VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT
On

ANALYSIS AND DESIGN OF ALGORITHMS (23CS4PCADA)

Submitted by

Bhavya J Makadia (1BM23CS064)

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 **Bhavya J Makadia (1BM23CS064)**, 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 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.

Anusha S
Assistant Professor
Department of CSE
BMSCE, Bengaluru

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

Index

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.

GitHub link: https://github.com/BhoomiSuresh/ADA.git

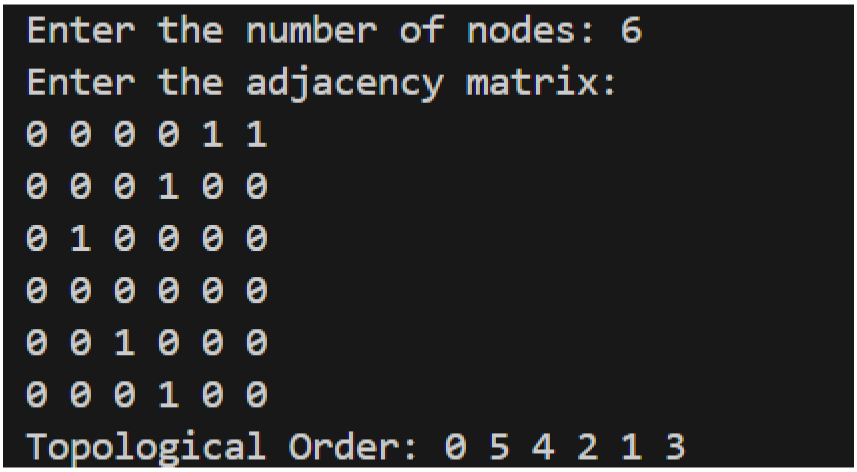
Lab program 1:

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

Code

```
#include <stdio.h>

int n, a[10][10], res[10], visited[10], top = 0;

void dfs(int node) {
    visited[node] = 1;
    for (int i = 0; i < n; i++) {
        if (a[node][i] == 1 && !visited[i]) {
            dfs(i);
        }
    }
    res[top++] = node;
}

void topologicalSort() {
    for (int i = 0; i < n; i++) {
        visited[i] = 0;
    }
    for (int i = 0; i < n; i++) {
        if (!visited[i]) {
            dfs(i);
        }
    }
}
int main() {
    printf("Enter the number of nodes: ");
    scanf("%d", &n);

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

    topologicalSort();

    printf("Topological Order: ");
    for (int i = top - 1; i >= 0; i--) {
        printf("%d ", res[i]);
    }
    printf("\n");

    return 0;
}
```

Screenshot of Output



```
Enter the number of nodes: 6
Enter the adjacency matrix:
0 0 0 0 1 1
0 0 0 1 0 0
0 1 0 0 0 0
0 0 0 0 0 0
0 0 1 0 0 0
0 0 0 1 0 0
Topological Order: 0 5 4 2 1 3
```

LeetCode Program related to Topological sorting

Code

```
bool canFinish(int numCourses, int** prerequisites, int prerequisitesSize, int* prerequisitesColSize) {

    N* adjList[numCourses];

    int visited[numCourses];

    int recStack[numCourses];


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

        visited[i]=0;

        recStack[i]=0;

    }
```

Lab program 2:

Implement Johnson Trotter algorithm to generate permutations.

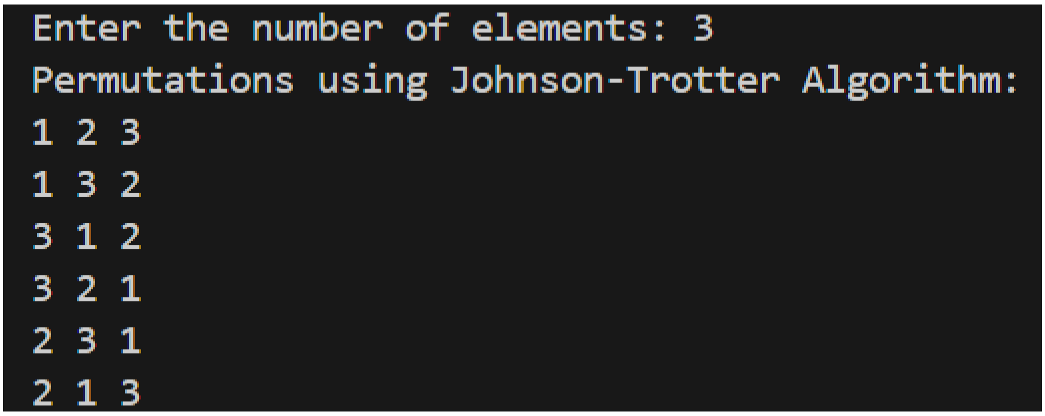
Code

```
#include <stdio.h>

#include <stdlib.h>

#define LEFT -1

#define RIGHT 1


// Function to print a permutation

void printPermutation(int perm[], int n) {

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

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

    }

    printf("\n");

}
```

```
// Function to find the largest mobile element
int getMobile(int perm[], int dir[], int n) {
    int mobile = 0;
    for (int i = 0; i < n; i++) {
        if ((dir[i] == LEFT && i != 0 && perm[i] > perm[i - 1]) ||
            (dir[i] == RIGHT && i != n - 1 && perm[i] > perm[i + 1])) {
            if (perm[i] > mobile) {
                mobile = perm[i];
            }
        }
    }
    return mobile;
}

// Function to get the index of the mobile element
int findIndex(int perm[], int n, int mobile) {
    for (int i = 0; i < n; i++) {
        if (perm[i] == mobile)
            return i;
    }
    return -1;
}

// Main function implementing Johnson-Trotter
void johnsonTrotter(int n) {
    int perm[n];
    int dir[n];
```

```
// Initialize permutation and directions
for (int i = 0; i < n; i++) {
    perm[i] = i + 1;
    dir[i] = LEFT;
}

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

for (int count = 1; count < 1 << n; count++) {
    int mobile = getMobile(perm, dir, n);
    if (mobile == 0) break;

    int pos = findIndex(perm, n, mobile);
    int swapPos = (dir[pos] == LEFT) ? pos - 1 : pos + 1;

    // Swap the mobile element with the adjacent in its direction
    int temp = perm[pos];
    perm[pos] = perm[swapPos];
    perm[swapPos] = temp;

    int tempDir = dir[pos];
    dir[pos] = dir[swapPos];
    dir[swapPos] = tempDir;

    pos = swapPos;

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

        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;
}
```

