VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT
on

Analysis and Design of Algorithms

Submitted by

Amshu G M(1BM21CS019)

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
May-2023 to July-2023

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**" carried out by **Amshu G M (1BM21CS019),** who is Bonafede student of **B.M.S. College of Engineering.** It is in partial fulfillment for the award of **Bachelor of Engineering in Computer Science and Engineering** of the Visvesvaraya Technological University, Belgaum during the academic semester May-2023 to July-2023. The Lab report has been approved as it satisfies the academic requirements in respect of **Analysis and Design of Algorithms (22CS4PCADA)** work prescribed for the said degree.

Name of the Lab-In charge: Sunayana S
Assistant Professor
Department of CSE
BMSCE, Bengaluru

Dr. Jyothi S Nayak
Professor and Head
Department of CSE
BMSCE, Bengaluru

Index Sheet

Course Outcome

CO1	Analyze time complexity of Recursive and Non-recursive algorithms using asymptotic notations.
CO2	Apply various design techniques for the given problem.
CO3	Apply the knowledge of complexity classes P, NP, and NP-Complete and prove certain problems are NP-Complete
CO4	Design efficient algorithms and conduct practical experiments to solve problems.

1. a) Breadth First Search

Aim: To print all the reachable nodes from a given root node in a digraph using BFS method

Code:

```
#include<stdio.h>
#include<conio.h>
void insert_rear(int q[],int *r, int item, int size)
{
    if(*r==size)
    printf("Queue overflow!\n");
    else
    {
        *r=*r+1;
        q[*r]=item;
    }
}
int delete_front(int q[],int *r, int *f)
{
    int del_item=-1;
    *f=*f+1;
    del_item=q[*f];


    return del_item;
}
int isEmpty(int q[], int *r, int *f)
{
    if(*r==-1 || *r==*f)
    return 1;
    else
    return 0;
}
void main()
{
    int n,i,j,r=-1,f=-1;
    printf("Enter the number of vertices:\n");
    scanf("%d",&n);
    printf("Enter the adjacency matrix representing the graph:\n");
    int graph[n][n];
    int vis[n],q[n];
    for(int i=0;i<n;i++)
    {    for(int j=0;j<n;j++)
```

```
        {
            scanf("%d",&graph[i][j]);
        }
    }
      for(int i=0;i<n;i++)
    {
        vis[i]=0;
    }
    int k=0;
    printf("Starting vertex:%d\n",k);
    printf("Breadth First Search of the graph\n");
    printf("%d ",k);
    vis[k]=1;
    insert_rear(q,&r,k,n);
    while(isEmpty(q,&r,&f)==0)
    {
        int node=delete_front(q,&r,&f);
        for(j=0;j<n;j++)
        {
            if(graph[node][j]==1 && vis[j]==0)
            {
                printf("%d ",j);
                vis[j]=1;
                insert_rear(q,&r,j,n);
            }
        }
    }
}
```

Output:

```
D:\Codes\c\ADA_LAB>.\a.exe
Enter the number of vertices:
6
Enter the adjacency matrix representing the graph:
0 1 1 0 1 1
1 0 0 1 0 0
1 1 0 0 0 1
0 0 1 0 1 1
1 0 0 0 0 1

1 0 1 1 0 0
Starting vertex:0
Breadth First Search of the graph
0 1 2 4 5 3
D:\Codes\c\ADA_LAB>
```

1. b) Depth First Search

Aim: To check whether a given graph is connected or not using DFS method

Code:

```
int graph[20][20];
void DFS(int i,int vis[],int n)
{
    int j;
    printf("%d ",i);
    vis[i]=1;
    for(j=0;j<n;j++)
    {
        if(graph[i][j]==1 && vis[j]==0)
        {
            DFS(j,vis,n);
        }
    }
}
void main()
{
    int n,i,j,top=-1,start;
    printf("Enter the number of vertices:\n");
    scanf("%d",&n);
    printf("Enter the adjacency matrix representing the graph:\n");
    int vis[n],st[n];
    for(int i=0;i<n;i++)
    {
        for(int j=0;j<n;j++)
        {
            scanf("%d",&graph[i][j]);
        }
    }
    for(int i=0;i<n;i++)
    {
        vis[i]=0;
    }
    printf("Enter the starting vertex:");
    scanf("%d",&start);
    printf("Depth First Search of the graph is\n");
    DFS(start,vis,n);
}
```

