VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

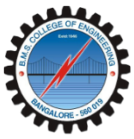
LAB REPORT
on

Analysis and Design of Algorithms

Submitted by

ARVIND ASHOK (1BM21CS032)

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
June-2023 to September-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 **Arvind Ashok (1BM21CS032),** who is a bonafide 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 June-2023 to September-2023. The Lab report has been approved as it satisfies the academic requirements in respect of a **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.

Write program to do the following:
a. Print all the nodes reachable from a given starting node in a digraph using BFS method.

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

int a[10][20], q[10], visited[10], n, i, j, f = 0, r = -1;

void bfs(int k) {
    for(i = 0; i < n; i++){
        if(a[k][i] && visited[i]==0){
            q[++r] = i;
        }
    }
    if(f <= r){
        visited[q[f]] = 1;
        bfs(q[f++]);
    }
}

int main() {
    int v;
    printf("\nEnter the number of vertices:");
    scanf("%d", &n);

    for(i=1; i <= n; i++) {
        q[i] = 0;
        visited[i] = 0;
    }

    printf("\nEnter graph data in matrix form:\n");
    for(i=0; i<n; i++) {
        for(j=0;j<n;j++) {
            scanf("%d", &a[i][j]);
        }
    }

    printf("\nEnter the starting vertex:");
    scanf("%d", &v);
    bfs(v);
```

```
    printf("\nThe node which are reachable are:\n");

    for(i=0; i < n; i++) {
        if(visited[i])
            printf("%d\t", i);
        else {
                    printf("\nBfs is not possible. Not all nodes are
  reachable!\n");
            break;
        }
    }
    return 0;
}
```

```
Enter the number of vertices
5
Enter graph data in the form of adjacency matrix
0 1 0 1 0
1 0 1 1 0
0 1 0 0 1
1 1 0 0 1
0 0 1 1 0

Enter the starting vertex
1
The nodes which are reachable are:
1       2       3       4       5
```

b. Check whether a given graph is connected or not using DFS method.

```
#include<stdio.h>
int graph[20][20], vis[10];

void DFS(int i,int n){
    int j;
    printf("Visited:%d\n",i);
    vis[i]=1;
    for(j=0;j<n;j++){
        if(graph[i][j]==1 && vis[j]==0){
            DFS(j,n);
        }
    }
}
```

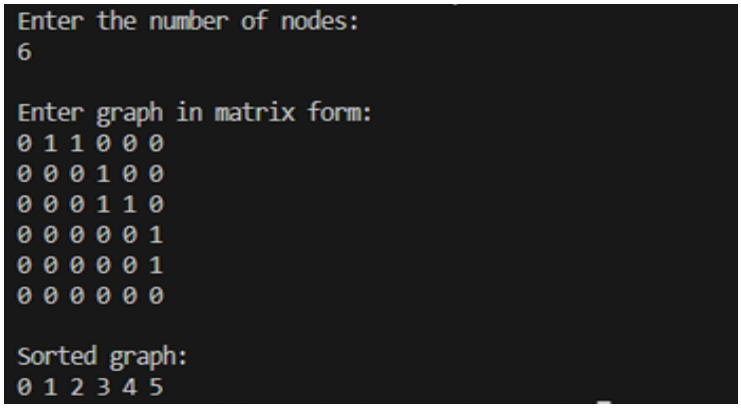
```
void main(){

    int n,i,j,top=-1;
    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;
    }
    DFS(0,n);
}
```

```
Enter number of vertices 5
0 1 0 1 0
1 0 1 1 0
0 1 0 0 1
1 1 0 0 1
0 0 1 1 0

0->1
1->2
2->4
4->3
```

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

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

int a[10][10], visited[10], s[10], t=-1, n;

void topological(int node){
    int i;
    visited[node] = 1;
    for (i = 0; i < n; i++){
```

```
        if (a[node][i]==1 && visited[node]==0){
            topological(i);
        }
    }
    s[++t] = node;
}

int main (){

    int i,j;

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

    for (i=0; i<n; i++)
        s[i] = 0;

    printf("\nEnter graph in matrix form:\n");
    for (i=0; i<n; i++){
        for (j=0; j<n; j++){
            scanf("%d", &a[i][j]);
        }
    }
    for (i=0; i<n; i++){
        topological(i);
    }
    printf ("\nSorted graph:\n");
    for (i = 0; i<=t; i++){
        printf("%d ", s[i]);
    }

        return 0;
}
```



