VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT
On

ANALYSIS AND DESIGN OF ALGORITHMS (23CS4PCADA)

Submitted by

K VEENA(1BM23CS135)

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 **K VEENA(1BM23CS135)** 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.

Prof. SARALA
Assistant Professor
Department of CSE
BMSCE, Bengaluru

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

Index Sheet

Sl. No	Experiment Title	Page No.
1	Write program to obtain the Topological ordering of vertices in a given digraph. LeetCode Program related to Topological sorting	5
2	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. LeetCode Program related to sorting.	9
3	Sort a given set of N integer elements using Quick Sort technique and compute its time taken. LeetCode Program related to sorting.	13
4	Find Minimum Cost Spanning Tree of a given undirected graph using Prim's algorithm. Find Minimum Cost Spanning Tree of a given undirected graph using Kruskal's algorithm.	16
5	From a given vertex in a weighted connected graph, find shortest paths to other vertices using Dijkstra's algorithm.	22
6	Implement Johnson Trotter algorithm to generate permutations.	25
7	Implement Fractional Knapsack using Greedy technique. LeetCode Program related to Greedy Technique algorithms.	29
8	Implement 0/1 Knapsack problem using dynamic programming. LeetCode Program related to Knapsack problem or Dynamic Programming.	32
9	Sort a given set of N integer elements using Heap Sort technique and compute its time taken.	36
10	Implement All Pair Shortest paths problem using Floyd's algorithm. LeetCode Program related to shortest distance calculation.	38
11	Implement "N-Queens Problem" using Backtracking.	41

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 1.1:

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

Program full details

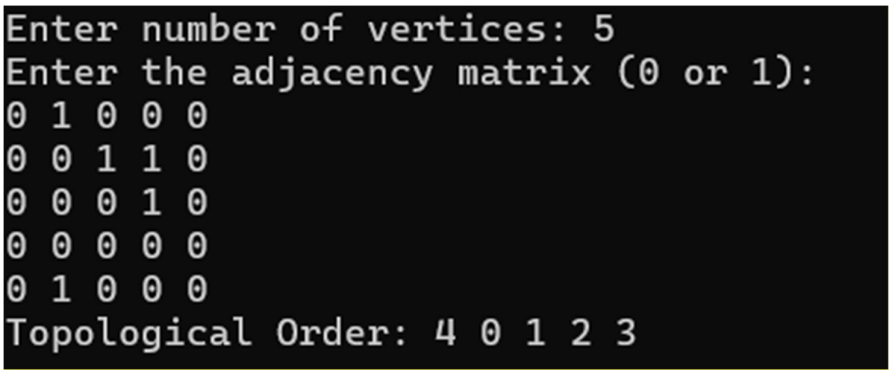
Code

```
#include <stdio.h>
#include <stdbool.h>
#define MAX 100

int graph[MAX][MAX];
bool visited[MAX];
int stack[MAX];
int top = -1;
int n;

void push(int v) {
   stack[++top] = v;
}


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


   push(node);
```

```
}


void topologicalSort() {

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

    for (int i = 0; i < n; i++) {
        if (!visited[i]) {
            dfs(i);
        }
    }
    printf("Topological Order: ");
    while (top != -1) {
        printf("%d ", stack[top--]);
    }
    printf("\n");
}

int main() {

    printf("Enter number of vertices: ");
    scanf("%d", &n);
    printf("Enter the adjacency matrix (0 or 1):\n");
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            scanf("%d", &graph[i][j]);
```

```
        }
    }
    topologicalSort();
    return 0;
}
```

Screenshot of Output



Lab program 1.2:

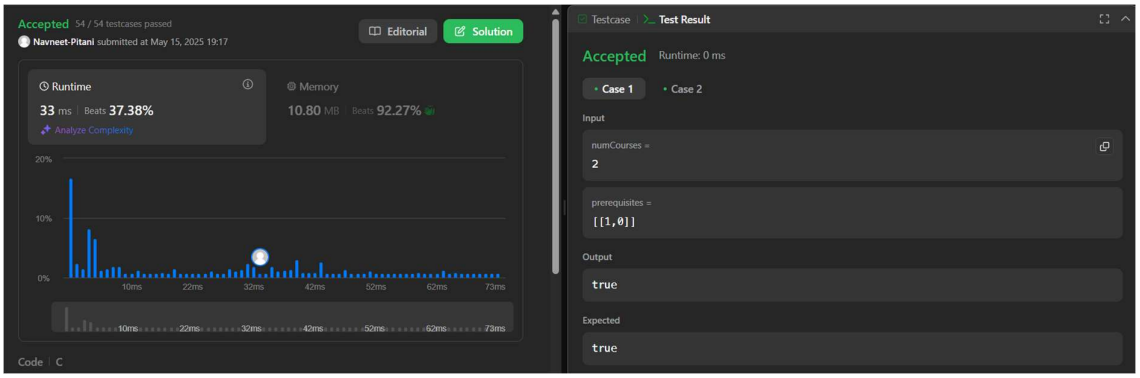
LeetCode Program related to Topological sorting