Screenshot of 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
```

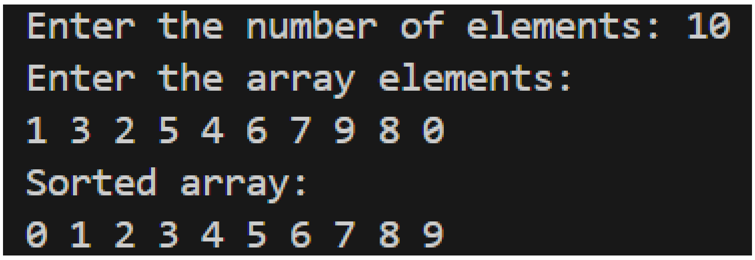
Lab program 3:

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>

void merge_sort(int a[], int low, int high);
void simple_sort(int a[], int low, int mid, int high);

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

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

    int a[n];  // Variable length array

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

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

    time_taken = (double)(end - start) / CLOCKS_PER_SEC;

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

    printf("Time taken to sort: %f seconds\n", time_taken);

    return 0;
}

void merge_sort(int a[], int low, int high) {
    if (low < high) {
        int mid = (low + high) / 2;
        merge_sort(a, low, mid);
        merge_sort(a, mid + 1, high);
        simple_sort(a, low, mid, high);
    }
}

void simple_sort(int a[], int low, int mid, int high) {
    int i = low, j = mid + 1, k = low;
    int c[high + 1];  // Temporary array for merging

    while (i <= mid && j <= high) {
        if (a[i] < a[j]) {
            c[k++] = a[i++];
        } else {
            c[k++] = a[j++];
        }
    }

    while (i <= mid) {
        c[k++] = a[i++];
    }

    while (j <= high) {
        c[k++] = a[j++];
    }

    for (i = low; i <= high; i++) {
        a[i] = c[i];
    }
}
```

Screenshot of Output



```
Enter the number of elements: 10
Enter the array elements:
1 3 2 5 4 6 7 9 8 0
Sorted array:
0 1 2 3 4 5 6 7 8 9
```

LeetCode Program related to sorting.

Code

```
def countRangeSum(self, nums, lower, upper):
    first = [0]
    for num in nums:
        first.append(first[-1] + num)
    def sort(lo, hi):
        mid = (lo + hi) / 2
        if mid == lo:
            return 0
        count = sort(lo, mid) + sort(mid, hi)
        i = j = mid
        for left in first[lo:mid]:
            while i < hi and first[i] - left <  lower: i += 1
            while j < hi and first[j] - left <= upper: j += 1
            count += j - i
        first[lo:hi] = sorted(first[lo:hi])
        return count
    return sort(0, len(first))
```

Lab program 4:

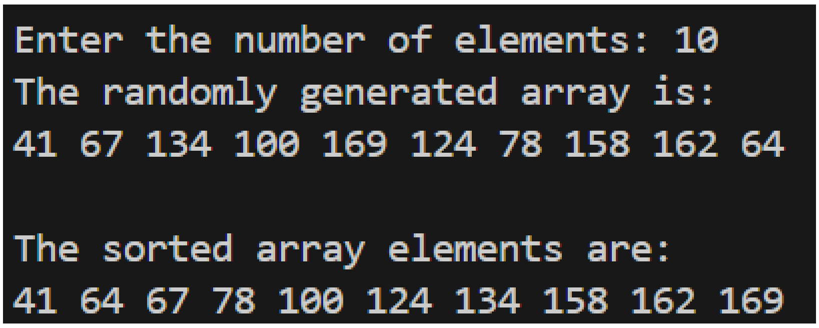
Sort a given set of N integer elements using Quick Sort technique and compute its time taken.