Implement Johnson Trotter algorithm to generate permutations.

```c
#include <stdio.h>

#include <stdlib.h>

void swap (int *a, int *b){

    int temp = *a;

    *a= *b;

    *b = temp;

}

int main(){

    int n,i;

    int val[5], dir[5];

    printf ("Enter number of elements:\n");

    scanf("%d",&n);



    for (i=0; i<n; i++){

        val[i]=i+1;

        dir[i]=-1;
```

```
    }
    int c=0;
    while (1){
        c++;
        for (int i = 0; i < n; i++) {
            printf ("%d ",val[i]);
        }
        printf ("  %d\n",c);
        int mobileIndex = -1;
        int mobile = 0;
       for (int i = 0; i < n; i++) {
            if (dir[i] == -1 && i != 0) {
                if (val[i] > val[i - 1] && val[i] > mobile) {
                    mobile = val[i];
                    mobileIndex = i;
                }
            }
            if (dir[i] == 1 && i != n - 1) {
                if (val[i] > val[i + 1] && val[i] > mobile) {
                    mobile = val[i];
                    mobileIndex = i;
                }
            }
        }
        if (mobileIndex == -1) {
```

```
            break;
        }
        if (dir[mobileIndex] == -1) {
            swap(&val[mobileIndex], &val[mobileIndex - 1]);
            swap(&dir[mobileIndex], &dir[mobileIndex - 1]);
        } else if (dir[mobileIndex] == 1) {
            swap(&val[mobileIndex], &val[mobileIndex + 1]);
            swap(&dir[mobileIndex], &dir[mobileIndex + 1]);
        }
        for (int i = 0; i < n; i++) {
            if (val[i] > mobile) {
                dir[i] = -dir[i];
            }
        }
    }

    return 0;
}
```

```
PS C:\Users\arvin\Downloads\ADA_Lab-main\ADA_Lab-main> .\a.exe
Enter number of elements:
4
1 2 3 4   1
1 2 4 3   2
1 4 2 3   3
4 1 2 3   4
4 1 3 2   5
1 4 3 2   6
1 3 4 2   7
1 3 2 4   8
3 1 2 4   9
3 1 4 2   10
3 4 1 2   11
4 3 1 2   12
4 3 2 1   13
3 4 2 1   14
3 2 4 1   15
3 2 1 4   16
2 3 1 4   17
2 3 4 1   18
2 4 3 1   19
4 2 3 1   20
4 2 1 3   21
2 4 1 3   22
2 1 4 3   23
2 1 3 4   24
```

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.

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

int res[10], n;
void SortedMerge(int arr[], int l, int m, int h) {
    int i=l, j=m+1, k=0;

    while (i<=m && j<=h){
        if (arr[i]<arr[j]){
            res[k++]=arr[i++];
        }
        else{
            res[k++]=arr[j++];
        }
```

```
    }
    while (i<=m){
        res[k++]=arr[i++];
    }
    while (j<=h){
        res[k++]=arr[j++];
    }
    for (int i=0; i<(h-l)+1; i++) {
        arr[l+i]=res[i];
    }
}
void Merge(int arr[], int l, int h){
    if (l<h){
        int m = (l+h)/2;
        Merge(arr, l, m);
        Merge(arr, m+1, h);
        SortedMerge(arr, l, m , h);
    }
}

int main(){
    int a[10],i;
    printf ("Enter size of array:\n");
    scanf("%d",&n);
    printf ("Enter elements:\n");
    for (i=0; i<n; i++){
        scanf("%d",&a[i]);
    }
    Merge(a,0,n-1);

    for (i=0; i<n; i++){
        printf ("%d  ",a[i]);
    }
    return 0;
}
```

```
PS C:\Users\arvin\Downloads\ADA_Lab-main\ADA_Lab-main> gcc .\mergesort.c
PS C:\Users\arvin\Downloads\ADA_Lab-main\ADA_Lab-main> .\a.exe
Enter size of array:
6
Enter elements:
72 3 5 2 1
6
1  2  3  5  6  72
```

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