Output:

```
D:\Codes\c\ADA_LAB>.\a.exe
Enter the number of vertices:
5
Enter the adjacency matrix representing the graph:
0 1 1 1 1
1 0 0 0 0
1 0 0 0 0
1 0 0 0 0
1 0 0 0 0
Enter the starting vertex:2
Depth First Search of the graph is
2 0 1 3 4
D:\Codes\c\ADA_LAB>
```

2. Topological Sorting

Aim: To obtain the Topological ordering of vertices in a given digraph

Code:

```c
#include <stdio.h>

int main()
{
    int n;
    printf("Enter the no of vertices: ");
    scanf("%d", &n);
    int a[n][n], indeg[n], flag[n];

    int i, j, k, count = 0;
    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;
        flag[i] = 0;
    }

    for (i = 0; i < n; i++)
        for (j = 0; j < n; j++)
            indeg[i] = indeg[i] + a[j][i];

    printf("\nThe topological order is: ");

    while (count < n)
    {
        for (k = 0; k < n; k++)
        {
            if ((indeg[k] == 0) && (flag[k] == 0))
            {
                printf("%d ", (k + 1));
                flag[k] = 1;
```

```
            }

            for (i = 0; i < n; i++)
            {
                if (a[i][k] == 1)
                    indeg[k]--;
            }
        }
        count++;
    }

    return 0;
}
```

Output:

```
D:\Codes\c\ADA_LAB>gcc Topological_Sort.c

D:\Codes\c\ADA_LAB>.\a.exe
Enter the no of vertices: 4
Enter the adjacency matrix:
0 1 1 0
0 0 0 1
1 0 0 0
1 1 1 0

The topological order is: 1 2 3 4
```

3. Johnson Trotter algorithm

Aim: To generate permutations of n numbers using Johnson Trotter algorithm

Code:

```c
#include <stdio.h>

#define MAXN 10

int p[MAXN];
int dir[MAXN];


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


void swap(int* a, int* b) {
    int temp = *a;
    *a = *b;
    *b = temp;
}


void generatePermutations(int n) {

    printPermutation(n);

    int mobile, mobileIndex;
    while (1) {
        mobile = 0;
        mobileIndex = -1;


        for (int i = 0; i < n; i++) {
            if ((dir[i] == -1 && i > 0 && p[i] > p[i - 1]) ||
                (dir[i] == 1 && i < n - 1 && p[i] > p[i + 1])) {
                if (p[i] > mobile) {
                    mobile = p[i];
```

```
                    mobileIndex = i;
                }
            }
        }


        if (mobileIndex == -1) {
            break;
        }


        if (dir[mobileIndex] == -1) {
            swap(&p[mobileIndex], &p[mobileIndex - 1]);
            swap(&dir[mobileIndex], &dir[mobileIndex - 1]);
            mobileIndex--;
        }
        else if (dir[mobileIndex] == 1) {
            swap(&p[mobileIndex], &p[mobileIndex + 1]);
            swap(&dir[mobileIndex], &dir[mobileIndex + 1]);
            mobileIndex++;
        }

        for (int i = 0; i < n; i++) {
            if (p[i] > mobile) {
                dir[i] = -dir[i];
            }
        }


        printPermutation(n);
    }
}

int main() {
    int n;

    printf("Enter the value of n (maximum %d): ", MAXN);
    scanf("%d", &n);

    if (n <= 0 || n > MAXN) {
        printf("Invalid value of n!\n");
        return 0;
    }
```

```
    for (int i = 0; i < n; i++) {
        p[i] = i + 1;
        dir[i] = -1;
    }


    generatePermutations(n);

    return 0;
}
```