Code

```
#include <stdio.h>

#include <stdlib.h>

#include <time.h>


#define MAX 5000


void quicksort(int[], int, int);

int partition(int[], int, int);


int main() {

  int i, n, a[MAX], ch = 1;

  clock_t start, end;

  double time_taken;


  while (ch) {

    printf("\nEnter the number of elements: ");

    scanf("%d", &n);


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

      a[i] = rand() % 200;


    printf("The randomly generated array is:\n");

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

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

    printf("\n");
```

```
        start = clock();
        quicksort(a, 0, n - 1);
        end = clock();

        printf("\nThe sorted array elements are:\n");
        for (i = 0; i < n; i++)
            printf("%d ", a[i]);
        printf("\n");

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

        printf("\nDo you wish to continue? (0 = No, 1 = Yes): ");
        scanf("%d", &ch);
    }

    return 0;
}

void quicksort(int a[], int low, int high) {
    int mid;
    if (low < high) {
        mid = partition(a, low, high);
        quicksort(a, low, mid - 1);
        quicksort(a, mid + 1, high);
    }
}

int partition(int a[], int low, int high) {
    int key = a[low];
    int i = low + 1;
    int j = high;
    int temp;

    while (i <= j) {
        while (i <= high && a[i] <= key)
            i++;
        while (a[j] > key)
            j--;
        if (i < j) {
            temp = a[i];
            a[i] = a[j];
            a[j] = temp;
        } else {
            temp = a[j];
            a[j] = a[low];
            a[low] = temp;
        }
    }
    return j;
}
```

Screenshot of Output



```
Enter the number of elements: 10
The randomly generated array is:
41 67 134 100 169 124 78 158 162 64

The sorted array elements are:
41 64 67 78 100 124 134 158 162 169
```

LeetCode Program related to sorting.

Code

```
class Solution:
    def findKthLargest(self, nums: List[int], k: int) -> int:
        maxHeap = nums
        for i in range(len(maxHeap)):
            maxHeap[i] = -maxHeap[i]
        heapify(maxHeap)
        for i in range(k-1):
            heappop(maxHeap)
        return -maxHeap[0]
```

Lab program 5:

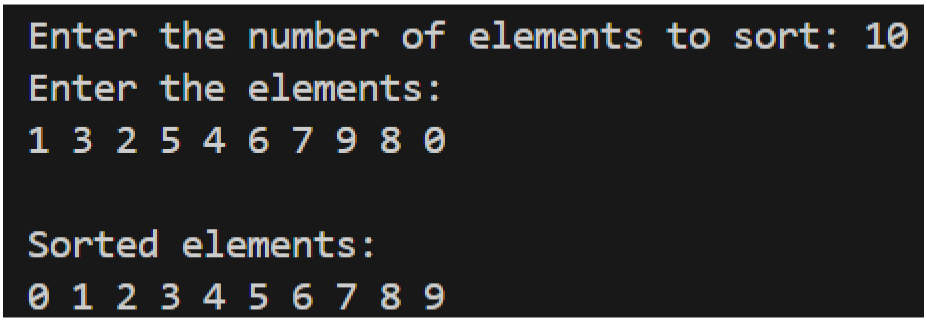
Sort a given set of N integer elements using Heap Sort technique and compute its time taken.

Code

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

// Heapify subtree rooted at index i in array a[], size n
void heapify(int a[], int n, int i) {
    int largest = i;        // Initialize largest as root
    int left = 2 * i + 1;   // left child index
    int right = 2 * i + 2;  // right child index

    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); // Recursively heapify affected subtree
    }
}

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

    // Extract elements one by one from heap
    for (int i = n - 1; i > 0; i--) {
        // Move current root to end
        int temp = a[0];
        a[0] = a[i];
        a[i] = temp;

        // call max heapify on the reduced heap
        heapify(a, i, 0);
    }
}
```

```
int main() {
    int n, choice = 1;

    while (choice) {
        printf("\nEnter the number of elements to sort: ");
        scanf("%d", &n);
        int a[n];

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

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

        printf("\nSorted elements:\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: %lf seconds\n", time_taken);

        printf("Do you wish to run again? (1/0): ");
        scanf("%d", &choice);
    }

    return 0;
}
```

Screenshot of Output



```
Enter the number of elements to sort: 10
Enter the elements:
1 3 2 5 4 6 7 9 8 0

