VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT
On

ANALYSIS AND DESIGN OF ALGORITHMS (23CS4PCADA)

Submitted by

THILAK K (1WA23CS021)

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
February-May 2025

B. M. S. College of Engineering,

Bull Temple Road, Bangalore 560019
(Affiliated To Visvesvaraya Technological University, Belgaum)
Department of Computer Science and Engineering

This is to certify that the Lab work entitled **"ANALYSIS AND DESIGN OF ALGORITHMS"** carried out by **THILAK K (1WA23CS021),**who is 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 year 2024-25. The Lab report has been approved as it satisfies the academic requirements in respect of Analysis and Design of Algorithms Lab - **(23CS4PCADA)** work prescribed for the said degree.

Pradeep Sadanand
Assistant Professor
Department of CSE
BMSCE, Bengaluru

Dr. Kavitha Sooda
Professor and Head
Department of CSE
BMSCE, Bengaluru

Index Sheet

Course outcomes:

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.

Lab program 1:

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

i)using dfs

CODE:

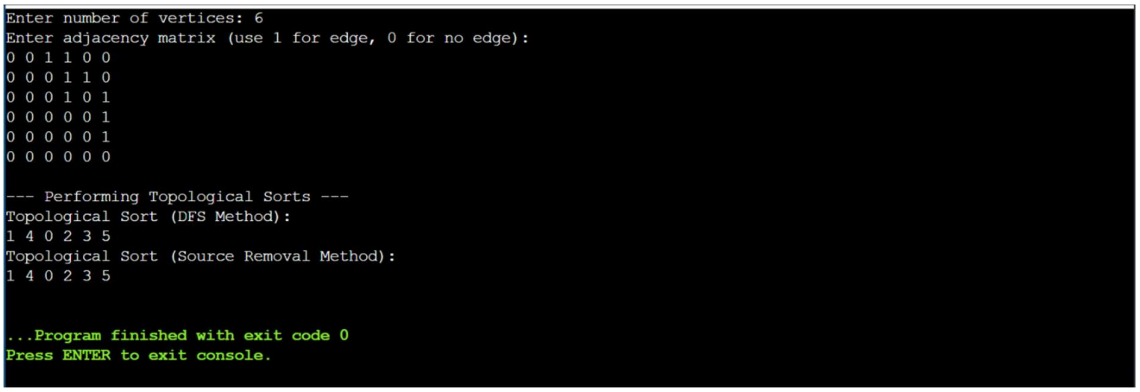
```
#include <stdio.h>

#define MAX 100

int visited[MAX], res[MAX], j = -1;  // For DFS method
int indegree[MAX], stack[MAX], top = -1;  // For Source Removal method

void dfs(int u, int adj[MAX][MAX], int n) {
   visited[u] = 1;
   for (int v = 0; v < n; v++) {
      if (adj[u][v] == 1 && visited[v] == 0) {
         dfs(v, adj, n);
      }
   }
   j++;
   res[j] = u; // Store node after exploring all its neighbors
}

void topologicalSortDFS(int adj[MAX][MAX], int n) {
   // Initialize visited array
   for (int i = 0; i < n; i++) {
      visited[i] = 0;
   }
   j = -1; // Reset result index

   // Perform DFS from all unvisited nodes
   for (int i = 0; i < n; i++) {
      if (visited[i] == 0) {
```

```
            dfs(i, adj, n);
        }
    }

    printf("Topological Sort (DFS Method):\n");
    for (int i = j; i >= 0; i--) {
        printf("%d ", res[i]);
    }
    printf("\n");
}

void topologicalSortSourceRemoval(int adj[MAX][MAX], int n) {
    top = -1;
    int result[MAX], t = 0;

    // Calculate in-degree of each vertex
    for (int j = 0; j < n; j++) {
        int sum = 0;
        for (int i = 0; i < n; i++) {
            sum += adj[i][j];
        }
        indegree[j] = sum;
    }

    // Push vertices with in-degree 0 to stack
    for (int i = 0; i < n; i++) {
        if (indegree[i] == 0) {
            stack[++top] = i;
        }
    }

    while (top != -1) {
```

```
        int u = stack[top--];
        result[t++] = u;

        for (int v = 0; v < n; v++) {
            if (adj[u][v] == 1) {
                indegree[v]--;
                if (indegree[v] == 0) {
                    stack[++top] = v;
                }
            }
        }
    }

    // Check if topological sorting was successful (no cycles)
    if (t != n) {
        printf("The graph has a cycle. Topological sort not possible.\n");
    } else {
        printf("Topological Sort (Source Removal Method):\n");
        for (int i = 0; i < t; i++) {
            printf("%d ", result[i]);
        }
        printf("\n");
    }
}

int main() {
    int n;
    int adj[MAX][MAX];

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