Code

```
bool dfs(int course, int** prerequisites, int prerequisitesSize, int*
prerequisitesColSize, int* visited, int numCourses) {
    if (visited[course] == 1) {
        return false;
    }
    if (visited[course] == 2) {
        return true;
    }
    visited[course] = 1;
    for (int i = 0; i < prerequisitesSize; i++) {
        if (prerequisites[i][0] == course) {
```

```
            int nextCourse = prerequisites[i][1];

            if (!dfs(nextCourse, prerequisites, prerequisitesSize,
prerequisitesColSize, visited, numCourses)) {

                return false;

            }

        }

    }

    visited[course] = 2;

    return true;

}

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

    int visited[numCourses];

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

        visited[i] = 0;

    }

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

        if (visited[i] == 0) {

            if (!dfs(i, prerequisites,prerequisitesSize,
prerequisitesColSize, visited, numCourses)) {

                return false;

            }

        }

    }

    return true;

}
```

Screenshot of Output



Lab program 2:

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

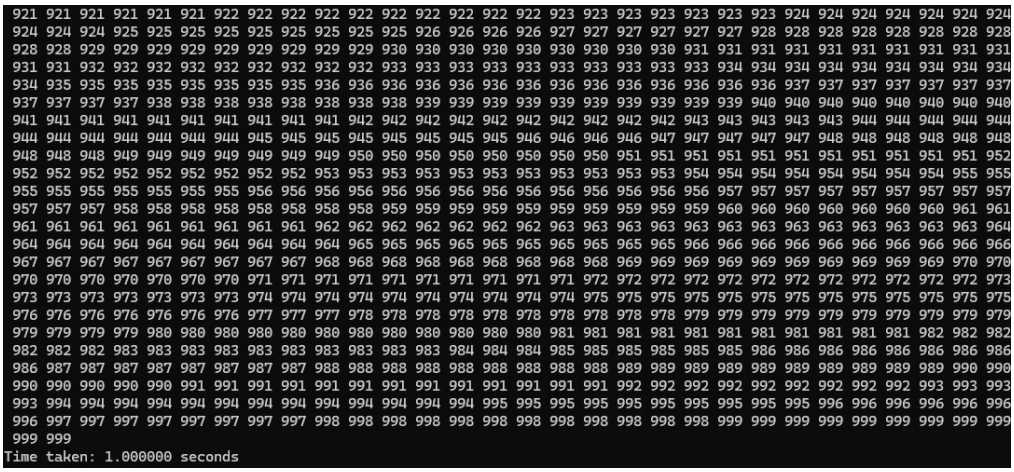
```
#include <stdio.h>

#include <stdlib.h>

#include <time.h>


void merge(int arr[], int left, int right, int mid) {

    int i, j, k;

    int n1 = mid - left + 1;

    int n2 = right - mid;

    int L[n1], R[n2];


    for(i = 0; i < n1; i++) {

        L[i] = arr[left + i];

    }

    for(j = 0; j < n2; j++) {

        R[j] = arr[mid + 1 + j];

    }


    i = 0;
```

```
   j = 0;
   k = left;

   while(i < n1 && j < n2) {
      if(L[i] <= R[j]) {
         arr[k] = L[i];
         i++;
      } else {
         arr[k] = R[j];
         j++;
      }
      k++;
   }

   while(i < n1) {
      arr[k] = L[i];
      i++;
      k++;
   }

   while(j < n2) {
      arr[k] = R[j];
      j++;
      k++;
   }
}

void mergeSort(int arr[], int left, int right) {
   if(left < right) {
```

```
        int mid = left + (right - left) / 2;

        mergeSort(arr, left, mid);
        mergeSort(arr, mid + 1, right);
        merge(arr, left, right, mid);
    }
}

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

int main() {
    int n;
    clock_t start, end;



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

    int arr[n];

    srand(time(NULL));


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

```
        arr[i] = rand() % 1000;
    }


    printf("Original Array: ");
    print(arr, n);

    start = clock();


    mergeSort(arr, 0, n - 1);


    end = clock();


    printf("Sorted Array: ");
    print(arr, n);

    printf("Time taken: %f seconds\n",1000* (double)(end - start) / CLOCKS_PER_SEC);




    return 0;
}
```

Screenshot of Output



Lab program 3:

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

Code

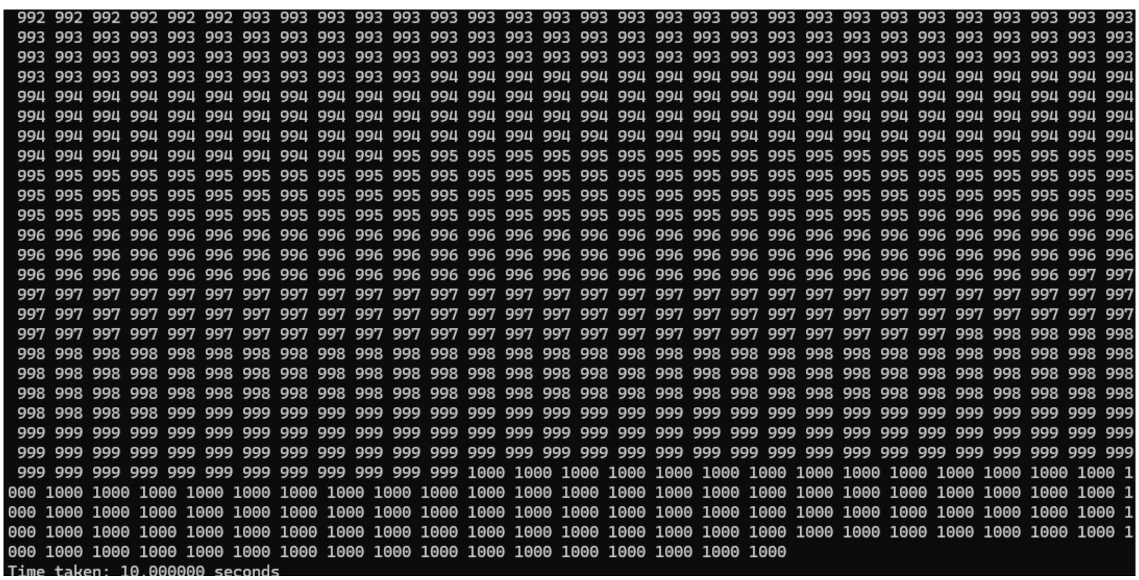
```
#include <stdio.h>