Output:

```
D:\Codes\c\ADA_LAB>gcc johnson_trott.c

D:\Codes\c\ADA_LAB>.\a.exe
Enter the value of n (maximum 10): 4
1 2 3 4
1 2 4 3
1 4 2 3
4 1 2 3
4 1 3 2
1 4 3 2
1 3 4 2
1 3 2 4
3 1 2 4
3 1 4 2
3 4 1 2
4 3 1 2
4 3 2 1
3 4 2 1
3 2 4 1
3 2 1 4
2 3 1 4
2 3 4 1
2 4 3 1
4 2 3 1
4 2 1 3
2 4 1 3
2 1 4 3
2 1 3 4
```

4. Merge Sort

Aim: To sort a given set of N integer elements using Merge Sort technique, compute its time taken for different values of N and record the time taken to sort

Code:

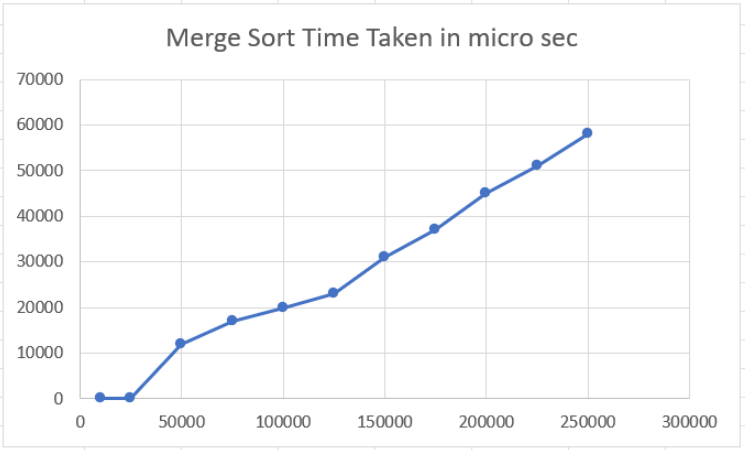
```
#include<stdio.h>
#include<time.h>
#include<stdlib.h>

void merge(int arr[],int l,int m,int h){
    int i,j,k;
    int n1=m-l+1;
    int n2=h-m;
    int temp1[n1];
    int temp2[n2];
    for(int i=0;i<n1;i++){
        temp1[i]=arr[l+i];
    }
    for(int i=0;i<n2;i++){
        temp2[i]=arr[m+i+1];
    }
    i=0;
    j=0;
    k=l;
    while(i<n1&&j<n2){
        if(temp1[i]<temp2[j]){
            arr[k]=temp1[i];
            i++;
        }
        else{
            arr[k]=temp2[j];
            j++;
        }
        k++;
    }
    while(i<n1){
        arr[k]=temp1[i];
        i++;
        k++;
    }
    while(j<n2){
```

```
        arr[k]=temp2[j];
        j++;
        k++;
    }

}

void mergesort(int arr[],int l, int h){
    if(l<h){
        int m=(l+h-1)/2;
        mergesort(arr,l,m);
        mergesort(arr,m+1,h);
        merge(arr,l,m,h);
    }
}

int main(){
    clock_t start,end;
    int arr[100000],n,low,high;
    printf("Enter size of array: ");
    scanf("%d",&n);
    for(int i=0;i<n;i++){
        arr[i]=rand()%1000;
    }
    low=0;
    high=n-1;
    printf("The Elements before sorting:\n ");
   for(int i=0;i<n;i++){
        printf("%d ",arr[i]);
    }
    start=clock();
    mergesort(arr,low,high);
    end=clock();
    double time=(double)(end-start)/CLOCKS_PER_SEC;
    printf("\nThe Elements after sorting:\n ");
    for(int i=0;i<n;i++){
        printf("%d ",arr[i]);
    }
    printf("\nTime taken:%lf in microseconds\n",time*1000000);
    return 0;
}
```