Sorted elements:
0 1 2 3 4 5 6 7 8 9
```

Lab program 6:

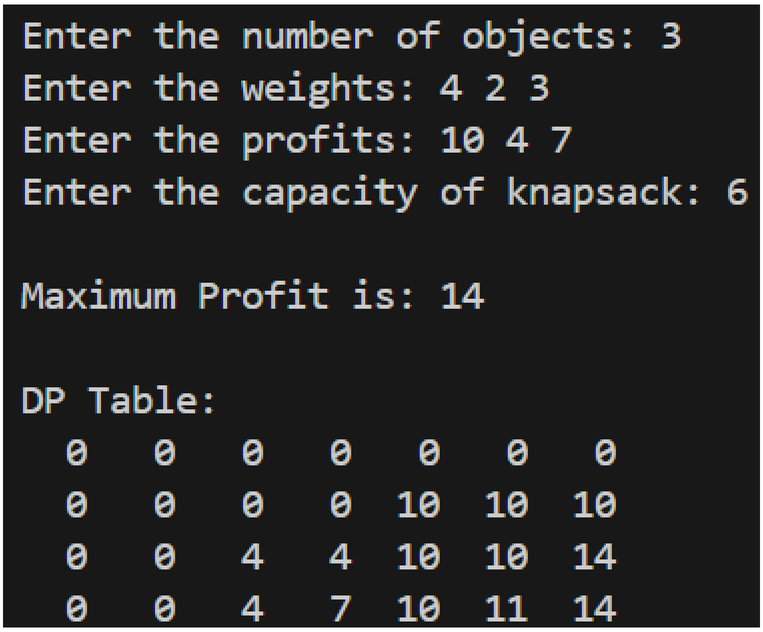
Implement 0/1 Knapsack problem using dynamic programming.

Code

```
#include <stdio.h>


int i, j, n, c, w[10], p[10], v[10][10];


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


void knapsack(int n, int w[10], int p[10], int c) {
   for (i = 0; i <= n; i++) {
      for (j = 0; j <= c; j++) {
         if (i == 0 || j == 0)
            v[i][j] = 0;
         else if (w[i] > j)
            v[i][j] = v[i - 1][j];
         else
            v[i][j] = max(v[i - 1][j], v[i - 1][j - w[i]] + p[i]);
```

```
        }
    }
    printf("\nMaximum Profit is: %d\n", v[n][c]);
    // Optional: Print the DP table
    printf("\nDP Table:\n");
    for (i = 0; i <= n; i++) {
        for (j = 0; j <= c; j++) {
            printf("%3d ", v[i][j]);
        }
        printf("\n");
    }
}

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

    printf("Enter the weights: ");
    for (i = 1; i <= n; i++) {
        scanf("%d", &w[i]);
    }

    printf("Enter the profits: ");
    for (i = 1; i <= n; i++) {
        scanf("%d", &p[i]);
    }

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

    knapsack(n, w, p, c);
    return 0;
}
```

Screenshot of Output



```
Enter the number of objects: 3
Enter the weights: 4 2 3
Enter the profits: 10 4 7
Enter the capacity of knapsack: 6

Maximum Profit is: 14

DP Table:
   0   0   0   0   0   0   0
   0   0   0   0  10  10  10
   0   0   4   4  10  10  14
   0   0   4   7  10  11  14
```

LeetCode Program related to Knapsack problem or Dynamic Programming.

Code

```
class Solution:
    def maxSizeSlices(self, slices: List[int]) -> int:
        n = len(slices) // 3
        def linear(arr):
            eat = [[0] + [-math.inf]*n] * 2
            for x in arr:
                eat.append([i and max(eat[-1][i], eat[-2][i-1]+x) for i in range(n+1)])
            return max(l[n] for l in eat)
        return max(linear(slices[1:]), linear(slices[:-1]))
```

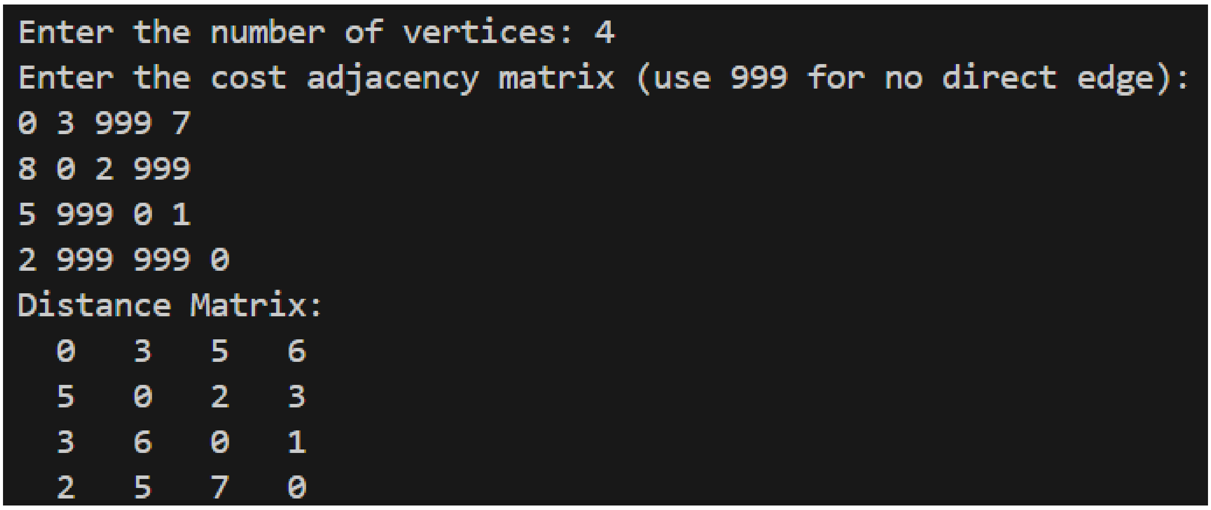
Lab program 7:

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

Code