#include <stdlib.h>

#include <time.h>



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



    for (int j = low; j <= high - 1; j++) {
        if (arr[j] < pivot) {
            i++;


            int temp = arr[i];
            arr[i] = arr[j];
            arr[j] = temp;
        }
```

```
    }

    int temp = arr[i + 1];
    arr[i + 1] = arr[high];
    arr[high] = temp;

    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);
    }
}

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

```
int main() {
    int n;
    clock_t start, end;

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

    int arr[n];

    srand(time(NULL));

    for (int i = 0; i < n; i++) {
        arr[i] = rand() % 1001;
    }

    printf("Original Array: ");
    print(arr, n);

    start = clock();

    quickSort(arr, 0, n - 1);
```

```
    end = clock();



    printf("Sorted Array: ");
    print(arr, n);



    printf("Time taken: %f seconds\n",1000* (double)(end - start) / CLOCKS_PER_SEC);





    return 0;
}
```

Screenshot of Output



```
 992 992 992 992 992 992 993 993 993 993 993 993 993 993 993 993 993 993 993 993 993 993 993 993 993 993 993 993 993 993
 993 993 993 993 993 993 993 993 993 993 993 993 993 993 993 993 993 993 993 993 993 993 993 993 993 993 993 993 993 993
 993 993 993 993 993 993 993 993 993 993 993 993 993 993 993 993 993 993 993 993 993 993 993 993 993 993 993 993 993 993
 993 993 993 993 993 993 993 993 993 993 993 994 994 994 994 994 994 994 994 994 994 994 994 994 994 994 994 994 994 994
 994 994 994 994 994 994 994 994 994 994 994 994 994 994 994 994 994 994 994 994 994 994 994 994 994 994 994 994 994 994
 994 994 994 994 994 994 994 994 994 994 994 994 994 994 994 994 994 994 994 994 994 994 994 994 994 994 994 994 994 994
 994 994 994 994 994 994 994 994 994 994 994 994 994 994 994 994 994 994 994 994 994 994 994 994 994 994 994 994 994 994
 994 994 994 994 994 994 994 994 994 994 995 995 995 995 995 995 995 995 995 995 995 995 995 995 995 995 995 995 995 995
 995 995 995 995 995 995 995 995 995 995 995 995 995 995 995 995 995 995 995 995 995 995 995 995 995 995 995 995 995 995
 995 995 995 995 995 995 995 995 995 995 995 995 995 995 995 995 995 995 995 995 995 995 995 995 995 995 995 995 995 995
 995 995 995 995 995 995 995 995 995 995 995 995 995 995 995 995 995 995 995 995 995 995 995 995 996 996 996 996 996 996
 996 996 996 996 996 996 996 996 996 996 996 996 996 996 996 996 996 996 996 996 996 996 996 996 996 996 996 996 996 996
 996 996 996 996 996 996 996 996 996 996 996 996 996 996 996 996 996 996 996 996 996 996 996 996 996 996 996 996 996 996
 996 996 996 996 996 996 996 996 996 996 996 996 996 996 996 996 996 996 996 996 996 996 996 996 996 996 996 996 997 997
 997 997 997 997 997 997 997 997 997 997 997 997 997 997 997 997 997 997 997 997 997 997 997 997 997 997 997 997 997 997
 997 997 997 997 997 997 997 997 997 997 997 997 997 997 997 997 997 997 997 997 997 997 997 997 997 997 997 997 997 997
 997 997 997 997 997 997 997 997 997 997 997 997 997 997 997 997 997 997 997 997 997 997 997 997 997 998 998 998 998 998
 998 998 998 998 998 998 998 998 998 998 998 998 998 998 998 998 998 998 998 998 998 998 998 998 998 998 998 998 998 998
 998 998 998 998 998 998 998 998 998 998 998 998 998 998 998 998 998 998 998 998 998 998 998 998 998 998 998 998 998 998
 998 998 998 998 998 998 998 998 998 998 998 998 998 998 998 998 998 998 998 998 998 998 998 998 998 998 998 998 998 998
 998 998 998 998 999 999 999 999 999 999 999 999 999 999 999 999 999 999 999 999 999 999 999 999 999 999 999 999 999 999
 999 999 999 999 999 999 999 999 999 999 999 999 999 999 999 999 999 999 999 999 999 999 999 999 999 999 999 999 999 999
 999 999 999 999 999 999 999 999 999 999 999 999 999 999 999 999 999 999 999 999 999 999 999 999 999 999 999 999 999 999
 999 999 999 999 999 999 999 999 999 999 999 999 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1
000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1
000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1
000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1
000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000
Time taken: 10.000000 seconds
```

Lab program 4:

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

Code

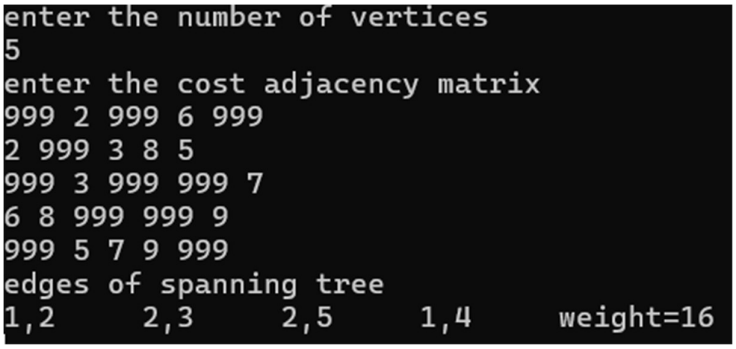
```
#include<stdio.h>
```

```
#include<conio.h>

int cost[10][10],vt[10],et[10][10],vis[10],j,n;
int sum=0;
int x=1;
int e=0;
void prims();

void main()
{
  int i;

  printf("enter the number of vertices\n");
  scanf("%d",&n);
  printf("enter the cost adjacency matrix\n");
  for(i=1;i<=n;i++)
  {
     for(j=1;j<=n;j++)
     {
          scanf("%d",&cost[i][j]);
     }
     vis[i]=0;
  }
  prims();
  printf("edges of spanning tree\n");
  for(i=1;i<=e;i++)
  {
       printf("%d,%d\t",et[i][0],et[i][1]);
  }
```

```
    printf("weight=%d\n",sum);
    getch();
}

void prims()
{
   int s,min,m,k,u,v;
   vt[x]=1;
   vis[x]=1;
   for(s=1;s<n;s++)
   {
      j=x;
      min=999;
      while(j>0)
      {
            k=vt[j];
            for(m=2;m<=n;m++)
            {
              if(vis[m]==0)
              {
                    if(cost[k][m]<min)
                    {
                      min=cost[k][m];
                      u=k;
                      v=m;
                    }
               }
           }
           j--;
```

```
    }
    vt[++x]=v;
    et[s][0]=u;
    et[s][1]=v;
    e++;
    vis[v]=1;
    sum=sum+min;
  }
}
```

Screenshot of Output



Lab program 5:

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

Code

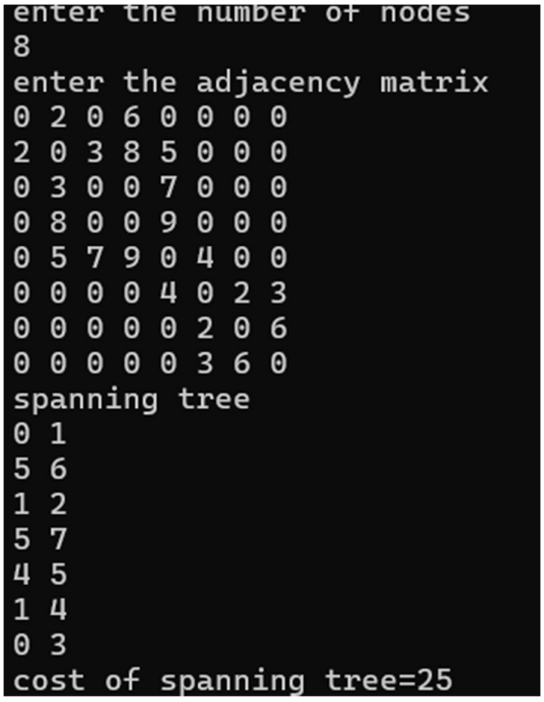
```
#include<stdio.h>
#include<conio.h>

int find(int v,int parent[10])
{
   while(parent[v]!=v)
   {
      v=parent[v];
   }
   return v;
```

```
}

void union1(int i,int j,int parent[10])
{
   if(i<j)
      parent[j]=i;
   else
      parent[i]=j;
}

void kruskal(int n,int a[10][10])
{
   int count,k,min,sum,i,j,t[10][10],u,v,parent[10];
   count=0;
   k=0;
   sum=0;
   for(i=0;i<n;i++)
      parent[i]=i;
   while(count!=n-1)
   {
      min=999;
      for(i=0;i<n;i++)
      {
       for(j=0;j<n;j++)
       {

          if(a[i][j]<min && a[i][j]!=0)
          {
            min=a[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+a[u][v];
       }
       a[u][v]=a[v][u]=999;
    }
    if(count==n-1)
    {
       printf("spanning tree\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("spanning tree does not exist\n");
  }


void main()
{
   int n,i,j,a[10][10];
```

```
    clrscr();
    printf("enter the number of nodes\n");
    scanf("%d",&n);
    printf("enter the adjacency matrix\n");
    for(i=0;i<n;i++)
      for(j=0;j<n;j++)
       scanf("%d",&a[i][j]);
    kruskal(n,a);
    getch();
}
```

Screenshot of Output



Lab program 6:

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

Code

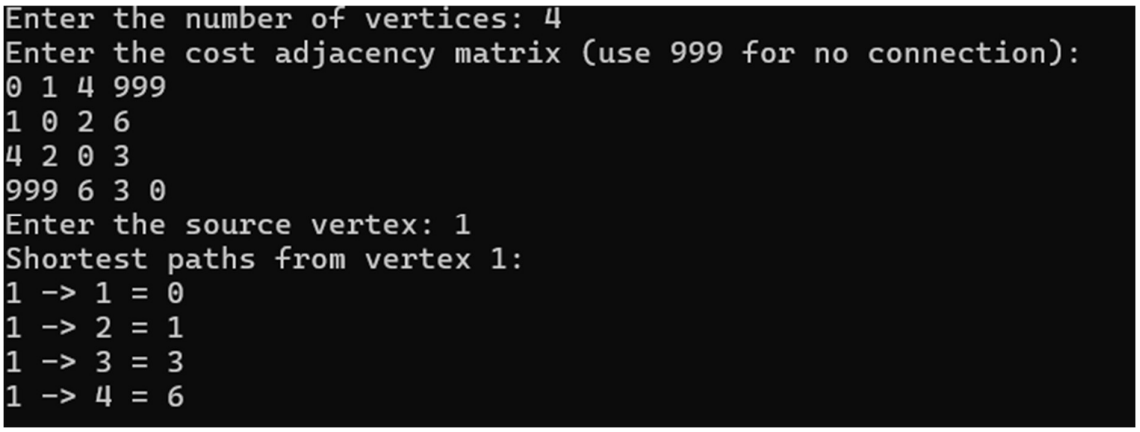
#include <stdio.h>

```
#define INF 999

void dijkstra(int n, int cost[10][10], int src) {
    int i, j, u, dis[10], vis[10], min;

    // Initialize distances and visited flags
    for (i = 1; i <= n; i++) {
        dis[i] = cost[src][i];
        vis[i] = 0;
    }

    vis[src] = 1;

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

        // Find the unvisited vertex with the smallest distance
        for (j = 1; j <= n; j++) {
            if (vis[j] == 0 && dis[j] < min) {
                min = dis[j];
                u = j;
            }
        }

        if (u == -1) break; // All reachable vertices visited

        vis[u] = 1;
```

```
        // Update distances to neighboring vertices
        for (j = 1; j <= n; j++) {
            if (vis[j] == 0 && dis[u] + cost[u][j] < dis[j]) {
                dis[j] = dis[u] + cost[u][j];
            }
        }
    }

    printf("Shortest paths from vertex %d:\n", src);
    for (i = 1; i <= n; i++) {
        if (dis[i] == INF)
            printf("%d -> %d = INF\n", src, i);
        else
            printf("%d -> %d = %d\n", src, i, dis[i]);
    }
}

int main() {
    int src, j, cost[10][10], n, i;

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

    printf("Enter the cost adjacency matrix (use 999 for no connection):\n");
    for (i = 1; i <= n; i++) {
        for (j = 1; j <= n; j++) {
            scanf("%d", &cost[i][j]);
        }
```

```
    }

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

    dijkstra(n, cost, src);

    return 0;
}
```

Screenshot of Output



Lab program 7:

Implement Johnson Trotter algorithm to generate permutations.

Code

```
#include <stdio.h>

#define LEFT_TO_RIGHT 1
#define RIGHT_TO_LEFT 0
```

```
int searchArr(int a[], int n, int mobile) {
    for (int i = 0; i < n; i++)
        if (a[i] == mobile)
            return i + 1;
    return -1;
}

int getMobile(int a[], int dir[], int n) {
    int mobile_prev = 0, mobile = 0;

    for (int i = 0; i < n; i++) {
        if (dir[a[i] - 1] == RIGHT_TO_LEFT && i != 0) {
            if (a[i] > a[i - 1] && a[i] > mobile_prev) {
                mobile = a[i];
                mobile_prev = mobile;
            }
        }
        if (dir[a[i] - 1] == LEFT_TO_RIGHT && i != n - 1) {
            if (a[i] > a[i + 1] && a[i] > mobile_prev) {
                mobile = a[i];
                mobile_prev = mobile;
            }
        }
    }

    return mobile;
}

void printOnePerm(int a[], int dir[], int n) {
```

```
    int mobile = getMobile(a, dir, n);
    int pos = searchArr(a, n, mobile);

    if (mobile == 0) return;

    if (dir[a[pos - 1] - 1] == RIGHT_TO_LEFT) {
        int temp = a[pos - 1];
        a[pos - 1] = a[pos - 2];
        a[pos - 2] = temp;
    } else if (dir[a[pos - 1] - 1] == LEFT_TO_RIGHT) {
        int temp = a[pos];
        a[pos] = a[pos - 1];
        a[pos - 1] = temp;
    }

    for (int i = 0; i < n; i++) {
        if (a[i] > mobile) {
            dir[a[i] - 1] = !dir[a[i] - 1]; // toggle direction
        }
    }

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

int fact(int n) {
    int res = 1;
    for (int i = 1; i <= n; i++)
```

```
        res = res * i;
    return res;
}

void printPermutation(int n) {
    int a[n], dir[n];

    for (int i = 0; i < n; i++) {
        a[i] = i + 1;
        printf("%d", a[i]);
    }
    printf("\n");

    for (int i = 0; i < n; i++)
        dir[i] = RIGHT_TO_LEFT;

    for (int i = 1; i < fact(n); i++)
        printOnePerm(a, dir, n);
}

int main() {
    int n = 4;
    printPermutation(n);
    return 0;
}
```

Screenshot of Output

```
1234
1243 1423 4123 4132 1432 1342 1324 3124 3142 3412 4312 4321 3421 3241 3214 2314 2341 2431 4231 4213 2413 2143 2134
```

Lab program 8.1:

Implement Fractional Knapsack using Greedy technique.