Output with input size vs time graph:

```
D:\Codes\c\ADA_LAB>gcc merge_sort.c

D:\Codes\c\ADA_LAB>.\a.exe
Enter size of array: 5
The Elements before sorting:
 41 467 334 500 169
The Elements after sorting:
 41 169 334 467 500
Time taken:0.000000 in microseconds
```

Merge Sort	
Input Size	Time Taken in micro sec
10000	0
25000	0
50000	12000
75000	17000
100000	20000
125000	23000
150000	31000
175000	37000
200000	45000
225000	51000
250000	58000



5. Quick Sort

Aim: To sort a given set of N integer elements using Quick Sort technique and compute its time taken

Code:

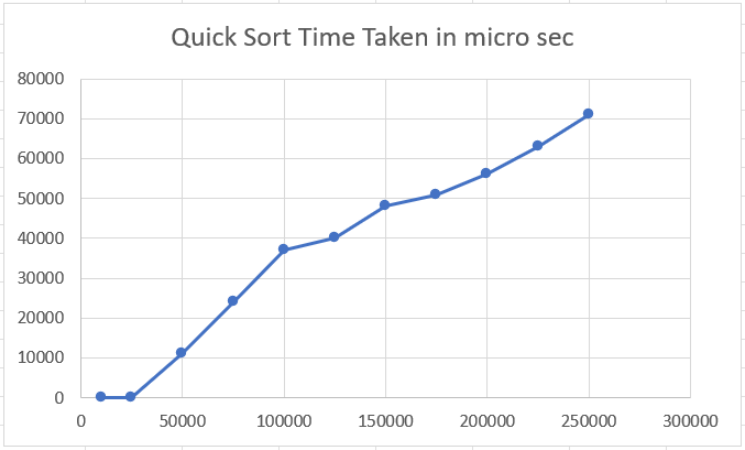
```
#include<stdio.h>
#include<time.h>
#include<stdlib.h>

void merge(int arr[],int l,int m,int h){
    int i,j,k;
    int n1=m-l+1;
    int n2=h-m;
    int temp1[n1];
    int temp2[n2];
    for(int i=0;i<n1;i++){
        temp1[i]=arr[l+i];
    }
    for(int i=0;i<n2;i++){
        temp2[i]=arr[m+i+1];
    }
    i=0;
    j=0;
    k=l;
    while(i<n1&&j<n2){
        if(temp1[i]<temp2[j]){
            arr[k]=temp1[i];
            i++;
        }
        else{
            arr[k]=temp2[j];
            j++;
        }
        k++;
    }
    while(i<n1){
        arr[k]=temp1[i];
        i++;
        k++;
    }
    while(j<n2){
        arr[k]=temp2[j];
        j++;
```

```
        k++;
    }

}

void mergesort(int arr[],int l, int h){
    if(l<h){
        int m=(l+h-1)/2;
        mergesort(arr,l,m);
        mergesort(arr,m+1,h);
        merge(arr,l,m,h);
    }
}

int main(){
    clock_t start,end;
    int arr[100000],n,low,high;
    printf("Enter size of array: ");
    scanf("%d",&n);
    for(int i=0;i<n;i++){
        arr[i]=rand()%1000;
    }
    low=0;
    high=n-1;
    printf("The Elements before sorting:\n ");
   for(int i=0;i<n;i++){
        printf("%d ",arr[i]);
    }
    start=clock();
    mergesort(arr,low,high);
    end=clock();
    double time=(double)(end-start)/CLOCKS_PER_SEC;
    printf("\nThe Elements after sorting:\n ");
    for(int i=0;i<n;i++){
        printf("%d ",arr[i]);
    }
    printf("\nTime taken:%lf in microseconds\n",time*1000000);
    return 0;
}
```