```
#include <stdio.h>


#define INF 999  // Representation of infinity


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 number of vertices: ");

    scanf("%d", &n);


    printf("Enter the cost adjacency matrix (use 999 for no direct edge):\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++) {
            if (D[i][j] == INF)
                printf("INF ");
            else
                printf("%3d ", D[i][j]);
        }
        printf("\n");
    }
    return 0;
}

void floyd(int a[][10], int n) {
    int i, j, k;
    // Initialize the distance matrix with input adjacency 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++) {
                if (D[i][k] + D[k][j] < D[i][j]) {
                    D[i][j] = D[i][k] + D[k][j];
                }
            }
        }
    }
}
int min(int a, int b) {
    return (a < b) ? a : b;
}
```

Screenshot of Output



```
Enter the number of vertices: 4
Enter the cost adjacency matrix (use 999 for no direct edge):
0 3 999 7
8 0 2 999
5 999 0 1
2 999 999 0
Distance Matrix:
  0   3   5   6
  5   0   2   3
  3   6   0   1
  2   5   7   0
```

LeetCode Program related to shortest distance calculation.

Code

```
import heapq
from collections import defaultdict
import sys

def countPaths(n, roads):
    mod = 10**9 + 7
    adj = defaultdict(list)
    for u, v, t in roads:
        adj[u].append((v, t))
        adj[v].append((u, t))

    shortesttime = [sys.maxsize] * n
    cnt = [0] * n
    pq = [(0, 0)]  # (time, node)

    shortesttime[0] = 0
    cnt[0] = 1

    while pq:
        time, node = heapq.heappop(pq)

        if time > shortesttime[node]:
            continue

        for nbr, rtime in adj[node]:
            if time + rtime < shortesttime[nbr]:
                shortesttime[nbr] = time + rtime
                cnt[nbr] = cnt[node]
                heapq.heappush(pq, (shortesttime[nbr], nbr))
            elif time + rtime == shortesttime[nbr]:
                cnt[nbr] = (cnt[nbr] + cnt[node]) % mod

    return cnt[-1]
```

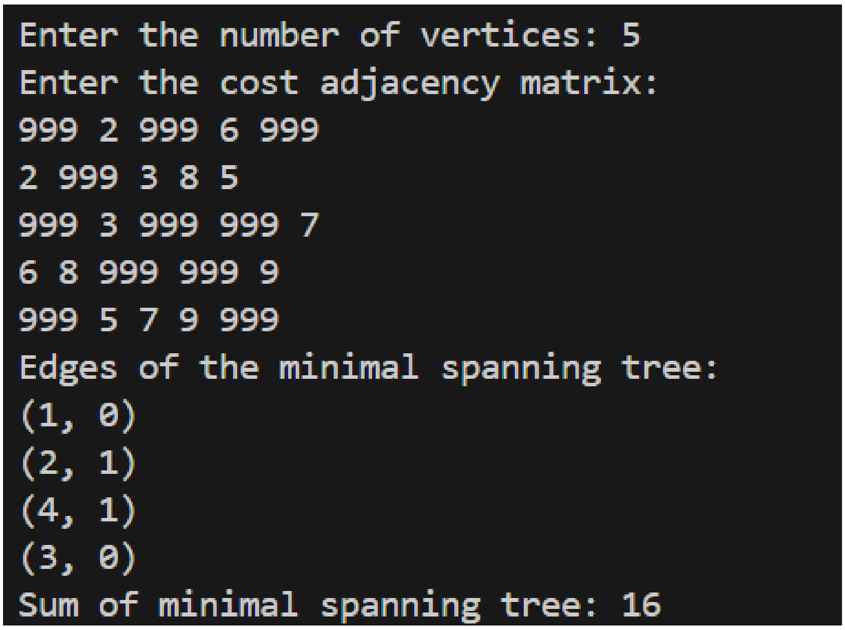
Lab program 8:

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

Code

```
#include <stdio.h>
#define INF 999  // Representation of infinity
int cost[10][10], t[10][2], sum;
int n;
void prims(int cost[10][10], int n);
int main() {
    int i, j;
    printf("Enter the number 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", &cost[i][j]);
        }
    }

    prims(cost, n);

    printf("Edges of the minimal spanning tree:\n");
    for (i = 0; i < n - 1; i++) {
        printf("(%d, %d)\n", t[i][0], t[i][1]);
    }

    printf("Sum of minimal spanning tree: %d\n", sum);
    return 0;
}

void prims(int cost[10][10], int n) {
    int i, j, u, v, k = 0;
    int min, source = 0;
    int p[10], d[10], s[10];