```
    printf("Enter adjacency matrix (use 1 for edge, 0 for no edge):\n");
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            scanf("%d", &adj[i][j]);
        }
    }

    printf("\n--- Performing Topological Sorts ---\n");
    topologicalSortDFS(adj, n);
    topologicalSortSourceRemoval(adj, n);

    return 0;
}
```

OUTPUT:



Lab program 2:

Implement Johnson Trotter algorithm to generate permutations.

CODE:

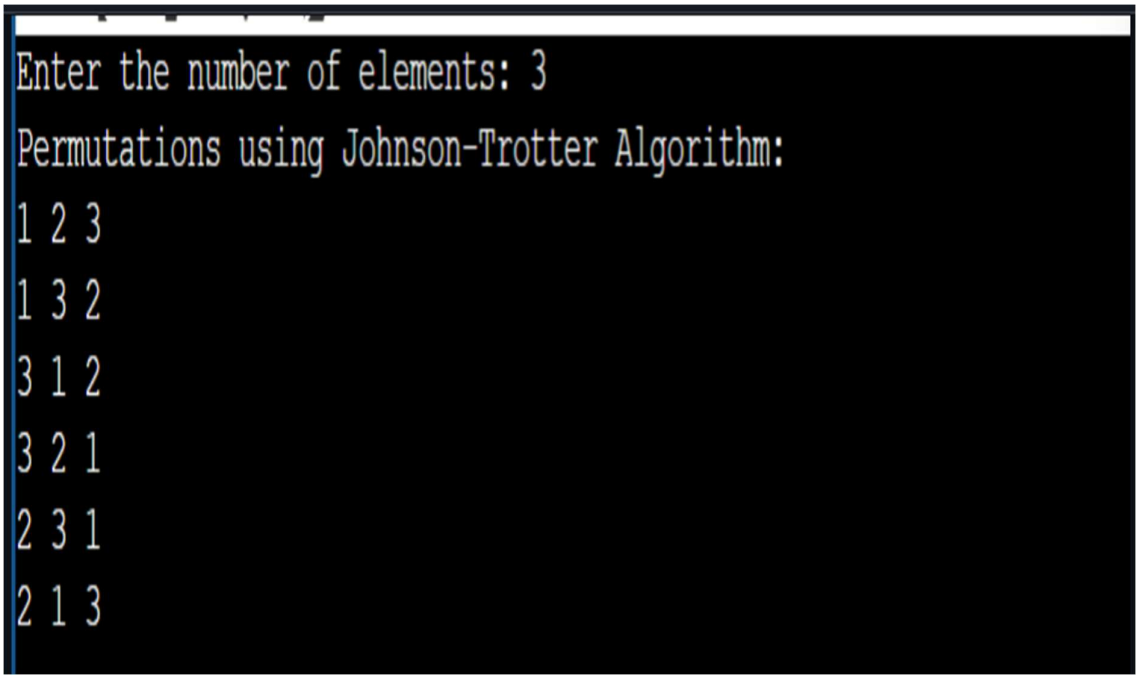
```
#include <stdio.h>

#define MAX 100
#define LEFT -1
#define RIGHT 1

typedef struct {
    int value;
    int dir; // -1 for LEFT, 1 for RIGHT
} Element;

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

int getMobile(Element perm[], int n) {
    int mobile = 0;
    for (int i = 0; i < n; i++) {
        int pos = i + perm[i].dir;
        if (pos >= 0 && pos < n && perm[i].value > perm[pos].value) {
            if (perm[i].value > mobile) {
                mobile = perm[i].value;
            }
```

