VISVESVARAYA TECHNOLOGICAL UNIVERSITY

“JnanaSangama”, Belgaum -590014, Karnataka.

LAB REPORT on

ANALYSIS AND DESIGN OF ALGORITHMS

Submitted by

BHAVYA GOYAL (1BM23CS063)

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 Feb 2025 to Jun 2025

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

CERTIFICATE

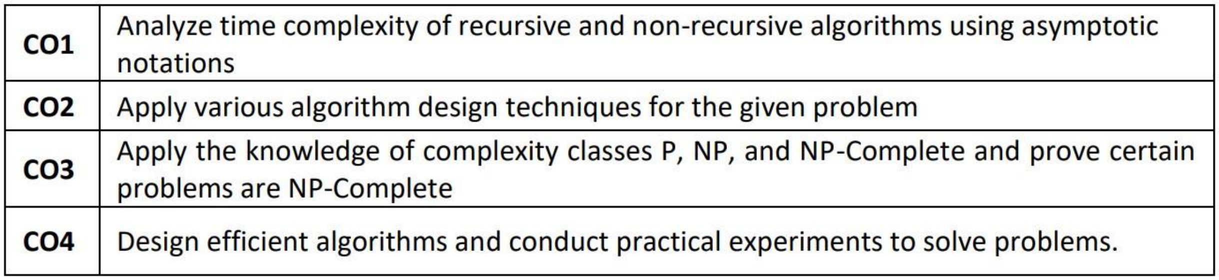
This is to certify that the Lab work entitled "**ANALYSIS AND DESIGN OF ALGORITHMS(23CS4PCADA)**" carried out by **BHAVYA GOYAL (1BM23CS063) ,** who is a bonafide student of **B. M. S. College of Engineering.** It is in partial fulfillment for the award of **Bachelor of Engineering in Computer Science and Engineering** of the Visvesvaraya Technological University, Belgaum during the year 2025. The Lab report has been approved as it satisfies the academic requirements in respect of **ANALYSIS AND DESIGN OF ALGORITHMS(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 Sheet

Course Outcome



CO1	Analyze time complexity of recursive and non-recursive algorithms using asymptotic notations
CO2	Apply various algorithm 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.

Question- 1:

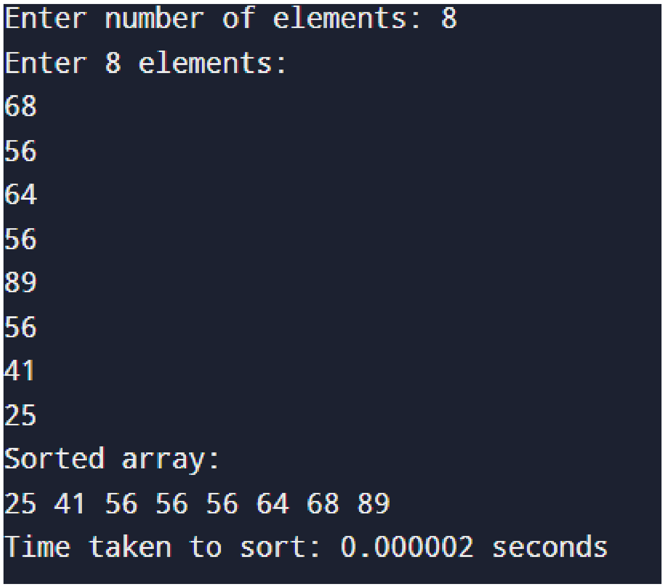
Sort N Number using Merge Sort

Code:

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

// Merge function
void merge(int arr[], int left, int mid, int right) {
    int n1 = mid - left + 1;
    int n2 = right - mid;

    int leftArr[n1], rightArr[n2];

    for (int i = 0; i < n1; i++)
        leftArr[i] = arr[left + i];
    for (int j = 0; j < n2; j++)
    rightArr[j] = arr[mid + 1 + j];

int i = 0, j = 0, k = left;

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

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

while (j < n2) {
    arr[k] = rightArr[j];
```

```
        j++;
        k++;
    }
}

// Merge Sort function
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, mid, right);
    }
}

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

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

    clock_t start, end;
    start = clock();

    mergeSort(arr, 0, n - 1);

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

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

    printf("Time taken to sort: %f seconds\n", time_taken);
    return 0;
}
```

Output:



```
Enter number of elements: 8
Enter 8 elements:
68
56
64
56
89
56
41
25
Sorted array:
25 41 56 56 56 64 68 89
Time taken to sort: 0.000002 seconds
```

ii) using source removal method

CODE:

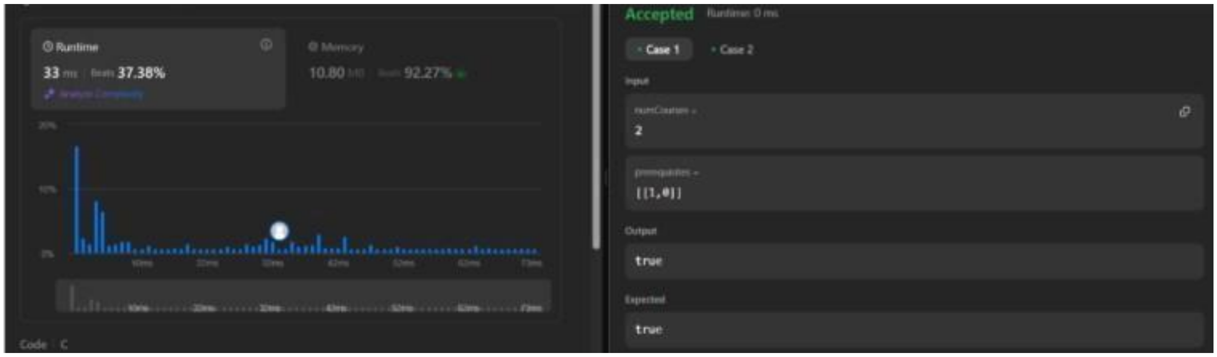
```
#include<stdio.h> int
a[10][10],n,t[10],indegree[10]; int
stack[10],top=-1; void
computeIndegree(int,int [][10]); void
tps_SourceRemoval(int,int [][10]);
int main(){

printf("Enter the no. of nodes: ");


scanf("%d",&n);
int i,j;
for(i=0;i<n;i++){ for(j=0;j<n;j++){
scanf("%d",&a[i][j]);
} }
computeIndegree(n,a);
tps_SourceRemoval(n,
a);
```

```
printf("Solution:"); for(i=0;i<n;i++){
printf("%d ",t[i]); } return 0; } void
computeIndegree(int n,int a[][10]){
int i,j,sum=0; for(i=0;i<n;i++){

sum=0;
for(j=0;j<n;j++){
sum=sum+a[j][i]; }
indegree[i]=sum;

} } void tps_SourceRemoval(int n,int
a[][10]){

int i,j,v;
for(i=0;i<n;i++){
if(indegree[i]==0){
stack[++top]=i;

}
}
```

```
int k=0; while(top!=-1){
v=stack[top--]; t[k++]=v;
for(i=0;i<n;i++){
if(a[v][i]!=0){
indegree[i]=indegree[i]-
1; if(indegree[i]==0){
stack[++top]=i; }

}
}
```

```
 }
 }
```

OUTPUT:



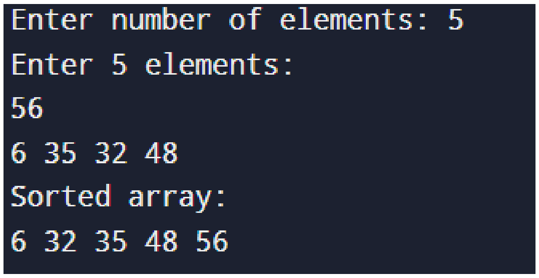
```
Runtime
33 ms | Beats 37.38%
Memory
10.80 MB | Beats 92.27%

Accepted   Runtime: 0 ms
Case 1   Case 2
Input
numCourses =
2
prerequisites =
[[1,0]]
Output
true
Expected
true
```

Question- 2: Sort a given set of N integer elements using Quick Sort technique and compute its time taken.

Code:

```
#include <stdio.h>


// Function to partition the array
int partition(int a[], int low, int high) {
    int pivot = a[low];
    int i = low + 1;
    int j = high;
    int temp;


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

    // Swap pivot with a[j]
    temp = a[low];
    a[low] = a[j];
    a[j] = temp;

    return j;
}

// Recursive Quick Sort function
void quicksort(int a[], int low, int high) {
    if (low < high) {
        int pi = partition(a, low, high);
        quicksort(a, low, pi - 1);
        quicksort(a, pi + 1, high);
    }
}

// Main function to test the quicksort
int main() {
    int n;
    printf("Enter number of elements: ");
    scanf("%d", &n);

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

    quicksort(a, 0, n - 1);

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

    return 0;
}
```

Output:



```
Enter number of elements: 5
Enter 5 elements:
56
6 35 32 48
Sorted array:
6 32 35 48 56
```

Question- 3:

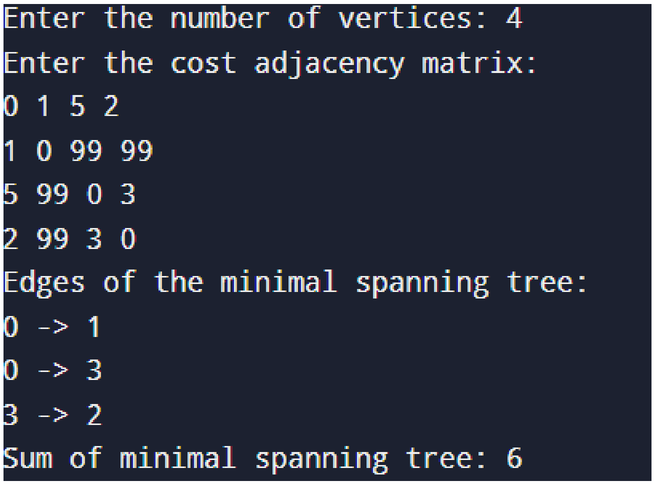
Find Minimum Cost Spanning Tree of a given undirected graph using Prim's algorithm.

Code:

```
#include <stdio.h>

int cost[10][10], n, f[10][2], sum;

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]);
            if (cost[i][j] == 0)
                cost[i][j] = 999;  // 999 represents infinity
        }
    }

    prims(cost, n);

    printf("Edges of the minimal spanning tree:\n");
    for (i = 1; i < n; i++)
        printf("%d -> %d\n", f[i][0], f[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, min, source;
    int p[10], d[10], s[10];
```

```
min = 999;
source = 0;

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

s[source] = 1;  // Include source in MST
sum = 0;

for (i = 1; i < n; i++) {
    min = 999;

    // Find the vertex v not in MST with minimum d[v]
    for (j = 0; j < n; j++) {
        if (!s[j] && d[j] < min) {
            min = d[j];
            v = j;
        }
        }


        u = p[v];

        f[i][0] = u;

        f[i][1] = v;

        sum += cost[u][v];

        s[v] = 1;


        // Update the distances

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

            if (!s[j] && cost[v][j] < d[j]) {

                d[j] = cost[v][j];

                p[j] = v;

            }

        }

    }

}
```

Output:



```
Enter the number of vertices: 4
Enter the cost adjacency matrix:
0 1 5 2
1 0 99 99
5 99 0 3
2 99 3 0
Edges of the minimal spanning tree:
0 -> 1
0 -> 3
3 -> 2
Sum of minimal spanning tree: 6
```

Question- 4:

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

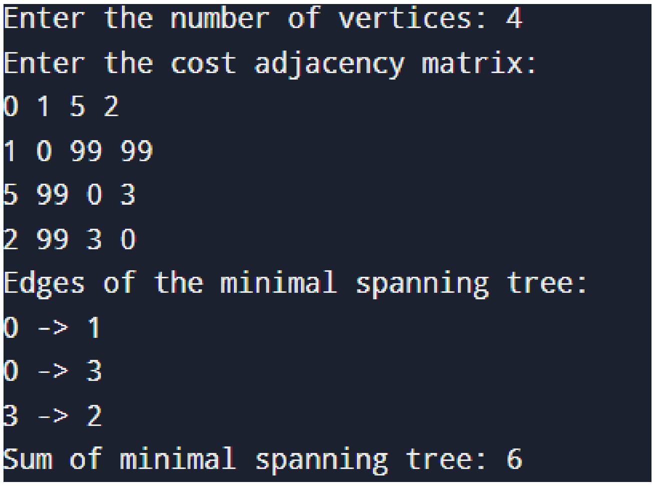
Code:

```
#include <stdio.h>


int cost[10][10], n, t[10][2], sum;


int final(int parent[10], int i) {
    while (parent[i])
```

```
        i = parent[i];
    return i;
}

void kruskal(int cost[10][10], int n) {
    int min, u, v, count, k;
    int parent[10];

    k = 0;
    sum = 0;
    count = 0;

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

    while (count < n - 1) {
        min = 999;

        // Find the minimum edge
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                if (cost[i][j] < min) {
                    min = cost[i][j];
                    u = i;
                    v = j;
                }
            }
        }

        // Check if it forms a cycle
        int x = final(parent, u);
        int y = final(parent, v);

        if (x != y) {
            t[k][0] = u;
            t[k][1] = v;
            k++;
            sum += cost[u][v];
            parent[y] = x;
            count++;
        }

        cost[u][v] = cost[v][u] = 999;  // Mark as visited
    }
}
```

```
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]);
            if (cost[i][j] == 0)
                cost[i][j] = 999;
        }
    }

    kruskal(cost, n);

    printf("Edges of the minimal spanning tree are:\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;

}
```

Output:



```
Enter the number of vertices: 4
Enter the cost adjacency matrix:
0 1 5 2
1 0 99 99
5 99 0 3
2 99 3 0
Edges of the minimal spanning tree:
0 -> 1
0 -> 3
3 -> 2
Sum of minimal spanning tree: 6
```

Question- 5:

Implement 0/1 Knapsack problem using dynamic programming.

Code:

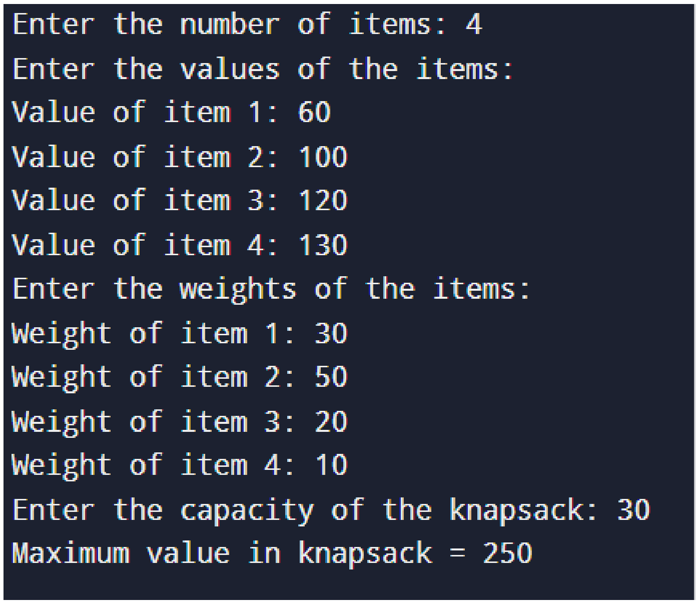
```
#include <stdio.h>

#include <stdlib.h>


// Function to return max of two integers

int max(int a, int b) {

    return (a > b) ? a : b;

}
```

```
// Function to solve 0/1 Knapsack problem
int knapsack(int capacity, int wt[], int val[], int n) {
    int i, w;
    int K[n + 1][capacity + 1];

    for (i = 0; i <= n; i++) {
        for (w = 0; w <= capacity; w++) {
            if (i == 0 || w == 0)
                K[i][w] = 0;
            else if (wt[i - 1] <= w)
                K[i][w] = max(val[i - 1] + K[i - 1][w - wt[i - 1]], K[i - 1][w]);
            else
                K[i][w] = K[i - 1][w];
        }
    }

    return K[n][capacity];
}

int main() {
    int n, capacity;
```

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

int val[n], wt[n];

printf("Enter the values of the items:\n");
for (int i = 0; i < n; i++) {
    printf("Value of item %d: ", i + 1);
    scanf("%d", &val[i]);
}

printf("Enter the weights of the items:\n");
for (int i = 0; i < n; i++) {
    printf("Weight of item %d: ", i + 1);
    scanf("%d", &wt[i]);
}

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

int maxValue = knapsack(capacity, wt, val, n);
printf("Maximum value in knapsack = %d\n", maxValue);
```

```
    return 0;
}
```

Output:



```
Enter the number of items: 4
Enter the values of the items:
Value of item 1: 60
Value of item 2: 100
Value of item 3: 120
Value of item 4: 130
Enter the weights of the items:
Weight of item 1: 30
Weight of item 2: 50
Weight of item 3: 20
Weight of item 4: 10
Enter the capacity of the knapsack: 30
Maximum value in knapsack = 250
```

Question- 6:

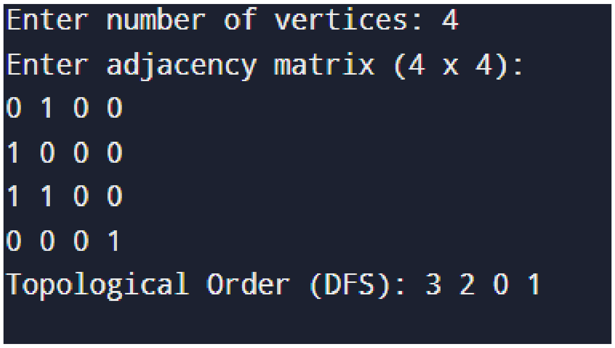
Write program to obtain the Topological ordering of vertices in a given digraph.

Code:

```
#include <stdio.h>
#define MAX 10

int adj[MAX][MAX];
int visited[MAX], stack[MAX], top = -1;
```

```
void dfs(int v, int n) {
    visited[v] = 1;

    for (int i = 0; i < n; i++) {
        if (adj[v][i] == 1 && !visited[i]) {
            dfs(i, n);
        }
    }

    // Push to stack after all adjacent nodes are visited
    stack[++top] = v;
}

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

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

int main() {
    int n;

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

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

    topologicalSortDFS(n);

    return 0;
}
```

Output:



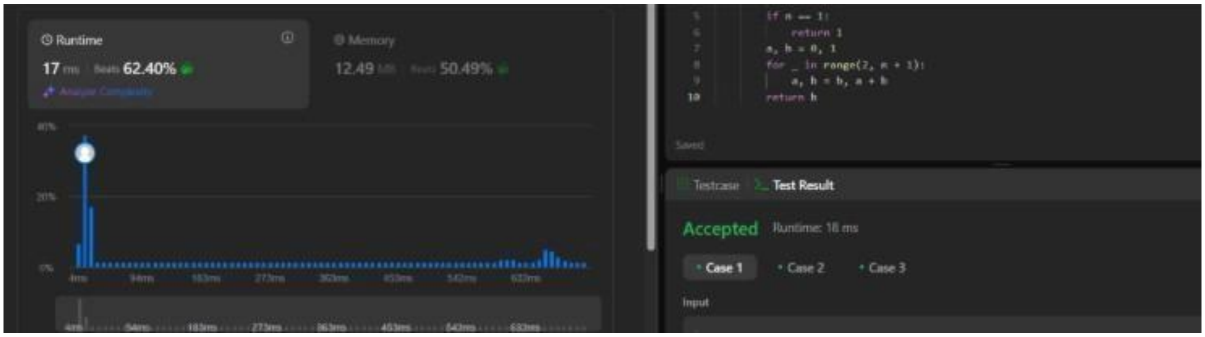
```
Enter number of vertices: 4
Enter adjacency matrix (4 x 4):
0 1 0 0
1 0 0 0
1 1 0 0
0 0 0 1
Topological Order (DFS): 3 2 0 1
```

LeetCode Program related to Knapsack problem or Dynamic Programming.

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
```

OUTPUT:



Runtime
17 ms Beats 62.40%
Analyze Complexity

Memory
12.49 MB Beats 50.49%

```
        if n == 1:
            return 1
        a, b = 0, 1
        for _ in range(2, n + 1):
            a, b = b, a + b
        return b
```

Saved

Testcase | Test Result

Accepted Runtime: 18 ms

Case 1 · Case 2 · Case 3

Input

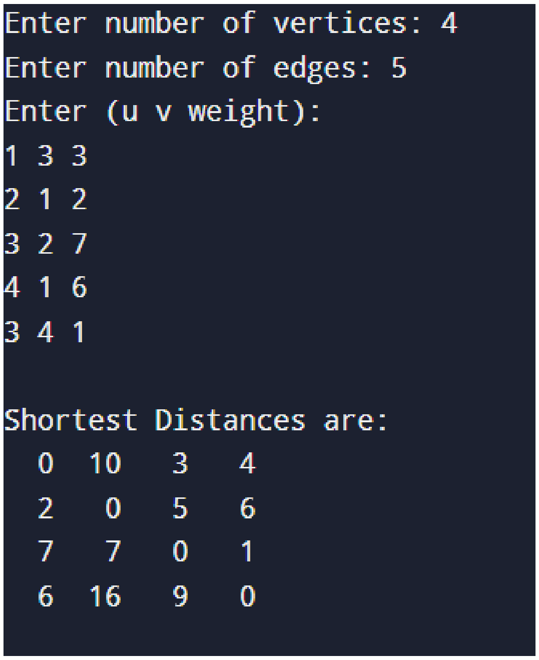
Question- 7:

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

Code:

```
#include <stdio.h>
#define INF 99999
#define MAX 100
void floydWarshall(int graph[MAX][MAX], int n) {
    int dist[MAX][MAX];
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            dist[i][j] = graph[i][j];
    for (int k = 0; k < n; k++) {
        for (int i = 0; i < n; i++) {
            for (int 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 shortest distances
    printf("\nShortest Distances are:\n");
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            if (dist[i][j] == INF)
                printf("INF ");
            else
                printf("%3d ", dist[i][j]);
        }
        printf("\n");
    }
}

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

    printf("Enter number of vertices: ");
    scanf("%d", &n);
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            graph[i][j] = (i == j) ? 0 : INF;
    printf("Enter number of edges: ");
    scanf("%d", &e);
    printf("Enter (u v weight):\n");
    for (int i = 0; i < e; i++) {
        int u, v, w;
        scanf("%d %d %d", &u, &v, &w);
        graph[u - 1][v - 1] = w;
    }
    floydWarshall(graph, n);
    return 0;
}
```

Output:



```
Enter number of vertices: 4
Enter number of edges: 5
Enter (u v weight):
1 3 3
2 1 2
3 2 7
4 1 6
3 4 1

Shortest Distances are:
  0  10   3   4
  2   0   5   6
  7   7   0   1
  6  16   9   0
```

LeetCode Program related to shortest distance calculation.

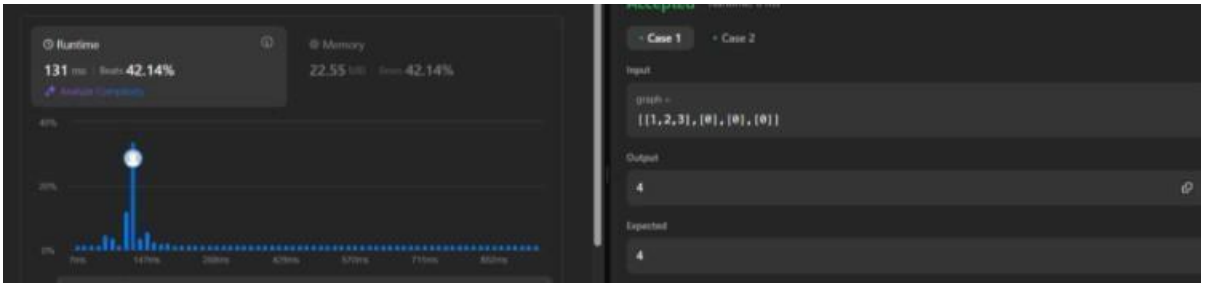
CODE:

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
```

Output:



131 ms | Beats 42.14%

22.55 MB | Beats 42.14%

graph =
[[1,2,3],[0],[0],[0]]

4

4

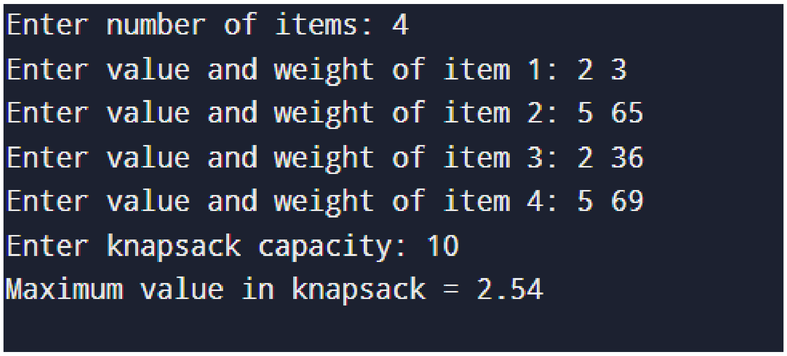
Question- 8:

Implement Fractional Knapsack using Greedy technique.

Code:

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


struct Item {
    int value, weight;
};
int compare(const void *a, const void *b) {
    double r1 = (double)((struct Item *)a)->value / ((struct Item *)a)->weight;
    double r2 = (double)((struct Item *)b)->value / ((struct Item *)b)->weight;
    return (r2 > r1) - (r2 < r1); // descending order
}
double fractionalKnapsack(int capacity, struct Item items[], int n) {
    qsort(items, n, sizeof(struct Item), compare);


    double totalValue = 0.0;


    for (int i = 0; i < n; i++) {
        if (capacity >= items[i].weight) {
            capacity -= items[i].weight;
            totalValue += items[i].value;
        } else {
            totalValue += ((double)items[i].value * capacity / items[i].weight);
            break;
        }
    }

    return totalValue;
}

int main() {
    int n, capacity;

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

    struct Item items[n];

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

    double maxValue = fractionalKnapsack(capacity, items, n);
    printf("Maximum value in knapsack = %.2f\n", maxValue);

    return 0;
}
```

Output:



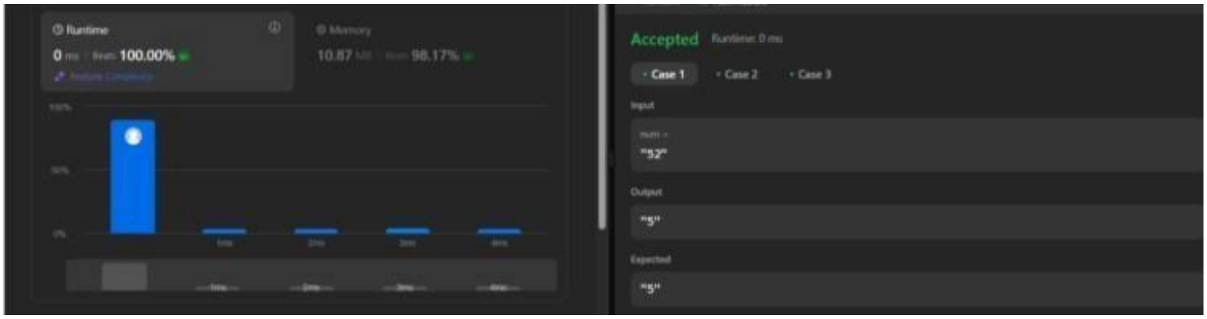
```
Enter number of items: 4
Enter value and weight of item 1: 2 3
Enter value and weight of item 2: 5 65
Enter value and weight of item 3: 2 36
Enter value and weight of item 4: 5 69
Enter knapsack capacity: 10
Maximum value in knapsack = 2.54
```

LeetCode Program related to Greedy Technique algorithms.

CODE:

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

32| Page 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

}
```

OUTPUT:



0 ms | Beats 100.00%

10.87 MB | Beats 98.17%

Accepted Runtime: 0 ms

Case 1 · Case 2 · Case 3

Input

num =
"52"

Output

"5"

Expected

"5"

Question- 9: From a given vertex in a weighted connected graph, find shortest paths to other vertices using Dijkstra's algorithm.

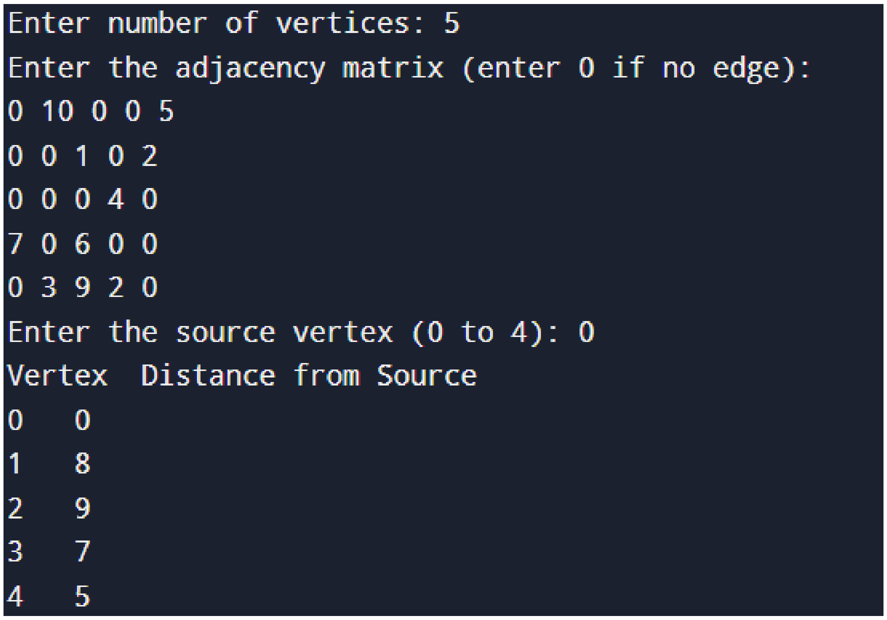
Code:

```
#include <stdio.h>

#include <limits.h>

#include <stdbool.h>


#define MAX 100

#define INF INT_MAX


// Function to find the vertex with the minimum distance

int minDistance(int dist[], bool visited[], int n) {

   int min = INF, min_index;

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

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

         min = dist[v];

         min_index = v;

      }

   }

   return min_index;

}


// Function to print the shortest path distances
```

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

// Dijkstra's algorithm implementation
void dijkstra(int graph[MAX][MAX], int n, int src) {
    int dist[MAX];      // Output array. dist[i] holds shortest distance from src to i
    bool visited[MAX];  // visited[i] is true if vertex i is included in the shortest path

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

    dist[src] = 0; // Distance of source vertex to itself is always 0

    // Find shortest path for all vertices
    for (int count = 0; count < n - 1; count++) {
        int u = minDistance(dist, visited, n);
        visited[u] = true;

        // Update distance of adjacent vertices
        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];
            }
        }
    }

    printSolution(dist, n);
}

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

    int graph[MAX][MAX];
    printf("Enter the adjacency matrix (enter 0 if no edge):\n");
    for (int i = 0; i < n; i++)
```

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

    int src;
    printf("Enter the source vertex (0 to %d): ", n - 1);
    scanf("%d", &src);

    dijkstra(graph, n, src);

    return 0;
}
```

Output:



```
Enter number of vertices: 5
Enter the adjacency matrix (enter 0 if no edge):
0 10 0 0 5
0 0 1 0 2
0 0 0 4 0
7 0 6 0 0
0 3 9 2 0
Enter the source vertex (0 to 4): 0
Vertex  Distance from Source
0   0
1   8
2   9
3   7
4   5
```

Question- 10:

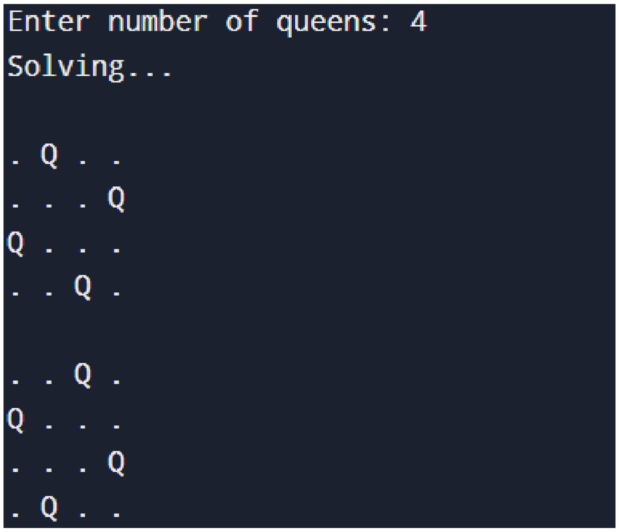
Implement “N-Queens Problem” using Backtracking.

Code:

```
#include <stdio.h>

#include <stdlib.h>

#include <math.h>


#define MAX 10

int board[MAX];


void printBoard(int n) {

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

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

            if (board[i] == j)

                printf("Q ");

            else

                printf(". ");

        }

        printf("\n");

    }

    printf("\n");

}
```

```
int isSafe(int row, int col) {

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

      if (board[i] == col || abs(board[i] - col) == abs(i - row))

         return 0;

   }

   return 1;

}


void NQueens(int row, int n) {

   if (row == n) {

      printBoard(n);

      return;

   }


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

      if (isSafe(row, col)) {

         board[row] = col;

         NQueens(row + 1, n);

      }

   }

}


int main() {
   int n;
   printf("Enter number of queens: ");
   scanf("%d", &n);
   printf("Solving...\n\n");
   NQueens(0, n);
   return 0;
}
```

Output:



```
Enter number of queens: 4
Solving...

. Q . .
. . . Q
Q . . .
. . Q .

. . Q .
Q . . .
. . . Q
. Q . .
```

Question- 11:

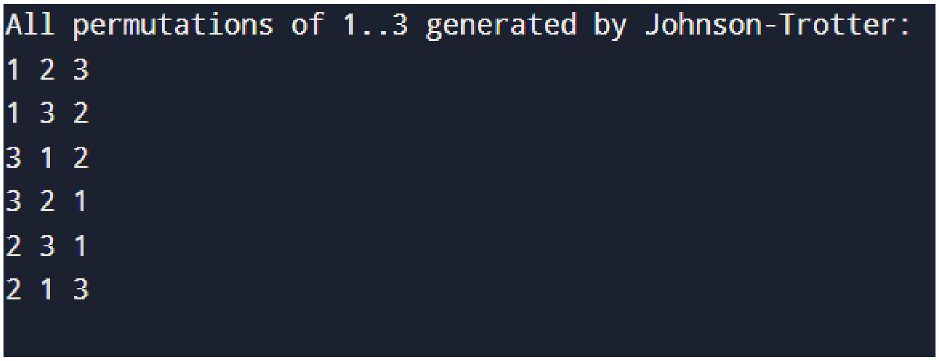
Implement Johnson Trotter algorithm to generate permutations.

Code:

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

#define LEFT -1
#define RIGHT 1

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

int getMobile(int *perm, int *dir, int n) {
    int mobile_prev = 0;
    int mobile_index = -1;
    for (int i = 0; i < n; i++) {
        int next_pos = i + dir[i];
        if (next_pos >= 0 && next_pos < n) {
            if (perm[i] > perm[next_pos] && perm[i] > mobile_prev) {
                mobile_prev = perm[i];
                mobile_index = i;
            }
        }
    }
    return mobile_index;
}

void johnsonTrotter(int n) {
    int *perm = malloc(n * sizeof(int));
    int *dir = malloc(n * sizeof(int));

    for (int i = 0; i < n; i++) {
        perm[i] = i + 1;
        dir[i] = LEFT;
    }

    printPermutation(perm, n);

    while (1) {
        int mobile = getMobile(perm, dir, n);
        if (mobile == -1)
            break;
        int swap_pos = mobile + dir[mobile];

        int temp = perm[mobile];
        perm[mobile] = perm[swap_pos];
        perm[swap_pos] = temp;

        int temp_dir = dir[mobile];
        dir[mobile] = dir[swap_pos];
        dir[swap_pos] = temp_dir;

        mobile = swap_pos;

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

        printPermutation(perm, n);
    }

    free(perm);
    free(dir);
```

```
}


int main() {
    int n = 3;
    printf("All permutations of 1..%d generated by Johnson-Trotter:\n", n);
    johnsonTrotter(n);
    return 0;
}
```

Output:



```
All permutations of 1..3 generated by Johnson-Trotter:
1 2 3
1 3 2
3 1 2
3 2 1
2 3 1
2 1 3
```

Question- 12:

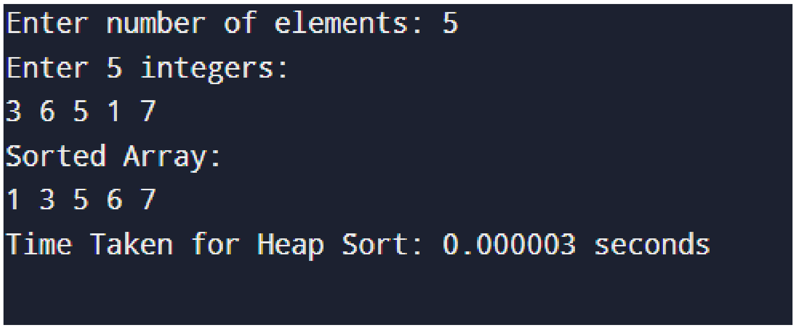
Sort a given set of N integer elements using Heap Sort technique and compute its time taken.

Code:

```
#include <stdio.h>
#include <time.h>
void heapify(int arr[], int n, int i) {
   int largest = i;        // Initialize largest as root
   int left = 2 * i + 1;    // left child
   int right = 2 * i + 2;   // right child

   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) {
    // Build heap (rearrange array)
    for (int i = n / 2 - 1; i >= 0; i--)
        heapify(arr, n, i);
    for (int i = n - 1; i > 0; i--) {
        // Move current root to end
        int temp = arr[0];
        arr[0] = arr[i];
        arr[i] = temp;
        heapify(arr, i, 0);
    }
}

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

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


    heapSort(arr, n);


    clock_t end = clock();


    printf("Sorted Array:\n");
    for (int i = 0; i < n; i++)
        printf("%d ", arr[i]);
    printf("\n");
    double time_taken = (double)(end - start) / CLOCKS_PER_SEC;
    printf("Time Taken for Heap Sort: %.6f seconds\n", time_taken);


    return 0;
}
```

Output:



```
Enter number of elements: 5
Enter 5 integers:
3 6 5 1 7
Sorted Array:
1 3 5 6 7
Time Taken for Heap Sort: 0.000003 seconds
```