    // Initialize
    for (i = 0; i < n; i++) {
        d[i] = cost[source][i];
        s[i] = 0;
        p[i] = source;
    }
    s[source] = 1;
    sum = 0;

    // Build MST
    for (i = 0; i < n - 1; i++) {
        min = INF;
        u = -1;

        // Find the vertex with minimum edge cost to MST
        for (j = 0; j < n; j++) {
            if (s[j] == 0 && d[j] < min) {
                min = d[j];
                u = j;
            }
        }
        if (u != -1) {
            t[k][0] = u;
            t[k][1] = p[u];
            k++;
            sum += cost[u][p[u]];
            s[u] = 1;

            // Update distances
            for (v = 0; v < n; v++) {
                if (s[v] == 0 && cost[u][v] < d[v]) {
                    d[v] = cost[u][v];
                    p[v] = u;
                }
            }
        }
    }}
```

Screenshot of Output



```
Enter the number of vertices: 5
Enter the cost adjacency matrix:
999 2 999 6 999
2 999 3 8 5
999 3 999 999 7
6 8 999 999 9
999 5 7 9 999
Edges of the minimal spanning tree:
(1, 0)
(2, 1)
(4, 1)
(3, 0)
Sum of minimal spanning tree: 16
```

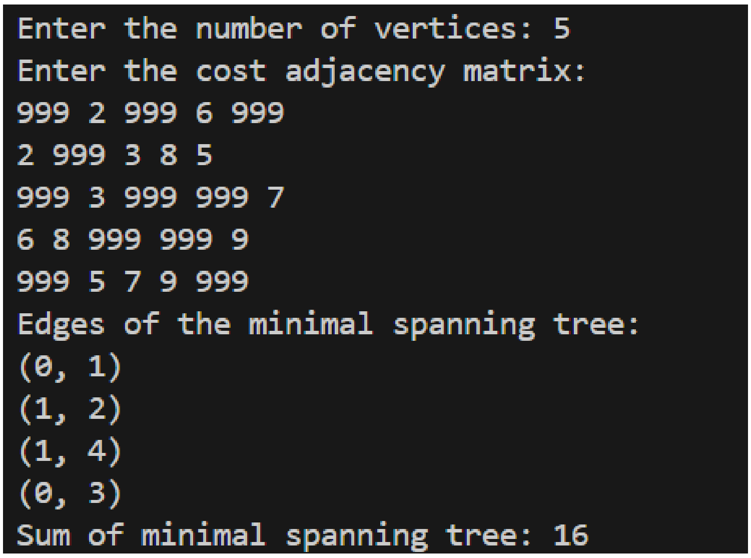
Find Minimum Cost Spanning Tree of a given undirected graph using Kruskal's algorithm.

Code

```
#include <stdio.h>
#define INF 999
int cost[10][10], t[10][2], sum;
int n;
void kruskal(int cost[10][10], int n);
int find(int parent[10], int i);
int main() {
  int i, j;
  printf("Enter the number 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", &cost[i][j]);
    }
  }

  kruskal(cost, n);

  printf("Edges of the minimal spanning tree:\n");
  for (i = 0; i < n - 1; i++) {
    printf("(%d, %d)\n", t[i][0], t[i][1]);
  }
  printf("Sum of minimal spanning tree: %d\n", sum);
  return 0;
}
```

```
void kruskal(int cost[10][10], int n) {
    int parent[10];
    int u, v, i, j, k = 0, count = 0;
    int min;

    sum = 0;

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

    while (count < n - 1) {
        min = INF;
        u = -1;
        v = -1;

        // Find the minimum edge (u, v) such that u and v are in different sets
        for (i = 0; i < n; i++) {
            for (j = 0; j < n; j++) {
                if (find(parent, i) != find(parent, j) && cost[i][j] < min) {
                    min = cost[i][j];
                    u = i;
                    v = j;
                }
            }
        }
        if (u != -1 && v != -1) {
            int root_u = find(parent, u);
            int root_v = find(parent, v);

            // Union the sets
            parent[root_u] = root_v;

            t[k][0] = u;
            t[k][1] = v;
            k++;
            sum += min;
            count++;
        }}}
int find(int parent[10], int i) {
    while (parent[i] != i) {
        i = parent[i];
    }
    return i;
}
```

Screenshot of Output