Output with input size vs time graph:

```
D:\Codes\c\ADA_LAB>gcc quick_sort.c

D:\Codes\c\ADA_LAB>.\a.exe
Enter size of array: 5
The Elements before sorting:
 41 467 334 500 169
The Elements after sorting:
 41 169 334 467 500
Time taken:0.000000 in microseconds
```

Quick Sort	
Input Size	Time Taken in micro sec
10000	0
25000	0
50000	11000
75000	24000
100000	37000
125000	40000
150000	48000
175000	51000
200000	56000
225000	63000
250000	71000



6. Heap Sort

Aim: To sort a given set of N integer elements using Heap Sort technique and compute its time taken

Code:

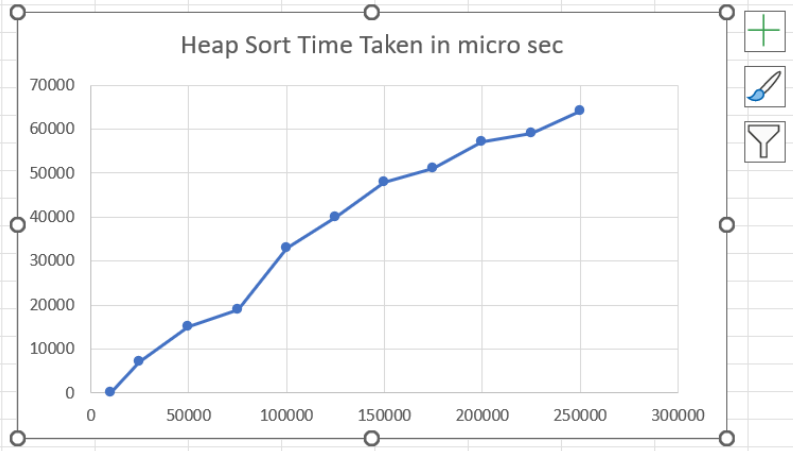
```
#include <stdio.h>
#include<time.h>
#include<stdlib.h>

void swap(int *a, int *b)
{
    int temp = *a;
    *a = *b;
    *b = temp;
}

void heapify(int arr[], int N, int i)
{
    int largest = i;
    int left = 2 * i + 1;
    int right = 2 * i + 2;

    if (left < N && arr[left] > arr[largest])
    {
        largest = left;
    }

    if (right < N && arr[right] > arr[largest])
    {
        largest = right;
    }

    if (largest != i)
    {
        swap(&arr[i], &arr[largest]);
        heapify(arr, N, largest);
    }
}

void heapSort(int arr[], int N)
{
    for (int i = N / 2 - 1; i >= 0; i--)
```

```
    {
        heapify(arr, N, i);
    }

    for (int i = N - 1; i >= 0; i--)
    {
        swap(&arr[0], &arr[i]);
        heapify(arr, i, 0);
    }
}

int main(void)
{
    int n;
    srand(time(0));
    clock_t start, end;
    printf("Enter the size of array: ");
    scanf("%d", &n);
    int arr[n];
    for (int i = 0; i < n; i++)
    {
        // scanf("%d", &arr[i]);
        arr[i]=rand()%1000;
    }
     printf("The Elements before sorting:\n ");
     for (int i = 0; i < n; i++)
    {
        printf("%d ", arr[i]);

    }
    start=clock();
    heapSort(arr, n);
    end=clock();
    double time=(double)(end-start)/CLOCKS_PER_SEC;
     printf("\nThe Elements after sorting:\n ");
    for (int i = 0; i < n; i++)
    {
        printf("%d ", arr[i]);

    }
   printf("\nTime taken:%lf in microseconds\n",time*1000000);
}
```

Output with input size vs time graph:

```
D:\Codes\c\ADA_LAB>gcc heapsort.c

D:\Codes\c\ADA_LAB>.\a.exe
Enter the size of array: 5
The Elements before sorting:
 85 748 575 194 98
The Elements after sorting:
 85 98 194 575 748
Time taken:0.000000 in microseconds
```