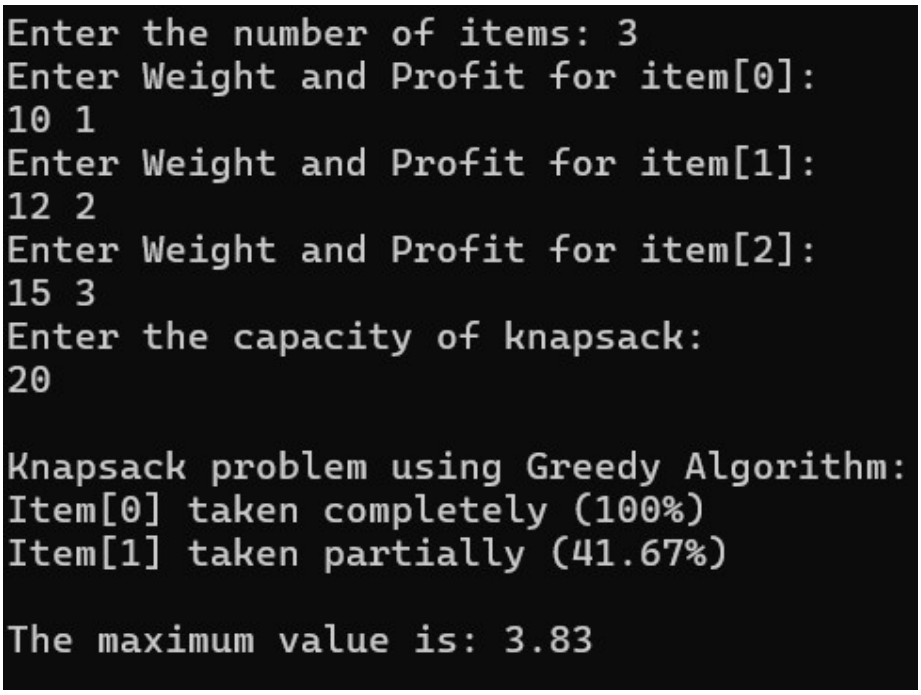
Code

```
#include <stdio.h>

int main() {
    float weight[50], profit[50], ratio[50];
    float Totalvalue = 0.0, temp, capacity, amount;
    int n, i, j;

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

    for (i = 0; i < n; i++) {
        printf("Enter Weight and Profit for item[%d]:\n", i);
        scanf("%f %f", &weight[i], &profit[i]);
    }

    printf("Enter the capacity of knapsack:\n");
    scanf("%f", &capacity);

    // Calculate profit/weight ratio
    for (i = 0; i < n; i++)
        ratio[i] = profit[i] / weight[i];

    // Sort items by descending ratio
    for (i = 0; i < n; i++) {
        for (j = i + 1; j < n; j++) {
            if (ratio[i] < ratio[j]) {
```

```
            // Swap ratio
            temp = ratio[i];
            ratio[i] = ratio[j];
            ratio[j] = temp;

            // Swap weight
            temp = weight[i];
            weight[i] = weight[j];
            weight[j] = temp;

            // Swap profit
            temp = profit[i];
            profit[i] = profit[j];
            profit[j] = temp;
        }
    }
}

printf("\nKnapsack problem using Greedy Algorithm:\n");
for (i = 0; i < n; i++) {
    if (weight[i] <= capacity) {
        // Take full item
        printf("Item[%d] taken completely (100%%)\n", i);
        Totalvalue += profit[i];
        capacity -= weight[i];
    } else {
        // Take fraction of item
        float fraction = capacity / weight[i];
        Totalvalue += profit[i] * fraction;
```

```
            printf("Item[%d] taken partially (%.2f%%)\n", i, fraction * 100);
            break; // Knapsack is now full
        }
    }

    printf("\nThe maximum value is: %.2f\n", Totalvalue);
    return 0;
}
```

Screenshot of Output



Lab program 8.2:

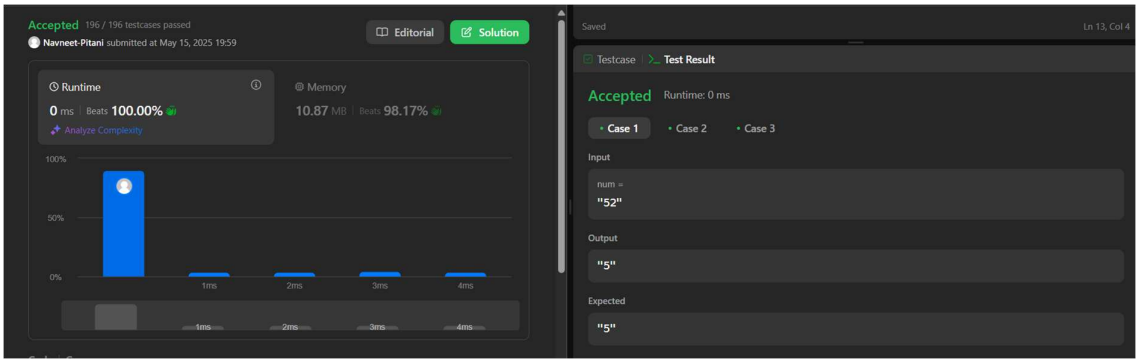
LeetCode Program related to Greedy Technique algorithms

Code

```
char* largestOddNumber(char* num) {
  int len = strlen(num);
```

```
    for (int i = len - 1; i >= 0; i--) {
        if ((num[i] - '0') % 2 == 1) {
            num[i + 1] = '\0'; // Truncate string at that position
            return num; // Return the longest odd-suffix (greedy)
        }
    }

    return ""; // No odd digit found
}
```

Screenshot of Output



Lab program 9.1:

Implement 0/1 Knapsack problem using dynamic programming.