```
Enter the number of vertices: 5
Enter the cost adjacency matrix:
999 2 999 6 999
2 999 3 8 5
999 3 999 999 7
6 8 999 999 9
999 5 7 9 999
Edges of the minimal spanning tree:
(0, 1)
(1, 2)
(1, 4)
(0, 3)
Sum of minimal spanning tree: 16
```

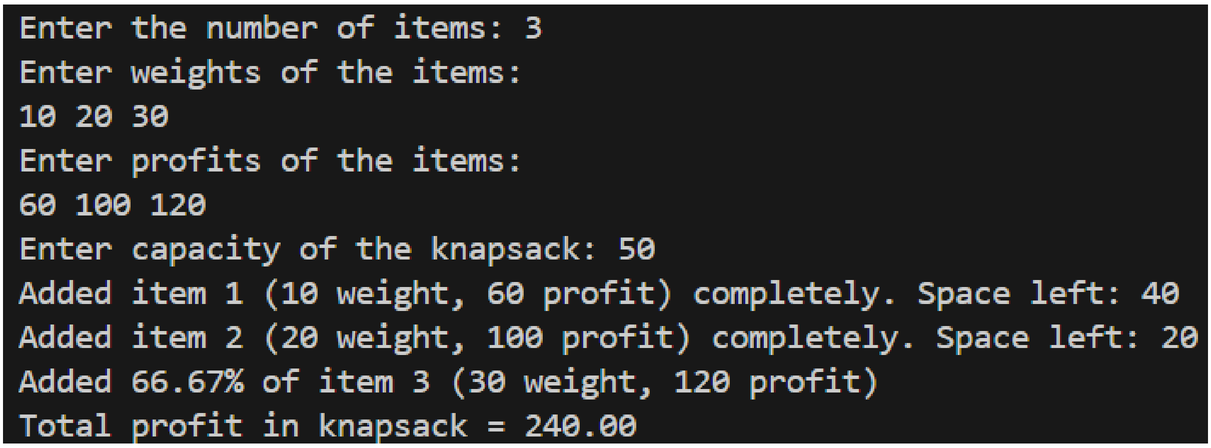
Lab program 9:

Implement Fractional Knapsack using Greedy technique.

Code

```
#include <stdio.h>

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

    int weights[n], profits[n];
    float ratio[n];
    int used[n];

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

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

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

    // Initialize used array to 0
    for (i = 0; i < n; i++) {
        used[i] = 0;
        ratio[i] = (float)profits[i] / weights[i];  // profit/weight ratio
    }

    int current_weight = capacity;
    float total_value = 0.0;

    while (current_weight > 0) {
        int maxi = -1;
        for (i = 0; i < n; i++) {
            if (!used[i]) {
                if (maxi == -1 || ratio[i] > ratio[maxi]) {
                    maxi = i;
                }
            }
        }
        if (maxi == -1) break;  // No items left
        used[maxi] = 1;

        if (weights[maxi] <= current_weight) {
            // Take whole item
            current_weight -= weights[maxi];
            total_value += profits[maxi];
            printf("Added item %d (%d weight, %d profit) completely. Space left: %d\n",
                   maxi + 1, weights[maxi], profits[maxi], current_weight);
        } else {
            // Take fraction of item
            float fraction = (float)current_weight / weights[maxi];
            total_value += profits[maxi] * fraction;
            printf("Added %.2f%% of item %d (%d weight, %d profit)\n",
                   fraction * 100, maxi + 1, weights[maxi], profits[maxi]);
            current_weight = 0;
        }
    }

    printf("Total profit in knapsack = %.2f\n", total_value);

    return 0;
}
```

Screenshot of Output



```
Enter the number of items: 3
Enter weights of the items:
10 20 30
Enter profits of the items:
60 100 120
Enter capacity of the knapsack: 50
Added item 1 (10 weight, 60 profit) completely. Space left: 40
Added item 2 (20 weight, 100 profit) completely. Space left: 20
Added 66.67% of item 3 (30 weight, 120 profit)
Total profit in knapsack = 240.00
```

LeetCode Program related to Greedy Technique algorithms.

Code

```
def maximumUnits(self, boxTypes: List[List[int]], truckSize: int) -> int:
        boxTypes.sort(key=lambda a:-a[1])
        max_units = 0
        for box in boxTypes:
                if truckSize < 0 : break
                max_units += min(truckSize, box[0]) * box[1]
                truckSize -= box[0]
        return max_units
```

Lab program 10:

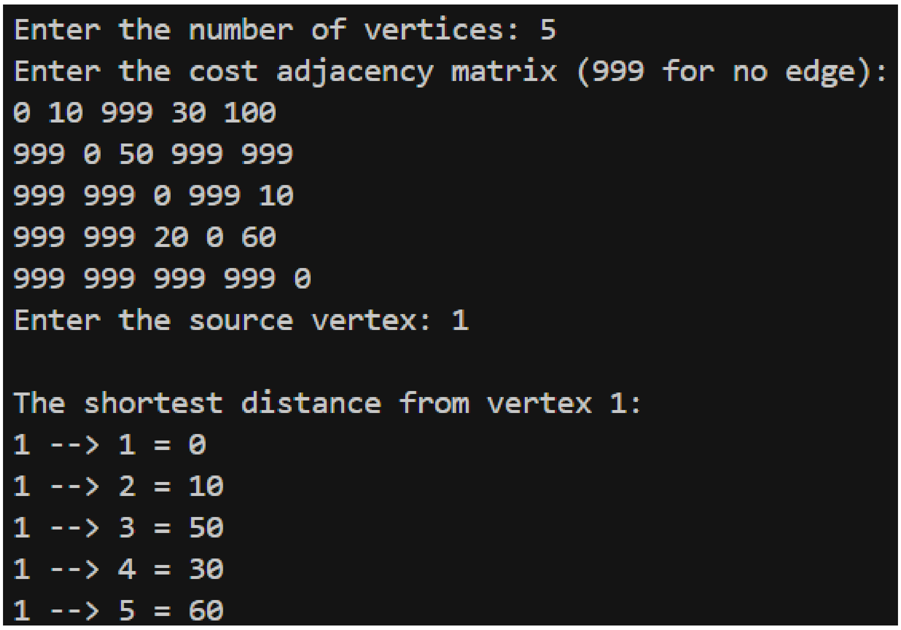
From a given vertex in a weighted connected graph, find shortest paths to other vertices using Dijkstra's algorithm.

Code