```c
#include <stdio.h>

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

int partition(int arr[], int low, int high) {
    int i=low, j=high+1;
    int pivot=arr[low];

    while (i<j){
        while (pivot >= arr[i]) i++;
        while (pivot < arr[j]) j--;

        if (i<j) swap(&arr[i], &arr[j]);
    }
    swap (&arr[low], &arr[j]);
    return j;
}

void quickSort(int arr[], int low, int high) {
    if (low < high) {
        int j = partition(arr, low, high);

        quickSort(arr, low, j-1);
        quickSort(arr, j+1, high);
    }
}
```

```c
int main() {
    int arr[10], n, i;

    printf("Enter no. of elemetns:\n");
    scanf ("%d", &n);

    printf ("Enter elements:\n");
    for (i=0; i<n; i++){
        scanf ("%d",&arr[i]);
    }
    quickSort(arr, 0, n-1);

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

    return 0;
}
```

```
PS C:\Users\arvin\Downloads\ADA_Lab-main\ADA_Lab-main> gcc .\quicksort.c
PS C:\Users\arvin\Downloads\ADA_Lab-main\ADA_Lab-main> .\a.exe
Enter no. of elemetns:
5
Enter elements:
91 4 2 6 3
Sorted array: 2  3  4  6  91
```

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

```c
#include <stdio.h>

#include <stdlib.h>



void swap (int *x, int *y){

    int temp = *x;

    *x = *y;

    *y = temp;
```

```
}


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

```c
}



int main (){

    int n;



    printf ("Enter number of elements: ");

    scanf ("%d", &n);

    int arr[n];



    printf ("Enter the elements: ");

    for (int i = 0; i < n; i++){

        scanf ("%d", &arr[i]);

    }

    heapsort (arr, n);

    printf ("Sorted elements: ");

    for (int i=0; i<n; i++){

        printf ("%d  ", arr[i]);

    }

    printf ("\n");

    return 0;

}
```

```
PS C:\Users\arvin\Downloads\ADA_Lab-main\ADA_Lab-main> gcc .\heapsort.c
PS C:\Users\arvin\Downloads\ADA_Lab-main\ADA_Lab-main> .\a.exe
Enter number of elements: 5
Enter the elements: 6 3 2 7 5
Sorted elements: 2  3  5  6  7
```

Implement 0/1 Knapsack problem using dynamic programming.

```c
#include <stdio.h>

int max(int a, int b) {
    return (a > b) ? a : b;
}
int knapsack(int w, int n, int p[], int weights[]) {
    int v[n+1][w+1];
    for (int i = 0; i <= n; i++) {
        for (int j = 0; j <= w; j++) {
            if (i == 0 || j == 0) {
                v[i][j] = 0;
            } else if (weights[i - 1] > j) {
                v[i][j] = v[i - 1][j];
            } else {
   v[i][j] = max(v[i - 1][j], v[i - 1][j - weights[i - 1]] + p[i - 1]);
            }
        }
    }
    return v[n][w];
}
int main() {
    int weights[6], profit[6], n, i;
    int capacity = 5;

    printf ("Enter the number of items:\n");
    scanf ("%d", &n);
    printf ("Enter the weights and profit:\n");
    for (i=0; i<n; i++){
        scanf ("%d %d", &weights[i], &profit[i]);
    }
    int maxProfit = knapsack(capacity, n, profit, weights);

    printf("Maximum Profit: %d\n", maxProfit);

    return 0;
}
```

```
Enter the number of items
4
Enter the weight and profit of each item
2 12
1 10
3 20
2 15
Enter the knapsack capacity 5
The knapsack table
0       0       0       0       0       0
0       0       12      12      12      12
0       10      12      22      22      22
0       10      12      22      30      32
0       10      15      25      30      37
Items designated at
1       1       0       1
```