	Heap Sort
Input Size	Time Taken in micro sec
10000	0
25000	7000
50000	15000
75000	19000
100000	33000
125000	40000
150000	48000
175000	51000
200000	57000
225000	59000
250000	64000



7. 0/1 Knapsack Problem

Aim: To optimize(maximize) the items in the knapsack for our requirement using 0/1 Knapsack algorithm

Code:

```c
#include <stdio.h>

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

    printf("Enter the price of each item: ");
    int price[n];
    int i;
    for (i = 0; i < n; i++)
    {
        scanf("%d", &price[i]);
    }

    printf("Enter the weight of each item: ");
    int weight[n];
    for (i = 0; i < n; i++)
    {
        scanf("%d", &weight[i]);
    }

    printf("Enter the max weight: ");
    int W;
    scanf("%d", &W);

    printf("\nThe dp table is:\n");
    int dp[n + 1][W + 1];
    for (i = 0; i <= n; i++)
    {
        for (int j = 0; j <= W; j++)
        {
            if (i == 0 || j == 0)
            {
                dp[i][j] = 0;
```

```
            }
            else if (weight[i - 1] <= j)
            {
                dp[i][j] = (price[i - 1] + dp[i - 1][j - weight[i - 1]]) > dp[i -
1][j] ? (price[i - 1] + dp[i - 1][j - weight[i - 1]]) : dp[i - 1][j];
            }
            else
            {
                dp[i][j] = dp[i - 1][j];
            }
            printf("%d  ", dp[i][j]);
        }
        printf("\n");
    }

    printf("\nThe maximum value we can get is: %d", dp[n][W]);

    return 0;
}
```

Output:

```
D:\Codes\c\ADA_LAB>.\a.exe
Enter the number of items: 4
Enter the price of each item: 2 3 4 1
Enter the weight of each item: 4 5 2 8
Enter the max weight: 12

The dp table is:
0  0  0  0  0  0  0  0  0  0  0  0  0
0  0  0  0  2  2  2  2  2  2  2  2  2
0  0  0  0  2  3  3  3  3  5  5  5  5
0  0  4  4  4  4  6  7  7  7  7  9  9
0  0  4  4  4  4  6  7  7  7  7  9  9

The maximum value we can get is: 9
```

8. Floyd's Algorithm

Aim: To find out the shortest path between all pairs of vertices

Code:

```c
#include <stdio.h>
#define INF 999
int V;
void printSolution(int dist[][V]);

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

    printSolution(dist);
}

void printSolution(int dist[][V])
{
    printf(
        "The following matrix shows the shortest distances"
        " between every pair of vertices \n");
    for (int i = 0; i < V; i++)
    {
        for (int j = 0; j < V; j++)
        {
                printf("%7d", dist[i][j]);
        }
        printf("\n");
    }
```

```c
}

// driver's code
int main()
{
    printf("Enter the number of vertices\n");
    scanf("%d", &V);
    int graph[V][V];
    printf("Enter the weighted Adjacency matrix (give 999 for infinty)\n");
    for (int i = 0; i < V; i++)
    {
        for (int j = 0; j < V; j++)
        {
            scanf("%d", &graph[i][j]);
        }
    }

    // Function call
    floydWarshall(graph);
    return 0;
}
```

Output:

```
D:\Codes\c\ADA_LAB>.\a.exe
Enter the number of vertices
5
give 999 for infinty
^C
D:\Codes\c\ADA_LAB>gcc Floyds.c

D:\Codes\c\ADA_LAB>.\a.exe
Enter the number of vertices
4
Enter the weighted Adjacency matrix (give 999 for infinty)
0 4 6 999
9 0 12 999
999 2 0 12
12 10 999 0
The following matrix shows the shortest distances between every pair of vertices
      0      4      6     18
      9      0     12     24
     11      2      0     12
     12     10     18      0
```

9. Prim's and Kruskal's algorithm

Aim: To find minimal spanning tree of a graph using Prim's and Kruskal's algorithms