```
        }
    }
    return mobile;
}

int findIndex(Element perm[], int n, int mobile) {
    for (int i = 0; i < n; i++) {
        if (perm[i].value == mobile)
            return i;
    }
    return -1;
}

void johnsonTrotter(int n) {
    Element perm[MAX];

    // Step 1: Initialize the first permutation
    for (int i = 0; i < n; i++) {
        perm[i].value = i + 1;
        perm[i].dir = LEFT;
    }

    printPermutation(perm, n); // Print the first permutation

    while (1) {
        int mobile = getMobile(perm, n);
        if (mobile == 0) break; // No mobile integer, we're done

        int pos = findIndex(perm, n, mobile);
```

```
        int swapWith = pos + perm[pos].dir;

        // Step 2: Swap mobile element with the adjacent one it points to
        Element temp = perm[pos];
        perm[pos] = perm[swapWith];
        perm[swapWith] = temp;

        pos = swapWith;

        // Step 3: Reverse the direction of all elements greater than mobile
        for (int i = 0; i < n; i++) {
            if (perm[i].value > mobile) {
                perm[i].dir = -perm[i].dir;
            }
        }

        printPermutation(perm, n);
    }
}

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

    printf("Permutations using Johnson-Trotter Algorithm:\n");
    johnsonTrotter(n);

    return 0;
```

}

OUTPUT:


Enter the number of elements: 3
Permutations using Johnson-Trotter Algorithm:
1 2 3
1 3 2
3 1 2
3 2 1
2 3 1
2 1 3

Lab program 3:

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>


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: ");
```