```
#include <stdio.h>


#define MAX 20
#define INF 999


int main() {
   int i, j, n, v, k, min, u;
   int c[MAX][MAX], s[MAX], d[MAX];


   printf("Enter the number of vertices: ");
   scanf("%d", &n);
   printf("Enter the cost adjacency matrix (999 for no edge):\n");
```

```
for (i = 1; i <= n; i++) {
        for (j = 1; j <= n; j++) {
            scanf("%d", &c[i][j]);
        }
    }

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

    // Initialization
    for (i = 1; i <= n; i++) {
        s[i] = 0;             // Not visited
        d[i] = c[v][i];       // Initial distances from source
    }

    d[v] = 0;  // Distance to itself is 0
    s[v] = 1;  // Mark source as visited

    for (k = 2; k <= n; k++) {
        min = INF;
        u = -1;

        // Find the vertex with minimum distance
        for (i = 1; i <= n; i++) {
            if (s[i] == 0 && d[i] < min) {
                min = d[i];
                u = i;
            }
        }
        if (u == -1) break; // No reachable vertex

        s[u] = 1;

        // Update distances
        for (i = 1; i <= n; i++) {
            if (s[i] == 0 && d[i] > d[u] + c[u][i]) {
                d[i] = d[u] + c[u][i];
            }
        }
    }

    // Print result
    printf("\nThe shortest distance from vertex %d:\n", v);
    for (i = 1; i <= n; i++) {
        if (d[i] != INF)
            printf("%d --> %d = %d\n", v, i, d[i]);
        else
            printf("%d --> %d = Unreachable\n", v, i);
    }

    return 0;
}
```

Screenshot of Output



```
Enter the number of vertices: 5
Enter the cost adjacency matrix (999 for no edge):
0 10 999 30 100
999 0 50 999 999
999 999 0 999 10
999 999 20 0 60
999 999 999 999 0
Enter the source vertex: 1

The shortest distance from vertex 1:
1 --> 1 = 0
1 --> 2 = 10
1 --> 3 = 50
1 --> 4 = 30
1 --> 5 = 60
```

Lab program 11:

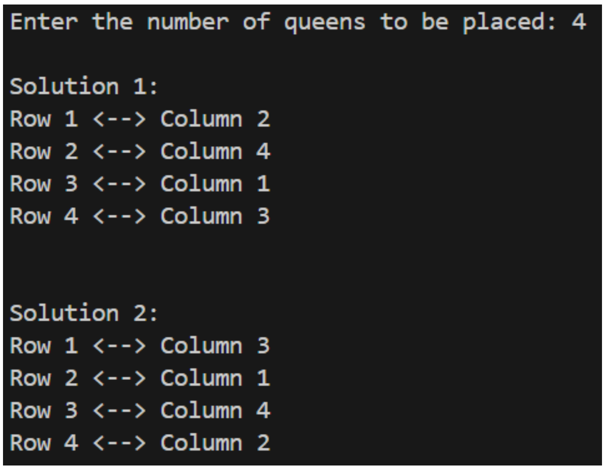
Implement "N-Queens Problem" using Backtracking.

Code

```
#include <stdio.h>

#include <math.h>


int x[20], count = 1;


int place(int k, int j) {

    for (int i = 1; i < k; i++) {

        if (x[i] == j || abs(x[i] - j) == abs(i - k))

            return 0; // Not safe to place queen

    }

    return 1; // Safe

}
```

```
void queens(int k, int n) {
    for (int j = 1; j <= n; j++) {
        if (place(k, j)) {
            x[k] = j;
            if (k == n) {
                printf("\nSolution %d:\n", count++);
                for (int i = 1; i <= n; i++)
                    printf("Row %d <--> Column %d\n", i, x[i]);
                printf("\n");
            } else {
                queens(k + 1, n);
      }}}}
int main() {
    int n;
    printf("Enter the number of queens to be placed: ");
    scanf("%d", &n);
    queens(1, n);
    return 0;
}
```

Screenshot of Output



```
Enter the number of queens to be placed: 4

Solution 1:
Row 1 <--> Column 2
Row 2 <--> Column 4
Row 3 <--> Column 1
Row 4 <--> Column 3


Solution 2:
Row 1 <--> Column 3
Row 2 <--> Column 1
Row 3 <--> Column 4
Row 4 <--> Column 2
```