Code

```
#include <stdio.h>

// Function to return the maximum of two numbers
int max(int a, int b) {
    return (a > b) ? a : b;
}

// Function to solve the 0/1 Knapsack problem
```

```
int knapsack(int weight[], int profit[], int n, int capacity) {
    int i, w;
    int K[n + 1][capacity + 1];

    // Build the DP table K[][] bottom up
    for (i = 0; i <= n; i++) {
        for (w = 0; w <= capacity; w++) {
            if (i == 0 || w == 0)
                K[i][w] = 0;
            else if (weight[i - 1] <= w)
                K[i][w] = max(profit[i - 1] + K[i - 1][w - weight[i - 1]], K[i - 1][w]);
            else
                K[i][w] = K[i - 1][w];
        }
    }

    // Optional: Print the items included
    printf("\nItems included:\n");
    w = capacity;
    for (i = n; i > 0 && w > 0; i--) {
        if (K[i][w] != K[i - 1][w]) {
            printf("Item %d (Weight: %d, Profit: %d)\n", i, weight[i - 1], profit[i - 1]);
            w -= weight[i - 1];
        }
    }

    return K[n][capacity];
}
```

```
int main() {
    int n, capacity;
    int weight[50], profit[50];
    int i;

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

    printf("Enter weight and profit for each item:\n");
    for (i = 0; i < n; i++) {
        printf("Item[%d] - Weight Profit: ", i + 1);
        scanf("%d %d", &weight[i], &profit[i]);
    }

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

    int maxProfit = knapsack(weight, profit, n, capacity);

    printf("\nMaximum profit: %d\n", maxProfit);
    return 0;
}
```

Screenshot of Output

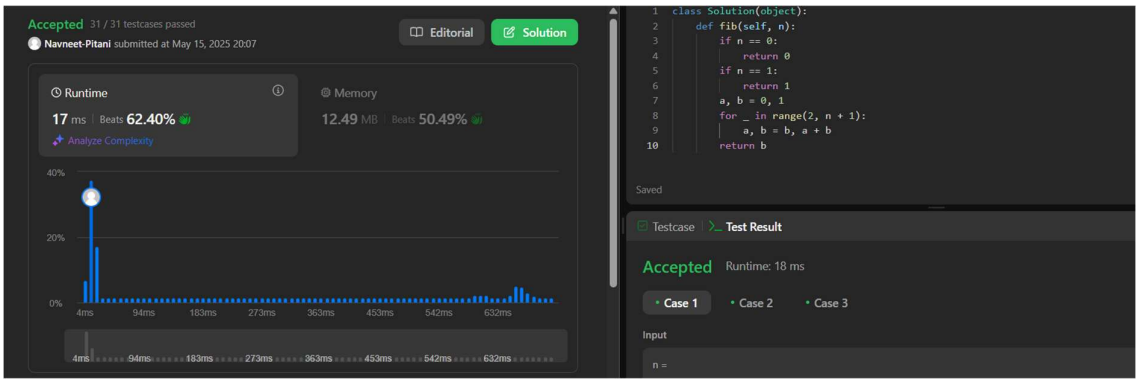
```
Enter number of items: 4
Enter weight and profit for each item:
Item[1] - Weight Profit: 2 12
Item[2] - Weight Profit: 3 15
Item[3] - Weight Profit: 1 25
Item[4] - Weight Profit: 2 10
Enter the capacity of knapsack: 4

Items included:
Item 3 (Weight: 1, Profit: 25)
Item 2 (Weight: 3, Profit: 15)

Maximum profit: 40
```

Lab program 9.2:

Code

```
class Solution(object):
    def fib(self, n):
        if n == 0:
            return 0
        if n == 1:
            return 1
        a, b = 0, 1
        for _ in range(2, n + 1):
            a, b = b, a + b
        return b
```

Screenshot of Output



Lab program 10:

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

Code

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


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) {
      int temp = arr[i];
      arr[i] = arr[largest];
      arr[largest] = temp;


      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--) {

      int temp = arr[0];
      arr[0] = arr[i];
      arr[i] = temp;


      heapify(arr, i, 0);
   }
}
```

```
int main() {
    int arr[1000], n;
    clock_t start, end;
    double time_taken;

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

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

    start = clock();

    heapSort(arr, n);

    end = clock();

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

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

    printf("\n\nTime taken by Heap Sort: %f seconds\n", time_taken);

    return 0;
}
```

Screenshot of Output

```
Enter number of elements: 7
Enter 7 integer elements:
50
25
30
75
100
45
80

Sorted array is:
25 30 45 50 75 80 100

Time taken by Heap Sort: 0.000000 seconds
```

Lab program 11.1:

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

Code

```
#include <stdio.h>


#define INF 99999  // Use a large number to represent infinity
#define MAX 100


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


    // Initialize the solution matrix same as input graph
    for (i = 0; i < n; i++)
        for (j = 0; j < n; j++)
            dist[i][j] = graph[i][j];
```

```
    // Floyd-Warshall algorithm
    for (k = 0; k < n; k++) {
        for (i = 0; i < n; i++) {
            for (j = 0; j < n; j++) {
                if (dist[i][k] + dist[k][j] < dist[i][j])
                    dist[i][j] = dist[i][k] + dist[k][j];
            }
        }
    }