Implement All Pair Shortest paths problem using Floyd's algorithm.

```c
#include <stdio.h>

#include <stdlib.h>

#include <limits.h>

int min (int a, int b) {

    return a < b ? a : b;

}

int main(){

    int n, graph[10][10], i, j, k;

    printf("Enter the number of vertices:\n");

    scanf("%d",&n);

    printf("Enter the weighs of graph in the form of an adjecency matrix:\n");

    for (int i=0; i<n; i++){

        for (int j=0; j<n; j++) {

            scanf ("%d", &graph[i][j]);

            if (i==j) graph[i][j]=0;

            else if (graph[i][j]==0) graph[i][j] = INT_MAX/3;

        }
```

```
    }

    for (k=0; k<n; k++) {

        for (i=0; i<n; i++){

            for (j=0; j<n; j++){

                graph[i][j] = min(graph[i][j], graph[i][k]+graph[k][j]);

            }

        }

    }

    for (i=0; i<n; i++){

        printf ("\n");

        for (j=0; j<n; j++) {

            printf ("%d  ", graph[i][j]);

        }

    }

    return 0;

}
```

```
Floyd's algorithm
enter the number of vertices
4
Enter the distance matrix for 4 vertices
0 999 3 999
2 0 999 999
999 7 0 1
6 999 999 0

Result
0       10      3       4
2       0       5       6
7       7       0       1
6       16      9       0
```

Find Minimum Cost Spanning Tree of a given undirected graph using Prim's and Kruskal's algorithm.

Prims:

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

int n;

int min(int key[], int vis[]){
    int min = INT_MAX, index;

    for (int i=0; i<n; i++){
        if (vis[i]==0 && key[i]<min){
            min=key[i];
            index = i;
        }
    }
    return index;
}

int main (){

    int i, j;

    printf ("Enter the number of nodes:\n");
    scanf ("%d",&n);
    int graph [n][n];
    printf ("Enter the graph in the form of adjecency matrix:\n");
    for(i=0; i<n; i++){
        for (j=0; j<n; j++){
            scanf ("%d", &graph[i][j]);
        }
    }

    int mst[n], key[n], vis[n];

    for (i=0; i<n; i++){
        key[i] = INT_MAX;
```

```
        vis[i] = 0;
    }

    key[0]=0;
    mst[0]=-1;

    for (i=0; i<n-1; i++){
        int x = min(key, vis);
        vis[x]=1;
        for (j=0; j<n; j++){
            if (graph[x][j] && vis[j]==0 && graph[x][j]<key[j]){
                mst[j]=x;
                key[j]=graph[x][j];
            }
        }
    }
    printf ("Graph:\n");
    for (i=1; i<n; i++){
        printf ("%d - %d   %d\n", mst[i], i, graph[i][mst[i]]);
    }

    return 0;
}
```

```
enter the number of vertices
5
enter the cost adjacency matrix
0 1 5 2 999
1 0 999 999 999
5 999 0 3 999
2 999 3 0 2
999 999 999 2 0
edges of spanning tree
1,2     1,4     4,5     4,3     weight=8
```

Kruskal's:

5

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


int comparator(const void* p1, const void* p2){
```

```
    const int(*x)[3] = p1;
    const int(*y)[3] = p2;

    return (*x)[2] - (*y)[2];
}


void makeSet(int parent[], int rank[], int n){
    for (int i = 0; i < n; i++) {
        parent[i] = i;
        rank[i] = 0;
    }
}


int findParent(int parent[], int component){
    if (parent[component] == component)
    return component;

    return parent[component]
= findParent(parent, parent[component]);
}


void unionSet(int u, int v, int parent[], int rank[], int n){

    u = findParent(parent, u);
    v = findParent(parent, v);

    if (rank[u] < rank[v]) {
        parent[u] = v;
    }
    else if (rank[u] > rank[v]) {
        parent[v] = u;
    }
    else {
        parent[v] = u;
        rank[u]++;
    }
}
```

```c
void kruskal(int n, int edge[n][3]){

    qsort(edge, n, sizeof(edge[0]), comparator);

    int parent[n];
    int rank[n];

    makeSet(parent, rank, n);

    int minCost = 0;

    printf("MST\n");
    for (int i = 0; i < n; i++) {
        int v1 = findParent(parent, edge[i][0]);
        int v2 = findParent(parent, edge[i][1]);
        int wt = edge[i][2];

        if (v1 != v2) {
            unionSet(v1, v2, parent, rank, n);
            minCost += wt;
            printf("%d - %d  %d\n", edge[i][0],
            edge[i][1], wt);
        }
    }

    printf("Cost: %d\n", minCost);
}


int main(){
    int n;
    printf("enter the number of edges\n");
    scanf("%d",&n);
    int edge[n][3];
    printf("enter the edges with (src, dest, wt) format\n");
    for(int i=0;i<n;i++)
    {
        for(int j=0;j<3;j++)
```

```
        {
            scanf("%d",&edge[i][j]);
        }
    }

    kruskal(n, edge);

    return 0;
}
```

```
Enter the number of nodes
6
Adjacency Matrix
999 3 1 6 999 999
3 999 5 999 3 999
1 5 999 5 6 4
6 999 5 999 999 2
999 3 6 999 999 6
999 999 4 2 6 999
Spanning tree
0 2
3 5
0 1
1 4
2 5
Cost is 13
```