Prim's Algorithm Code:

```c
#include <stdio.h>

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

    printf("Enter the adjacency matrix(use 999 as infinity):\n");
    int adj[n][n];
    int i, j, k;
    for (i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
        {
            scanf("%d", &adj[i][j]);
        }
    }

    int visited[n];
    for (i = 0; i < n; i++)
    {
        visited[i] = 0;
    }

    printf("Enter the starting vertex: ");
    int start;
    scanf("%d", &start);
    visited[start] = 1;

    printf("\nThe minimal spanning tree is:\nEdge : Weight\n");
    for (k = 0; k < n - 1; k++)
    {
        int min = 999;
        int u = 0;
        int v = 0;
        for (i = 0; i < n; i++)
```

```
        {
            if (visited[i])
            {
                for (j = 0; j < n; j++)
                {
                    if (!visited[j] && adj[i][j])
                    {
                        if (min > adj[i][j])
                        {
                            min = adj[i][j];
                            u = i;
                            v = j;
                        }
                    }
                }
            }
        }
        printf("%d - %d : %d\n", u, v, adj[u][v]);
        visited[v] = 1;
    }
}
```

Output:

```
D:\Codes\c\ADA_LAB>gcc prims.c

D:\Codes\c\ADA_LAB>.\a.exe
Enter the number of vertices: 4
Enter the adjacency matrix(use 999 as infinity):
0 8 12 999
999 0 7 3
2 6 0 999
13 999 7 0
Enter the starting vertex: 2

The minimal spanning tree is:
Edge : Weight
2 - 0 : 2
2 - 1 : 6
1 - 3 : 3
```

Kruskal's Algorithm Code:

```c
#include <stdio.h>

int find(int v, int *parent)
{
    while (parent[v] != v)
    {
        v = parent[v];
    }
    return v;
}

void union1(int i, int j, int *parent)
{
    if (i < j)
        parent[j] = i;
    else
        parent[i] = j;
}

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

    printf("Enter the adjacency matrix(use 999 as infinity):\n");
    int adj[n][n];
    int i;
    for (i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
        {
            scanf("%d", &adj[i][j]);
        }
    }
    int parent[n];
    for (i = 0; i < n; i++)
    {
        parent[i] = i;
    }

    int count = 0, k = 0, min, sum = 0, j, t[n][n], u, v;
```

```
    while (count != n - 1)
    {
        min = 999;
        for (i = 0; i < n; i++)
        {
            for (j = 0; j < n; j++)
            {
                if (adj[i][j] < min && adj[i][j] != 0)
                {
                    min = adj[i][j];
                    u = i;
                    v = j;
                }
            }
        }
        i = find(u, parent);
        j = find(v, parent);

        if (i != j)
        {
            union1(i, j, parent);
            t[k][0] = u;
            t[k][1] = v;
            k++;
            count++;
            sum = sum + adj[u][v];
        }
        adj[u][v] = adj[v][u] = 999;
    }
    if (count == n - 1)
    {
        printf("The minimal spanning tree is as:\n");
        for (i = 0; i < n - 1; i++)
        {
            printf("%d -> %d\n", t[i][0], t[i][1]);
        }
        printf("Cost of spanning tree = %d\n", sum);
    }
    else
    {
        printf("\nSpanning tree does not exist!");
    }
}
```

Output:

```
D:\Codes\c\ADA_LAB>gcc kruskal.c

D:\Codes\c\ADA_LAB>.\a.exe
Enter the number of vertices: 4
Enter the adjacency matrix(use 999 as infinity):
0 999 999 2
3 0 999 8
1 4 0 12
999 999 6 0
The minimal spanning tree is as:
2 -> 0
0 -> 3
1 -> 0
Cost of spanning tree = 6
```

10. Dijkstra's Algorithm

Aim: To find shortest paths to other vertices from a given vertex in a weighted connected graph using Dijkstra's algorithm

Code:

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

int V;

int minDistance(int dist[], bool sptSet[])
{
    int min = INT_MAX, 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;
}

void printSolution(int dist[])
{
    printf("Vertex \t\t Distance from Source\n");
    for (int i = 0; i < V; i++)
    {
        printf("%d \t\t\t\t %d\n", i+1, dist[i]);
    }
}

void dijkstra(int graph[V][V], int src)
{
    int dist[V];

    bool sptSet[V];

    for (int i = 0; i < V; i++)
    {
```

```
        dist[i] = INT_MAX, sptSet[i] = false;
    }

    dist[src] = 0;

    for (int count = 0; count < V - 1; count++)
    {
        int u = minDistance(dist, sptSet);
        sptSet[u] = true;

        for (int v = 0; v < V; v++)
            if (!sptSet[v] && graph[u][v] && dist[u] != INT_MAX && dist[u] +
graph[u][v] < dist[v])
            {
                dist[v] = dist[u] + graph[u][v];
            }
    }

    printSolution(dist);
}

int main()
{
    printf("Enter the number of vertices: ");
    scanf("%d", &V);
    int graph[V][V],source;

    printf("Enter cost matrix(use 999 for infinity):\n");
    for (int i = 0; i < V; i++)
    {
        printf("Enter the row %d: ", i + 1);
        for (int j = 0; j < V; j++)
        {
            scanf("%d", &graph[i][j]);
        }
    }
    printf("Enter the Source vertex named from 1 to %d:",V);
    scanf("%d",&source);

    dijkstra(graph, source-1);

    return 0;
}
```

Output:

```
D:\Codes\c\ADA_LAB>gcc dijkstras.c

D:\Codes\c\ADA_LAB>.\a.exe
Enter the number of vertices: 4
Enter cost matrix(use 999 for infinity):
Enter the row 1: 0 8 13 999
Enter the row 2: 12 0 999 999
Enter the row 3: 7 5 0 999
Enter the row 4: 999 999 9 0
Enter the Source vertex named from 1 to 4:2
Vertex          Distance from Source
1                               12
2                               0
3                               25
4                               999
```

11. N – Queen's Problem

Aim: To calculate a solution to place N queens in an N x N chess board such that no two queens cancel each other

Code:

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

int n;

bool isSafe(int **arr, int x, int y)
{
  int row, col;

  for (row = 0; row < x; row++)
  {
    if (arr[row][y] == 1)
    {
      return false;
    }
  }

  for (row = x, col = y; row >= 0 && col >= 0; row--, col--)
  {
    if (arr[row][col] == 1)
    {
      return false;
    }
  }
  for (row = x, col = y; row >= 0 && col < n; row--, col++)
  {
    if (arr[row][col] == 1)
    {
      return false;
    }
  }

  return true;
}
```

```c
bool nQueen(int **arr, int x)
{
  if (x >= n)
  {
    return true;
  }

  for (int col = 0; col < n; col++)
  {
    if (isSafe(arr, x, col))
    {
      arr[x][col] = 1;

      if (nQueen(arr, x + 1))
      {
        return true;
      }

      arr[x][col] = 0;
    }
  }

  return false;
}

int main(void)
{
  printf("Enter the size of board: ");
  scanf("%d", &n);
  int **arr = (int **)malloc(n * sizeof(int *));

  int i, j;
  for (i = 0; i < n; i++)
  {
    arr[i] = (int *)malloc(n * sizeof(int));
    for (j = 0; j < n; j++)
    {
      arr[i][j] = 0;
    }
  }

  if (nQueen(arr, 0))
  {
    for (i = 0; i < n; i++)
```

```
    {
      for (j = 0; j < n; j++)
      {
        printf("%d ", arr[i][j]);
      }
      printf("\n");
    }
  }
  else
  {
    printf("\nSolution does not exist!");
  }
}
```

Output:

```
D:\Codes\c\ADA_LAB>.\a.exe
Enter the size of board: 5
1 0 0 0 0
0 0 1 0 0
0 0 0 0 1
0 1 0 0 0
0 0 0 1 0
```