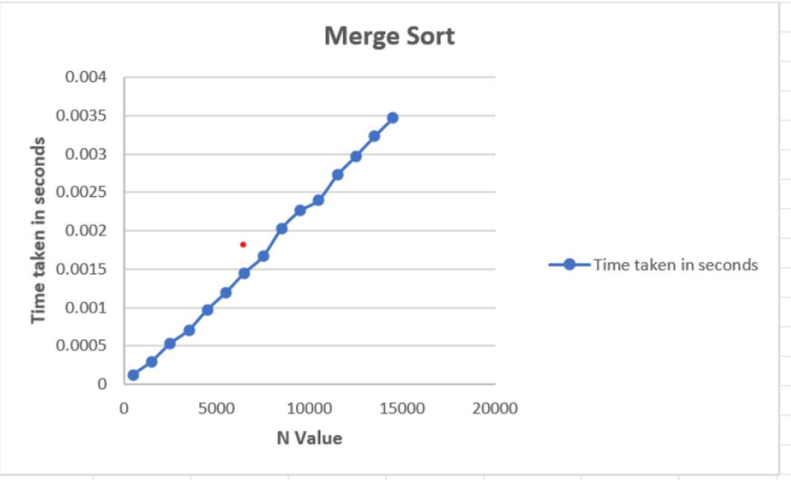
```
        scanf("%d", &n);
        printf("\nEnter array elements: ");
        for(i = 0; i < n; i++)
        {
          scanf("%d", &a[i]);
        }
        start = clock();
        split(a, 0, n - 1);
        end = clock();
        printf("\nSorted array is: ");
        for(i = 0; i < n; i++)
          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] = n - i;
          }
          start = clock();
          split(a, 0, n - 1);
          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:**

N Value	Time taken in seconds
500	0.000128
1500	0.0003
2500	0.000532
3500	0.000707
4500	0.000973
5500	0.001197
6500	0.001452
7500	0.001668
8500	0.00203
9500	0.002261
10500	0.002404
11500	0.002728
12500	0.002974
13500	0.003232
14500	0.003469



Lab program 4:

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

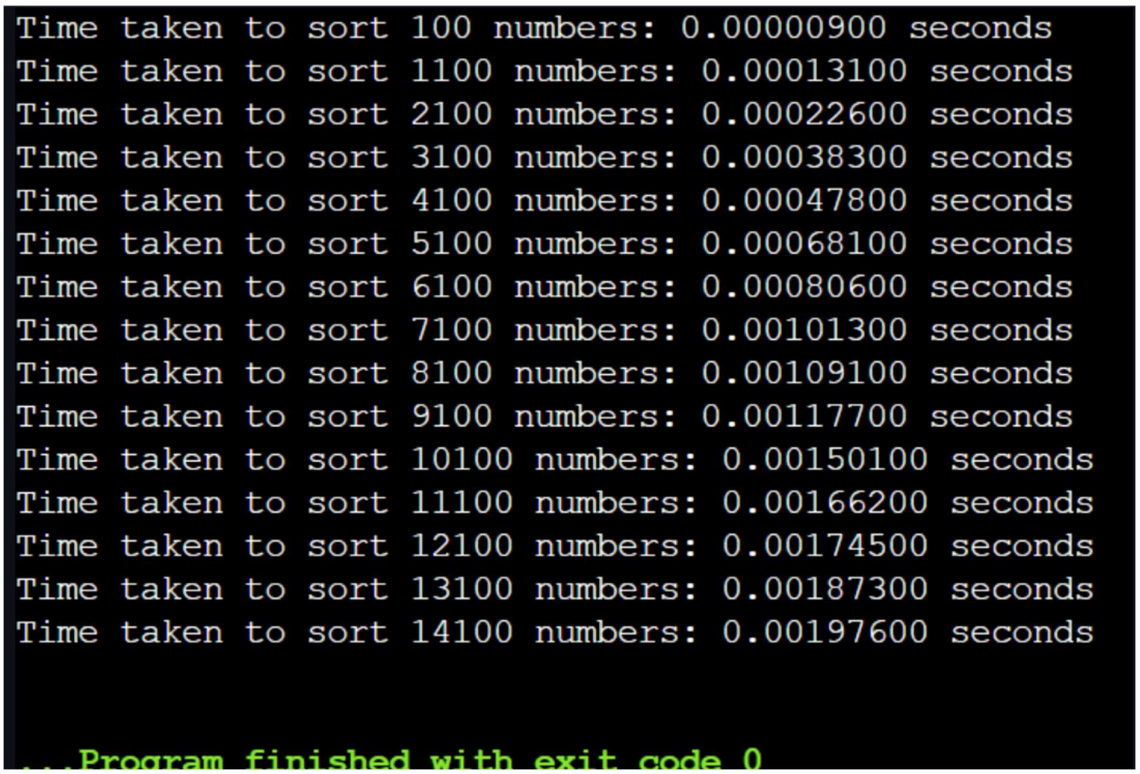
CODE:

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

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

int partition(int arr[], int low, int high)
{
   int pivot = arr[high];
   int i = low - 1;
   for (int j = low; j < high; j++)
   {
      if (arr[j] < pivot)
      {
         i++;
         swap(&arr[i], &arr[j]);
      }
   }
   swap(&arr[i + 1], &arr[high]);
   return i + 1;
}

void quickSort(int arr[], int low, int high)
{
```

```
    if (low < high)
    {
        int pi = partition(arr, low, high);
        quickSort(arr, low, pi - 1);
        quickSort(arr, pi + 1, high);
    }
}

int main()
{
    srand(time(NULL));
    clock_t start, end;
    int arr[15000];
    int n = 100;

    while (n <= 14500)
    {
        for (int i = 0; i < n; i++)
        {
            arr[i] = rand() % 10000;
        }

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

        double time_taken = ((double)(end - start)) / CLOCKS_PER_SEC;
        printf("Time taken to sort %d numbers: %0.8f seconds\n", n, time_taken);

        n += 1000;
    }
```

```
    return 0;
}
```

OUTPUT:



Lab program 5:

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

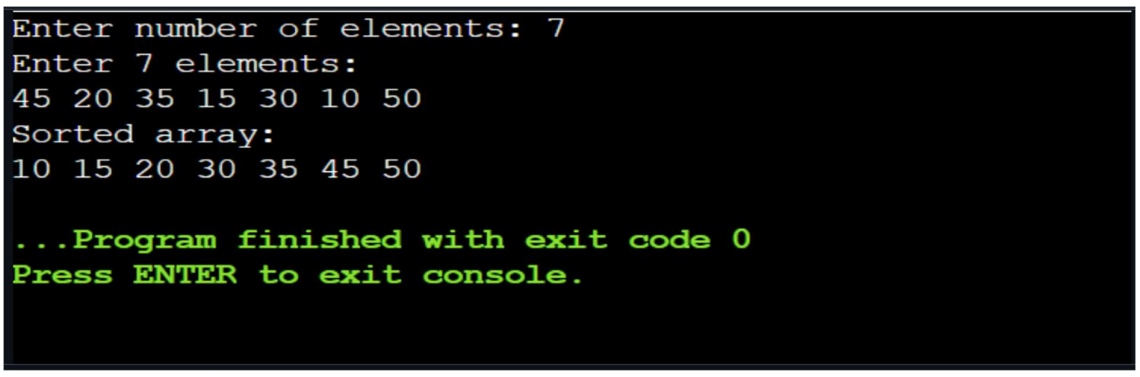
CODE:

```
#include <stdio.h>


#define MAX 100


void topDownHeapify(int a[], int n) {
    for (int k = 1; k < n; k++) {
        int item = a[k];
        int c = k;
        int p = (c - 1) / 2;
        while (c > 0 && item > a[p]) {
            a[c] = a[p];
            c = p;
            p = (c - 1) / 2;
        }
        a[c] = item;
    }
}


void bottomUpHeapify(int a[], int n) {
    for (int p = (n - 1) / 2; p >= 0; p--) {
        int item = a[p];
        int c = 2 * p + 1;


        while (c < n) {
            if (c + 1 < n && a[c] < a[c + 1]) {
                c = c + 1;
```

```
            }

            if (item < a[c]) {
                a[p] = a[c];
                p = c;
                c = 2 * p + 1;
            } else {
                break;
            }
        }
        a[p] = item;
    }
}

void heapSort(int a[], int n) {
    // Step 1: Build the heap using top-down
    topDownHeapify(a, n);

    // Step 2: Repeatedly remove max and fix heap using bottom-up
    for (int i = n - 1; i > 0; i--) {
        // Swap max (a[0]) with last element
        int temp = a[0];
        a[0] = a[i];
        a[i] = temp;

        // Recreate heap on reduced array
        bottomUpHeapify(a, i);
    }
}
```

```
int main() {
    int a[MAX], n;

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

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

    heapSort(a, n);

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

    return 0;
}
```

OUTPUT:



Lab program 6:

Implement 0/1 Knapsack problem using dynamic programming.

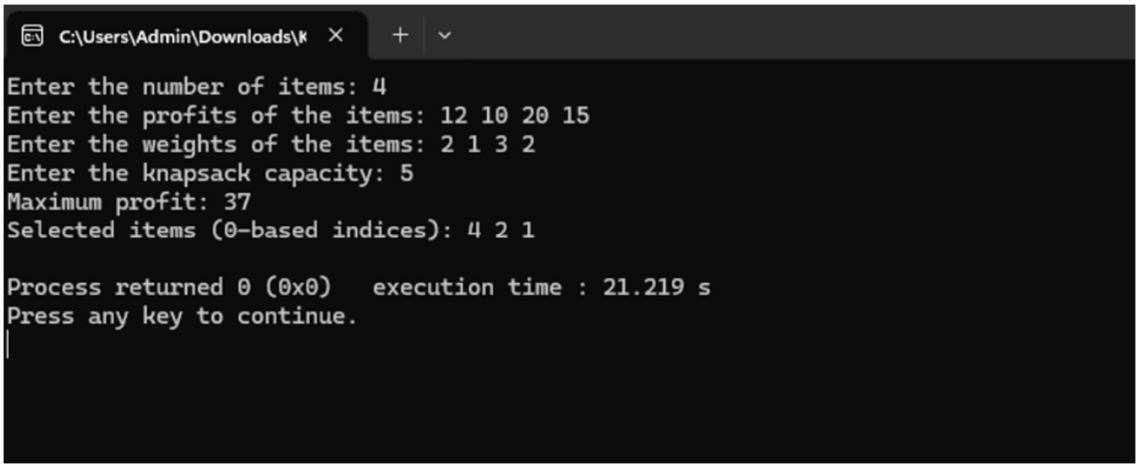
CODE:

```
#include <stdio.h>

#define MAX 100

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

void knapsack(int n, int W, int wt[], int val[]) {
   int F[MAX][MAX]; // DP table

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

   // Maximum value that can be put in knapsack of capacity W
   printf("Maximum profit: %d\n", F[n][W]);
```

```
    // To print the selected items (optional)
    int res = F[n][W];
    int w = W;
    printf("Selected items (0-based indices): ");
    for (int i = n; i > 0 && res > 0; i--) {
        if (res == F[i - 1][w])
            continue; // item i-1 not included
        else {
            printf("%d ", i); // item i-1 included
            res -= val[i - 1];
            w -= wt[i - 1];
        }
    }
    printf("\n");
}

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

    int val[n], wt[n];

    printf("Enter the profits of the items: ");
    for (int i = 0; i < n; i++)
        scanf("%d", &val[i]);

    printf("Enter the weights of the items: ");
    for (int i = 0; i < n; i++)
```