    // Print the final shortest distance matrix
    printf("\nAll-Pairs Shortest Paths (Floyd-Warshall):\n");
    for (i = 0; i < n; i++) {
        for (j = 0; j < n; j++) {
            if (dist[i][j] == INF)
                printf("INF ");
            else
                printf("%3d ", dist[i][j]);
        }
        printf("\n");
    }
}

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

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

```
    printf("Enter the adjacency matrix (use 99999 for no direct path):\n");
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            scanf("%d", &graph[i][j]);
        }
    }
    floydWarshall(graph, n);

    return 0;
}
```

Screenshot of Output

```
Enter number of vertices: 4
Enter the adjacency matrix (use 99999 for no direct path):
0 4 3 9
99 0 1 99
99 990 99999
5 2 6 0
2 99 99999 99999

All-Pairs Shortest Paths (Floyd-Warshall):
  0   4   3   8
  8   0   1   6
  7  11   5   5
  2   6   0   2
```

Lab program 11.2:

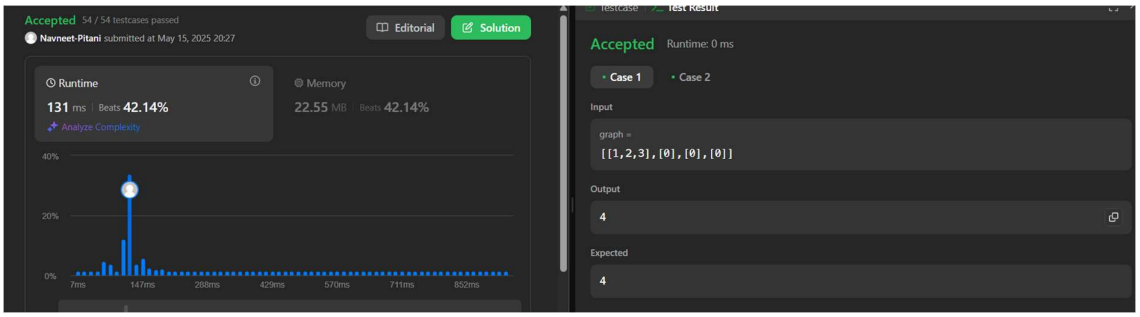
LeetCode Program related to shortest distance calculation

Code

```
class Solution:
    def shortestPathLength(self, graph: List[List[int]]) -> int:
        n=len(graph)
        queue=deque([(i,1<<i) for i in range(n)])
        seen=set(queue)
        ans=0
        while queue:
            for _ in range(len(queue)):
```

```
        u,m=queue.popleft()
        if m==(1<<n)-1:
            return ans
        for v in graph[u]:
            if (v,m|1<<v) not in seen:
                queue.append((v,m|1<<v))
                seen.add((v,m|1<<v))
    ans+=1
```

Screenshot of Output



Lab program 12:

Implement “N-Queens Problem” using Backtracking.

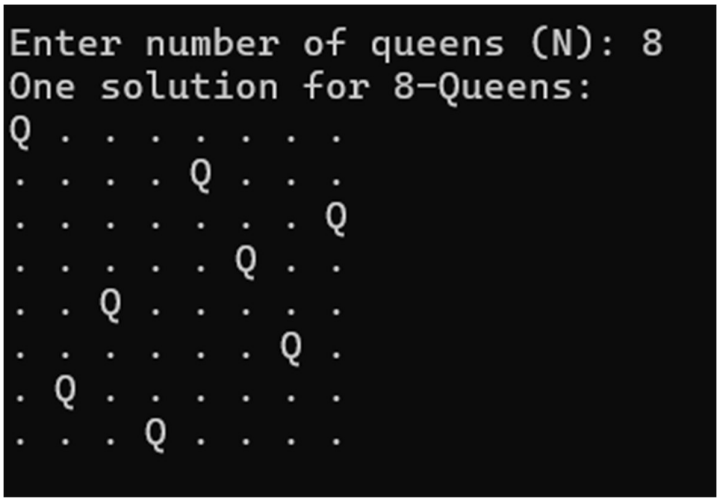
Code

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

#define MAX 20

int board[MAX];
int found = 0;
```

```
// Function to print one solution
void printSolution(int n) {
    printf("One solution for %d-Queens:\n", n);
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= n; j++) {
            if (board[i] == j)
                printf("Q ");
            else
                printf(". ");
        }
        printf("\n");
    }
    found = 1;
}

// Check if placing queen at (k, i) is safe
int isSafe(int k, int i) {
    for (int j = 1; j < k; j++) {
        if (board[j] == i || fabs(board[j] - i) == fabs(j - k))
            return 0;
    }
    return 1;
}

// Recursive backtracking to find one solution
void nQueens(int k, int n) {
    for (int i = 1; i <= n && !found; i++) {
        if (isSafe(k, i)) {
```

```
            board[k] = i;
            if (k == n)
                printSolution(n);
            else
                nQueens(k + 1, n);
        }
    }
}

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

    if (n < 1 || n > MAX) {
        printf("Please enter N between 1 and %d.\n", MAX);
        return 1;
    }

    nQueens(1, n);

    if (!found)
        printf("No solution exists for N = %d\n", n);

    return 0;
}
```

Screenshot of Output


Enter number of queens (N): 8
One solution for 8-Queens:
Q . . . .
. . . . Q . . .
. . . . Q
. . . . . Q . .
. . Q . . . . .
. . . . . . Q .
. Q . . . . . .
. . . Q . . . .