weight and value of each item:
10 60
20 100
30 120
Maximum value in knapsack = 240.00
```

Program - 10

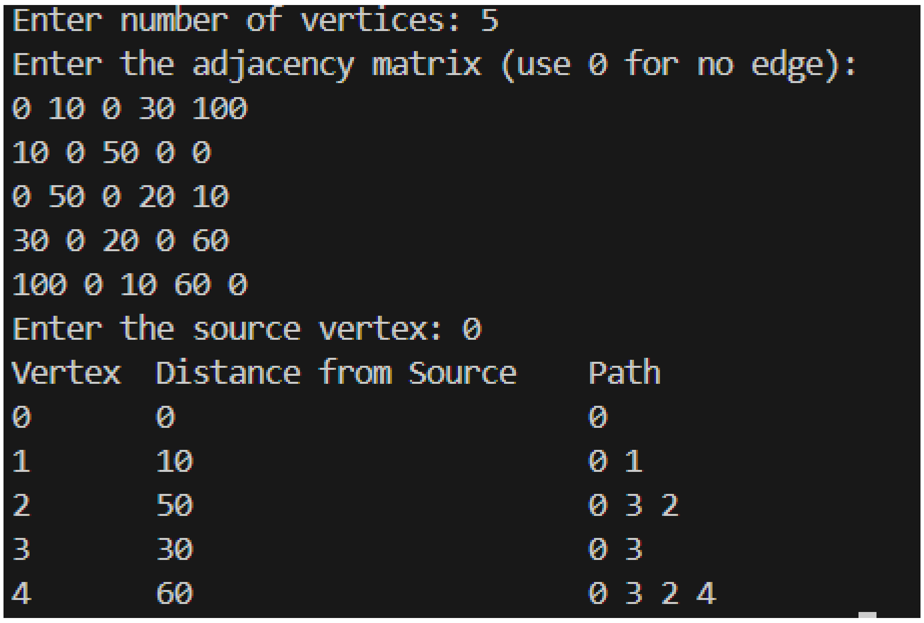
Question:

From a given vertex in a weighted connected graph, find shortest paths to other vertices using Dijkstra's algorithm.

Code

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

#define MAX 100
#define INF 99999  // Representation of infinity

int findMinVertex(int dist[], int visited[], int n) {
    int min = INF, minIndex = -1;
    for (int v = 0; v < n; v++) {
        if (!visited[v] && dist[v] < min) {
            min = dist[v];
            minIndex = v;
        }
    }
    return minIndex;
}

void dijkstra(int graph[MAX][MAX], int n, int src) {
    int dist[MAX], visited[MAX], parent[MAX];

    for (int i = 0; i < n; i++) {
        dist[i] = INF;
        visited[i] = 0;
        parent[i] = -1;
    }
    dist[src] = 0;

    for (int i = 0; i < n - 1; i++) {
        int u = findMinVertex(dist, visited, n);
        visited[u] = 1;

        for (int v = 0; v < n; v++) {
            if (!visited[v] && graph[u][v] && dist[u] + graph[u][v]
< dist[v]) {
                dist[v] = dist[u] + graph[u][v];
                parent[v] = u;
            }
        }
    }

    printf("Vertex\tDistance from Source\tPath\n");
    for (int i = 0; i < n; i++) {
        printf("%d\t%d\t\t\t", i, dist[i]);
        int j = i;
        int path[MAX], path_len = 0;
        while (j != -1) {
            path[path_len++] = j;
            j = parent[j];
        }
        for (int k = path_len - 1; k >= 0; k--) {
            printf("%d ", path[k]);
        }
        printf("\n");
    }
}

int main() {
    int n, src;
    int graph[MAX][MAX];

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

    printf("Enter the adjacency matrix (use 0 for no edge):\n");
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            scanf("%d", &graph[i][j]);

    printf("Enter the source vertex: ");
    scanf("%d", &src);

    dijkstra(graph, n, src);

    return 0;
}
```

Output



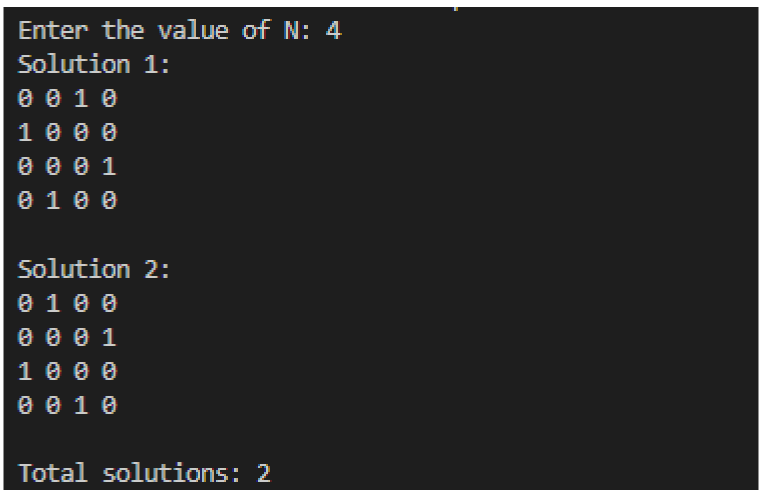
```
Enter number of vertices: 5
Enter the adjacency matrix (use 0 for no edge):
0 10 0 30 100
10 0 50 0 0
0 50 0 20 10
30 0 20 0 60
100 0 10 60 0
Enter the source vertex: 0
Vertex  Distance from Source    Path
0       0                       0
1       10                      0 1
2       50                      0 3 2
3       30                      0 3
4       60                      0 3 2 4
```

Program - 11
Question:
Implement “N-Queens Problem” using Backtracking.

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

int **board;
int N;
int solutionCount = 0;


int isSafe(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;
}
```

```
void solveNQueens(int col) {
    if (col == N) {
        solutionCount++;
        printf("Solution %d:\n", solutionCount);
        for (int i = 0; i < N; i++) {
            for (int j = 0; j < N; j++)
                printf("%d ", board[i][j]);
            printf("\n");
        }
        printf("\n");
        return;
    }

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

int main() {
    printf("Enter the value of N: ");
    scanf("%d", &N);
    board = (int **)malloc(N * sizeof(int *));
    for (int i = 0; i < N; i++) {
        board[i] = (int *)malloc(N * sizeof(int));
        for (int j = 0; j < N; j++)
            board[i][j] = 0;
    }

    solveNQueens(0);

    if (solutionCount == 0)
        printf("No solutions found.\n");
    else
        printf("Total solutions: %d\n", solutionCount);
    for (int i = 0; i < N; i++)
        free(board[i]);
    free(board);

    return 0;
}
```

Output



```
Enter the value of N: 4
Solution 1:
0 0 1 0
1 0 0 0
0 0 0 1
0 1 0 0

Solution 2:
0 1 0 0
0 0 0 1
1 0 0 0
0 0 1 0

Total solutions: 2
```