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

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

    knapsack(n, W, wt, val);

    return 0;
}
```

OUTPUT:



Lab program 7:

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

CODE:

```
#include <stdio.h>

int a[10][10], D[10][10], n;

void floyd(int [][10], int);
int min(int, int);

int main() {
   int i, j;

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

   printf("Enter the cost adjacency matrix:\n");
   for (i = 0; i < n; i++) {
      for (j = 0; j < n; j++) {
         scanf("%d", &a[i][j]);
      }
   }

   floyd(a, n);

   printf("Distance Matrix:\n");
   for (i = 0; i < n; i++) {
      for (j = 0; j < n; j++) {
```

```
            printf("%d ", D[i][j]);
        }
        printf("\n");
    }

    return 0;
}

void floyd(int a[][10], int n) {
    int i, j, k;

    // Initialize distance matrix
    for (i = 0; i < n; i++) {
        for (j = 0; j < n; j++) {
            D[i][j] = a[i][j];
        }
    }

    // Floyd-Warshall algorithm
    for (k = 0; k < n; k++) {
        for (i = 0; i < n; i++) {
            for (j = 0; j < n; j++) {
                D[i][j] = min(D[i][j], D[i][k] + D[k][j]);
            }
        }
    }
}

int min(int a, int b) {
```

```
    return (a < b) ? a : b;
}
```

OUTPUT:

Lab program 8:

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

CODE:

```
//This is Prim's Algorithm

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

#define MAX 100
#define INF 9999

int main()
{
   int G[MAX][MAX], i, j, n;
   int selected[MAX];
   int no_edge = 0;
   int x, y;

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

   printf("Enter the adjacency matrix (enter 0 if no edge):\n");

   for (i = 0; i < n; i++)
   {
      for (j = 0; j < n; j++)
      {
         scanf("%d", &G[i][j]);
         if (G[i][j] == 0)
```

```
        {
            G[i][j] = INF;
        }
    }
}

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

selected[0] = 1;

printf("\nEdge \tWeight\n");

while (no_edge < n - 1)
{
    int min = INF;
    x = 0;
    y = 0;

    for (i = 0; i < n; i++)
    {
        if (selected[i])
        {
            for (j = 0; j < n; j++)
            {
                if (!selected[j] && G[i][j] < min)
                {
```

```
                min = G[i][j];
                x = i;
                y = j;
            }
        }
    }

    printf("%d - %d\t%d\n", x, y, G[x][y]);
    selected[y] = 1;
    no_edge++;
  }

  return 0;
}
```

OUTPUT:

```
Enter the number of vertices: 4
Enter the adjacency matrix (enter 0 if no edge):
0 10 6 0
10 0 5 15
6 5 0 4
0 15 4 0

Edge    Weight
0 - 2   6
2 - 3   4
2 - 1   5


...Program finished with exit code 0
Press ENTER to exit console.
```

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

CODE:

```
//This is Kruskals Algorithm

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

#define MAX 100

int parent[MAX];

int find(int i)
{
   while (parent[i] != i)
   {
      i = parent[i];
   }
   return i;
}

void union_sets(int i, int j)
{
   int a = find(i);
   int b = find(j);
   parent[a] = b;
}

int main()
{
```

```
int n, i, j, u, v;
int a, b, weight;
int min, mincost = 0;
int cost[MAX][MAX];