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

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

int n;

int min (int dist[n], int flag[n]){
    int min = INT_MAX/2, index;
    for (int i=0; i<n; i++){
        if (flag[i]==0 && dist[i]<min){
            min=dist[i];
            index=i;
        }
    }
```

```
    return index;
}

void dijkstra (int graph[n][n], int src){

    int dist[n], flag[n], i;

    for (i=0; i<n; i++){
        dist[i]=INT_MAX/2;
        flag[i]=0;
    }
    dist[src]=0;

    for (i=0; i<n; i++){
        int u = min (dist, flag);

        flag[i]=1;
        for (int v=0; v<n; v++){
                if (flag[v]==0 && graph[u][v] && dist[u]!=INT_MAX/2 &&
dist[u]+graph[u][v] < dist[v])
                dist[v] = dist[u]+graph[u][v];
        }
    }
    printf ("Vertex  Distance\n");
    for (i=0; i<n; i++){
        printf ("%d \t %d\n", i, dist[i]);
    }
}

int main(){

    int i,j;
    printf ("Enter the number of vertices:\n");
    scanf ("%d", &n);
    int graph[n][n];
    printf ("Enter graph in the form of adjecency matrix:\n");
    for (i=0; i<n; i++){
        for (j=0; j<n; j++){
            scanf ("%d", &graph[i][j]);
        }
```

```
    }
    printf ("Enter the source node:\n");
    scanf ("%d", &i);
    dijkstra (graph, i);

    return 0;
}
```

```
Enter no. of vertices:6

Enter the adjacency matrix:
0 25 100 35 999 999
999 0 999 27 14 999
999 999 0 50 999 999
999 999 999 0 29 999
999 999 999 999 0 21
999 999 48 999 999 0

Enter the starting node:0

Distance of node1 = 25
Path = 1<-0
Distance of node2 = 100
Path = 2<-0
Distance of node3 = 35
Path = 3<-0
Distance of node4 = 39
Path = 4<-1<-0
Distance of node5 = 60
Path = 5<-4<-1<-0
```

Implement “N-Queens Problem” using Backtracking.

```
#include <stdio.h>

#define N 8
int board[N][N];

int isSafe(int row, int col) {
    int i, j;

    for (i = 0; i < col; i++) {
        if (board[row][i]) {
            return 0;
```

```
        }
    }

    for (i = row, j = col; i >= 0 && j >= 0; i--, j--) {
        if (board[i][j]) {
            return 0;
        }
    }

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

    return 1;
}

int solveNQueens(int col) {
    if (col >= N) {
        return 1;
    }

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

            if (solveNQueens(col + 1)) {
                return 1;
            }

            board[i][col] = 0;
        }
    }

    return 0;
}

int main() {
```

```
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            board[i][j] = 0;
        }
    }
    if (solveNQueens(0)) {

        for (int i = 0; i < N; i++) {
            for (int j = 0; j < N; j++) {
                printf("%d ", board[i][j]);
            }
            printf("\n");
        }
    }
    else {
        printf("No solution exists!\n");
    }

    return 0;
}
```

```
PS C:\Users\arvin\Downloads\ADA_Lab-main\ADA_Lab-main> gcc .\queens.c
PS C:\Users\arvin\Downloads\ADA_Lab-main\ADA_Lab-main> .\a.exe
1 0 0 0 0 0 0 0
0 0 0 0 0 0 1 0
0 0 0 0 1 0 0 0
0 0 0 0 0 0 0 1
0 1 0 0 0 0 0 0
0 0 0 1 0 0 0 0
0 0 0 0 0 1 0 0
0 0 1 0 0 0 0 0
```

```
0 0 1 0
1 0 0 0
0 0 0 1
0 1 0 0
```