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

printf("Enter the adjacency matrix (0 if no edge):\n");
for (i = 0; i < n; i++)
{
   for (j = 0; j < n; j++)
   {
      scanf("%d", &cost[i][j]);
      if (cost[i][j] == 0)
         cost[i][j] = 9999;
   }
}

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

int ne = 0;
printf("\nEdge \tWeight\n");

while (ne < n - 1)
{
   min = 9999;
   for (i = 0; i < n; i++)
```

```
        {
            for (j = 0; j < n; j++)
            {
                if (find(i) != find(j) && cost[i][j] < min)
                {
                    min = cost[i][j];
                    a = u = i;
                    b = v = j;
                }
            }
        }
        union_sets(u, v);
        printf("%d - %d\t%d\n", a, b, min);
        mincost += min;
        ne++;
    }
    printf("\nMinimum cost = %d\n", mincost);
    return 0;
}
```

OUTPUT:

```
Enter the number of vertices: 4
Enter the adjacency matrix (0 if no edge):
0 10 6 0
10 0 5 15
6 5 0 4
0 15 4 0

Edge     Weight
2 - 3    4
1 - 2    5
0 - 2    6

Minimum cost = 15


...Program finished with exit code 0
Press ENTER to exit console.
```

Lab program 9:

Implement Fractional Knapsack using Greedy technique.

CODE:

```
#include <stdio.h>

// Structure to hold item details
typedef struct {
    int weight;
    int profit;
    float ratio;
} Item;

// Function to swap two items
void swap(Item *a, Item *b) {
    Item temp = *a;
    *a = *b;
    *b = temp;
}

// Function to sort items by decreasing profit/weight ratio
void sortItems(Item items[], int n) {
    for(int i = 0; i < n - 1; i++) {
        for(int j = 0; j < n - i - 1; j++) {
            if(items[j].ratio < items[j + 1].ratio) {
                swap(&items[j], &items[j + 1]);
            }
        }
    }
}
```

```
// Fractional Knapsack Function
float fractionalKnapsack(Item items[], int n, int capacity) {
    sortItems(items, n);

    float totalProfit = 0.0;
    int currentWeight = 0;

    for(int i = 0; i < n; i++) {
        if(currentWeight + items[i].weight <= capacity) {
            // Take whole item
            currentWeight += items[i].weight;
            totalProfit += items[i].profit;
        } else {
            // Take fractional part
            int remain = capacity - currentWeight;
            totalProfit += (items[i].profit * ((float)remain / items[i].weight));
            break;
        }
    }

    return totalProfit;
}

int main() {
    int n, capacity;

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

```
    Item items[n];

    for(int i = 0; i < n; i++) {
        printf("Enter profit and weight of item %d: ", i + 1);
        scanf("%d %d", &items[i].profit, &items[i].weight);
        items[i].ratio = (float)items[i].profit / items[i].weight;
    }

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

    float maxProfit = fractionalKnapsack(items, n, capacity);
    printf("Maximum profit = %.2f\n", maxProfit);

    return 0;
}
```

OTUPUT:

```
Enter number of items: 3
Enter profit and weight of item 1: 30 20
Enter profit and weight of item 2: 40 25
Enter profit and weight of item 3: 35 10
Enter knapsack capacity: 40
Maximum profit = 82.50


...Program finished with exit code 0
Press ENTER to exit console.
```

Lab program 10:

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

CODE:

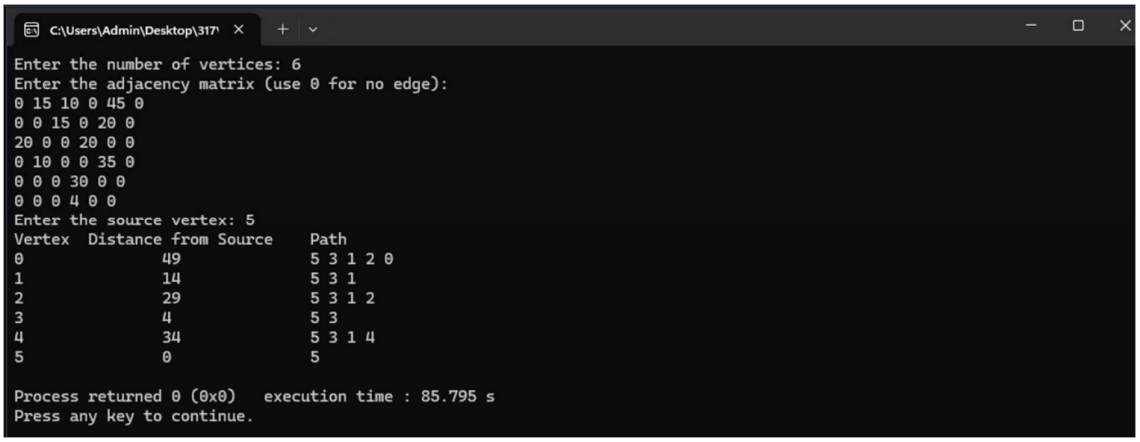
```
#include<stdio.h>

#include<limits.h>

#define INF 99999

#define MAX 100


int n;


int minDistance(int dist[],int visited[])

{

   int min=INF,min_index=-1;

   for(int v=0;v<n;v++)

   {

      if(dist[v]<=min && !visited[v])

      {

         min=dist[v];

         min_index=v;

      }

   }

   return min_index;

}


void dijkstra(int graph[MAX][MAX],int src)

{

   int dist[MAX],parent[MAX],visited[MAX];


   //initialise the matrices
```

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

dist[src]=0;
for(int count=0;count<n-1;count++)
{
   int u=minDistance(dist,visited);
   if(u==-1)break;
   visited[u]=1;

   for(int v=0;v<n;v++)
   {
      if(!visited[v]&&graph[u][v]&&dist[u]!=INF&&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\t%d\t\t", i, dist[i]);
   int path[MAX], path_len = 0;
```

```
        int temp = i;
        while (temp != -1)
        {
            path[path_len++] = temp;
            temp = parent[temp];
        }
        for (int j = path_len - 1; j >= 0; j--)
            printf("%d ", path[j]);
        printf("\n");
    }
}
int main() {
    printf("Enter the number of vertices: ");
    scanf("%d", &n);

    int graph[MAX][MAX];

    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]);
        }
    }

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

    dijkstra(graph, source);
```

}

OUTPUT:


C:\Users\Admin\Desktop\317
Enter the number of vertices: 6
Enter the adjacency matrix (use 0 for no edge):
0 15 10 0 45 0
0 0 15 0 20 0
20 0 0 20 0 0
0 10 0 0 35 0
0 0 0 30 0 0
0 0 0 4 0 0
Enter the source vertex: 5
Vertex Distance from Source Path
0 49 5 3 1 2 0
1 14 5 3 1
2 29 5 3 1 2
3 4 5 3
4 34 5 3 1 4
5 0 5
Process returned 0 (0x0) execution time : 85.795 s
Press any key to continue.

Lab program 11:

Implement “N-Queens Problem” using Backtracking.

CODE:

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

#define MAX 100

int board[MAX]; // board[row] = column where queen is placed
int n;

// Check if placing queen at board[row] = col is valid
int isSafe(int row) {
    for (int prev = 1; prev < row; prev++) {
        if (board[prev] == board[row] ||                    // same column
            abs(prev - row) == abs(board[prev] - board[row])) {  // same diagonal
            return 0;
        }
    }
    return 1;
}

// Solves the N-Queens problem using backtracking (iterative)
void solveNQueens() {
    int row = 1;
    board[row] = 0;

    while (row != 0) {
```

```
        board[row]++;

        while (board[row] <= n && !isSafe(row)) {
            board[row]++;
        }

        if (board[row] <= n) {
            if (row == n) {
                // Print one solution
                printf("Solution: ");
                for (int i = 1; i <= n; i++) {
                    printf("(%d,%d) ", i, board[i]);
                }
                printf("\n");
            } else {
                row++;
                board[row] = 0;
            }
        } else {
            row--; // Backtrack
        }
    }
}

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

    if (n < 1 || n > MAX) {
```

```
        printf("Invalid number of queens. Enter between 1 and %d.\n", MAX);
        return 1;
    }

    printf("Solutions for %d-Queens problem:\n", n);
    solveNQueens();

    return 0;
}
```

OUTPUT: