**VISVESVARAYA TECHNOLOGICAL UNIVERSITY**

**“JnanaSangama”, Belgaum -590014, Karnataka.**

****

**LAB REPORT**

**On**

**ANALYSIS AND DESIGN OF ALGORITHMS (23CS4PCADA)**

**Submitted by**

**K L SRUJAN (1BM24CS408)**

**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 L SRUJAN**(1BM24CS408)**, 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. Ramya KM Dr. Kavitha Sooda**

Assistant Professor Professor and Head

Department of CSE Department of CSE

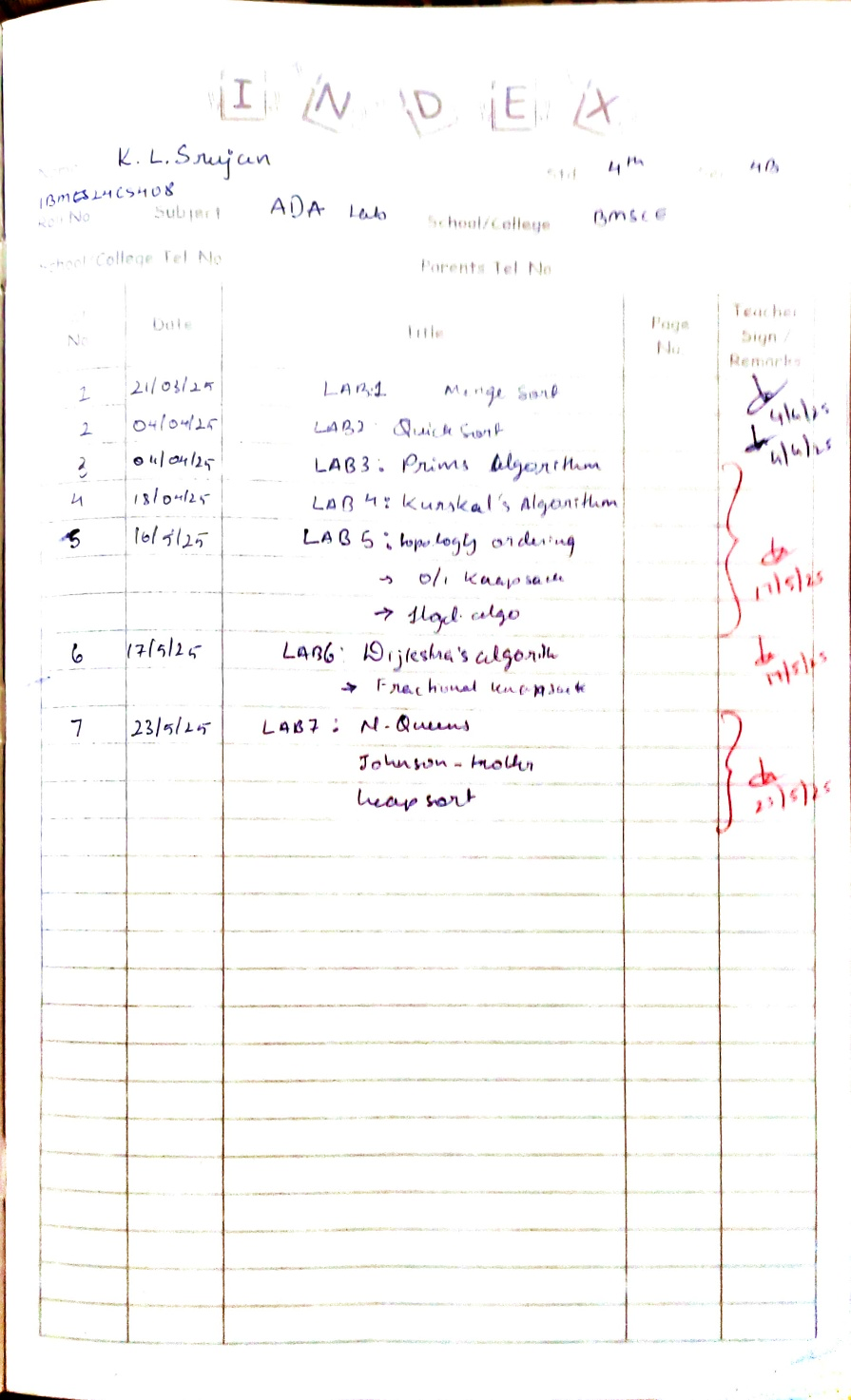
BMSCE, Bengaluru BMSCE, Bengaluru

**Index Sheet**

|  |  |  |
| --- | --- | --- |
| **Sl.No** | **Experiment Title** | **Page No.** |
| 1 | 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. | 5 |
| 2 | Sort a given set of N integer elements using Quick Sort technique and compute its time taken. | 9 |
| 3 | 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. | 11 |
| 4 | Write program to obtain the Topological ordering of vertices in a given digraph | 17 |
| 5 | Implement 0/1 Knapsack problem using dynamic programming. | 19 |
| 6 | Implement All Pair Shortest paths problem using Floyd’s  algorithm. | 22 |
| 7 | Implement Fractional Knapsack using Greedy technique. | 24 |
| 8 | From a given vertex in a weighted connected graph, find shortest paths to other vertices using Dijkstra’s algorithm. | 27 |
| 9 | Implement “N-Queens Problem” using Backtracking. | 30 |
| 10 | Implement Johnson Trotter algorithm to generate permutations. | 32 |
| 11 | Sort a given set of N integer elements using Heap Sort technique and compute its time taken. | 34 |

**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:**

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> /\* To recognize exit function when compiling with gcc \*/

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("\nTime 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] = random(1000); // Using the correct way to populate the array

a[i] = n - i; // Reverse order to maximize sorting time

}

start = clock();

split(a, 0, n - 1);

// Dummy loop to create delay

for(j = 0; j < 500000; j++) {

temp = 38 / 600;

}

end = clock();

printf("\nTime taken to sort %d numbers is %f Secs", n, (((double)(end - start)) / CLOCKS\_PER\_SEC));

n = n + 1000;

}

break;

case 3:

exit(0);

}

getchar(); // Consume the newline character left in the input buffer

}

}

void split(int a[], int low, int high) {

int mid;

if(low < high) {

mid = (low + high) / 2;

split(a, low, mid); // Recursively sort the left half

split(a, mid + 1, high); // Recursively sort the right half

combine(a, low, mid, high); // Merge the two halves

}

}

void combine(int a[], int low, int mid, int high) {

int c[15000], i, j, k;

i = k = low;

j = mid + 1;

// Merging the two halves

while(i <= mid && j <= high) {

if(a[i] < a[j]) {

c[k] = a[i];

++k;

++i;

} else {

c[k] = a[j];

++k;

++j;

}

}

// Copying the remaining elements from the left half

if(i > mid) {

while(j <= high) {

c[k] = a[j];

++k;

++j;

}

}

// Copying the remaining elements from the right half

if(j > high) {

while(i <= mid) {

c[k] = a[i];

++k;

++i;

}

}

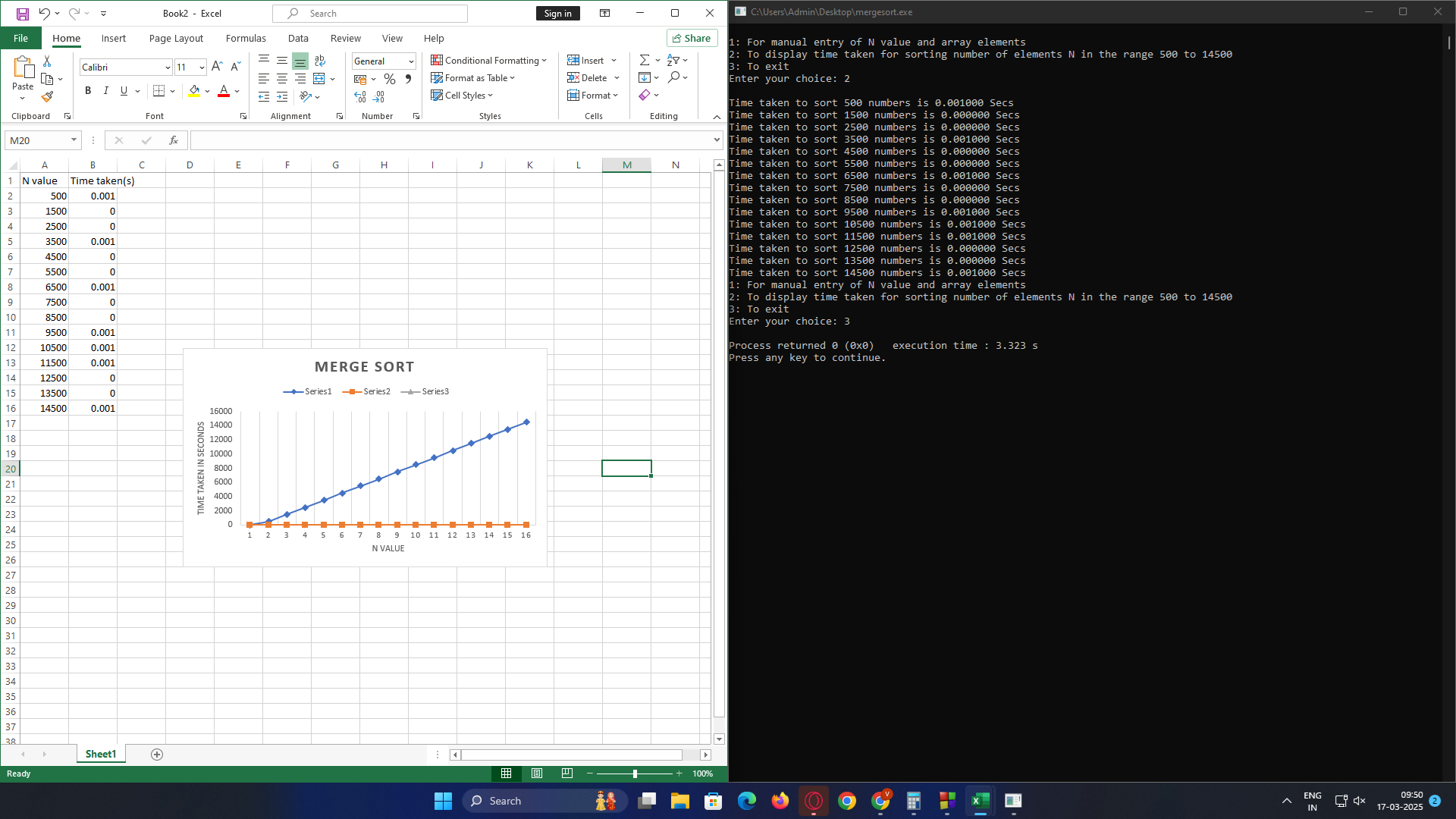
// Copying the merged array back to the original array

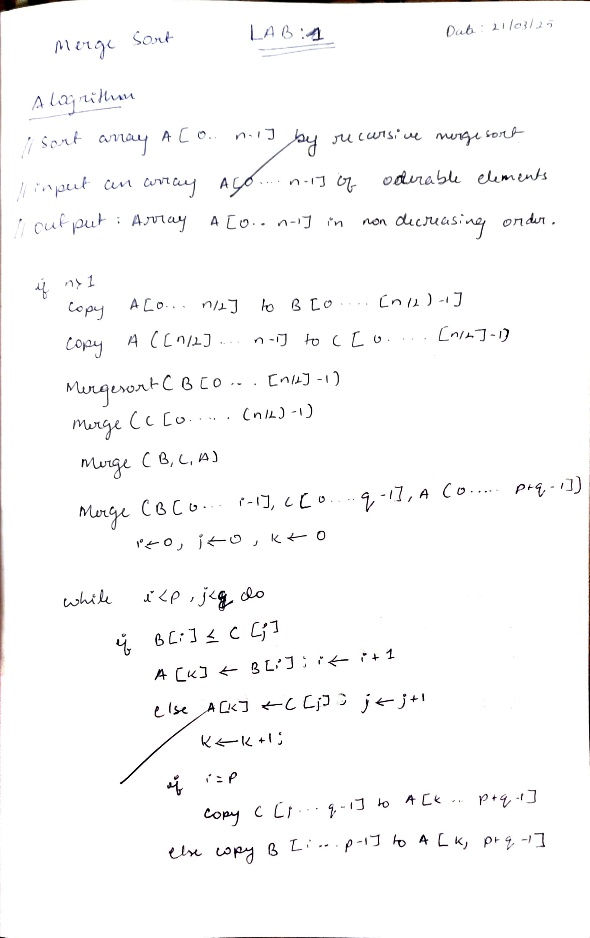
for(i = low; i <= high; i++) {

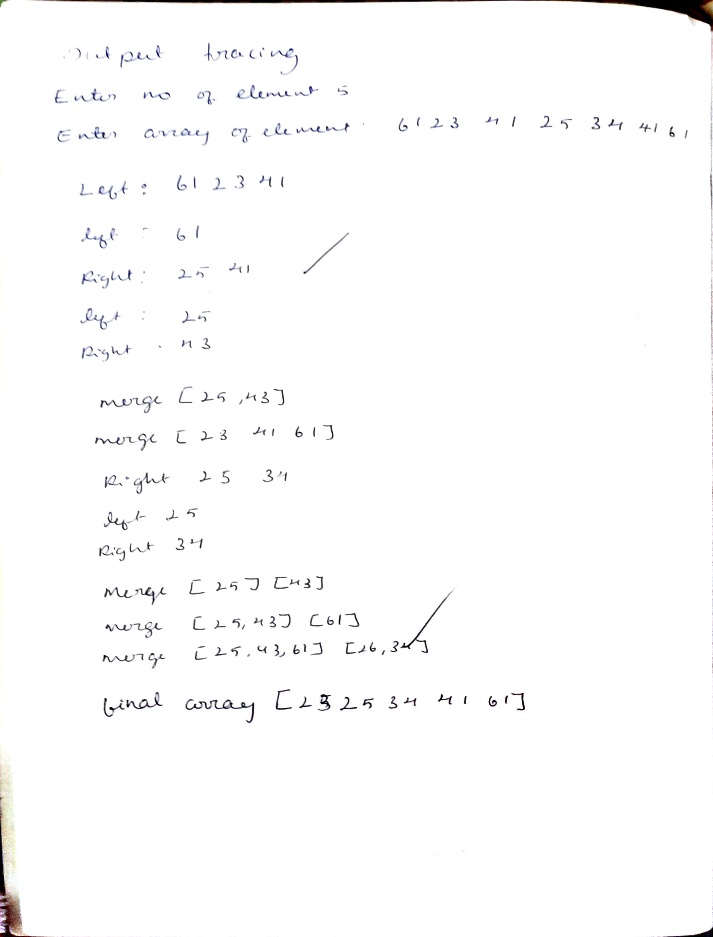
a[i] = c[i];

}

}

**OUTPUT:**







**Lab program 2:**

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

**Code:**

#include <stdio.h>

#include <stdlib.h>

#define MAX 100

int adj[MAX][MAX];

int visited[MAX];

int stack[MAX];

int top = -1;

void dfs(int v, int n) {

visited[v] = 1;

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

if (adj[v][i] && !visited[i]) {

dfs(i, n);

}

}

stack[++top] = v;

}

void topologicalSort(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, e, u, v;

printf("Enter number of vertices: ");

scanf("%d", &n);

printf("Enter number of edges: ");

scanf("%d", &e);

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

printf("Enter edge (u v): ");

scanf("%d %d", &u, &v);

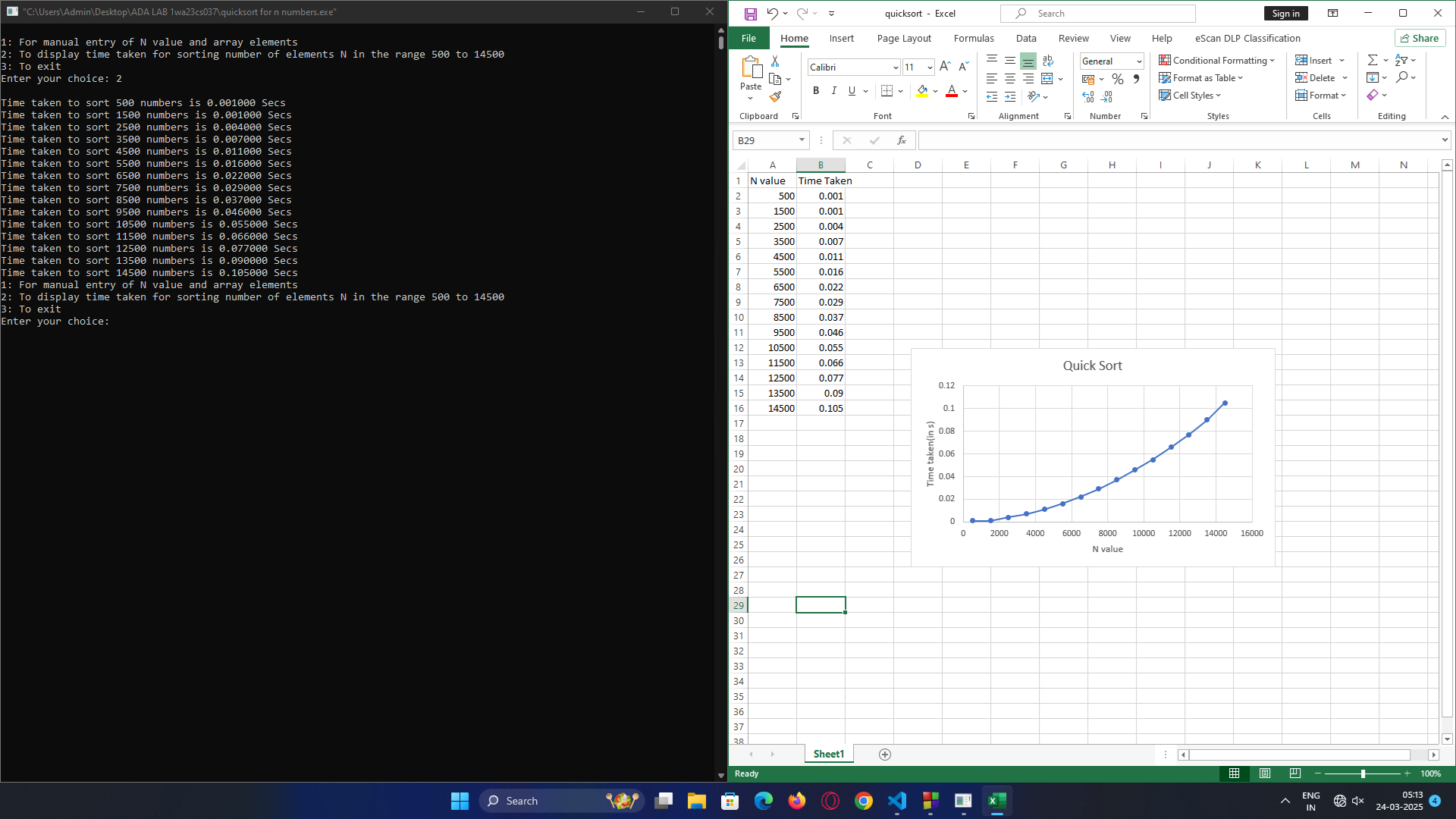
adj[u][v] = 1;

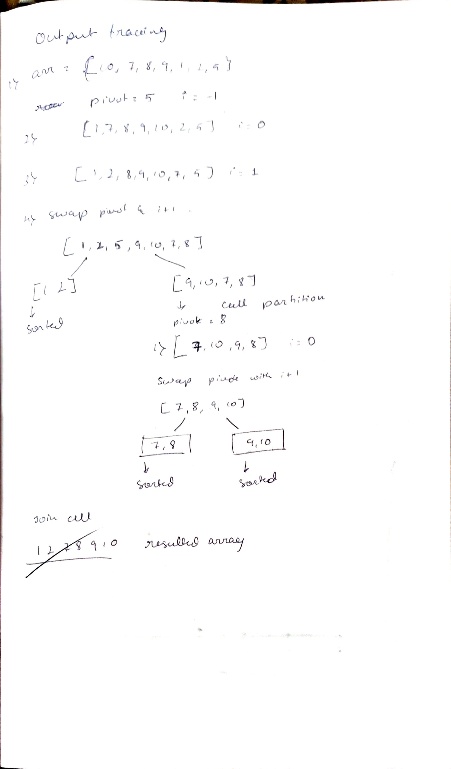
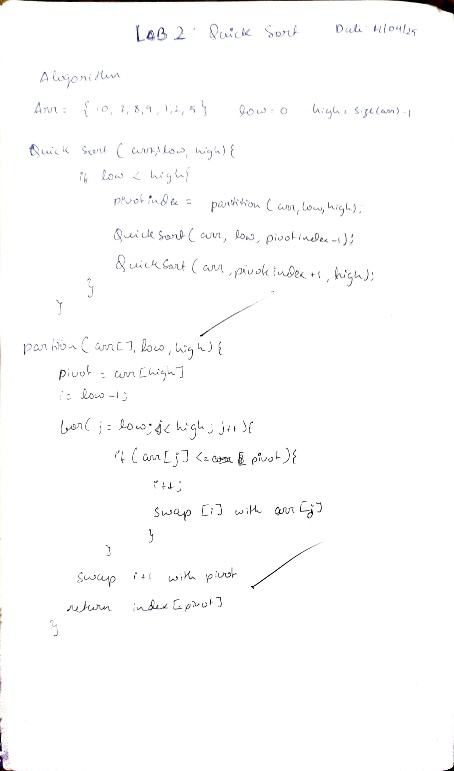
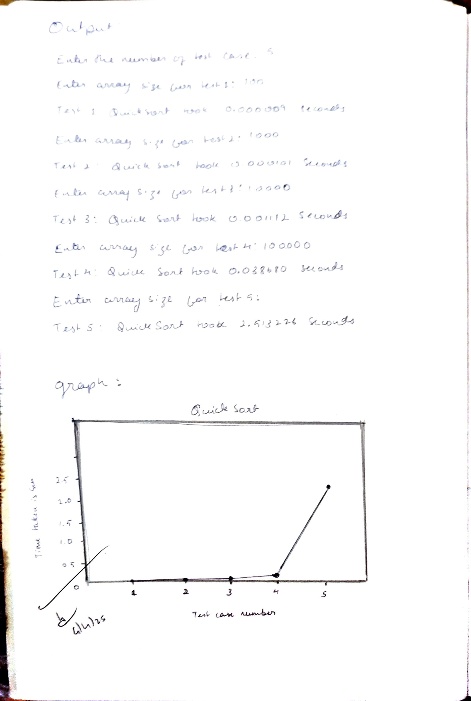
}

topologicalSort(n);

return 0;

}

**OUTPUT:**



**Lab program 3.1:**

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

**Code:**

#include <stdio.h>

#include <limits.h>

#define MAX\_VERTICES 100

void primMST(int graph[MAX\_VERTICES][MAX\_VERTICES], int V) {

int parent[MAX\_VERTICES];

int key[MAX\_VERTICES];

int inMST[MAX\_VERTICES];

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

key[i] = INT\_MAX;

inMST[i] = 0;

parent[i] = -1;

}

key[0] = 0;

parent[0] = -1;

for (int count = 0; count < V - 1; count++) {

int min = INT\_MAX, u = -1;

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

if (!inMST[v] && key[v] < min) {

min = key[v];

u = v;

}

}

inMST[u] = 1;

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

if (graph[u][v] != 0 && !inMST[v] && graph[u][v] < key[v]) {

key[v] = graph[u][v];

parent[v] = u;

}

}

}

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

int totalWeight = 0;

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

printf("%d - %d \t%d\n", parent[i], i, graph[i][parent[i]]);

totalWeight += graph[i][parent[i]];

}

printf("Total weight of MST: %d\n", totalWeight);

}

int main() {

int V, E;

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

scanf("%d", &V);

printf("Enter the number of edges: ");

scanf("%d", &E);

int graph[MAX\_VERTICES][MAX\_VERTICES] = {0};

printf("\nEnter the edges (vertex1 vertex2 weight):\n");

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

int u, v, weight;

scanf("%d %d %d", &u, &v, &weight);

graph[u][v] = weight;

graph[v][u] = weight;

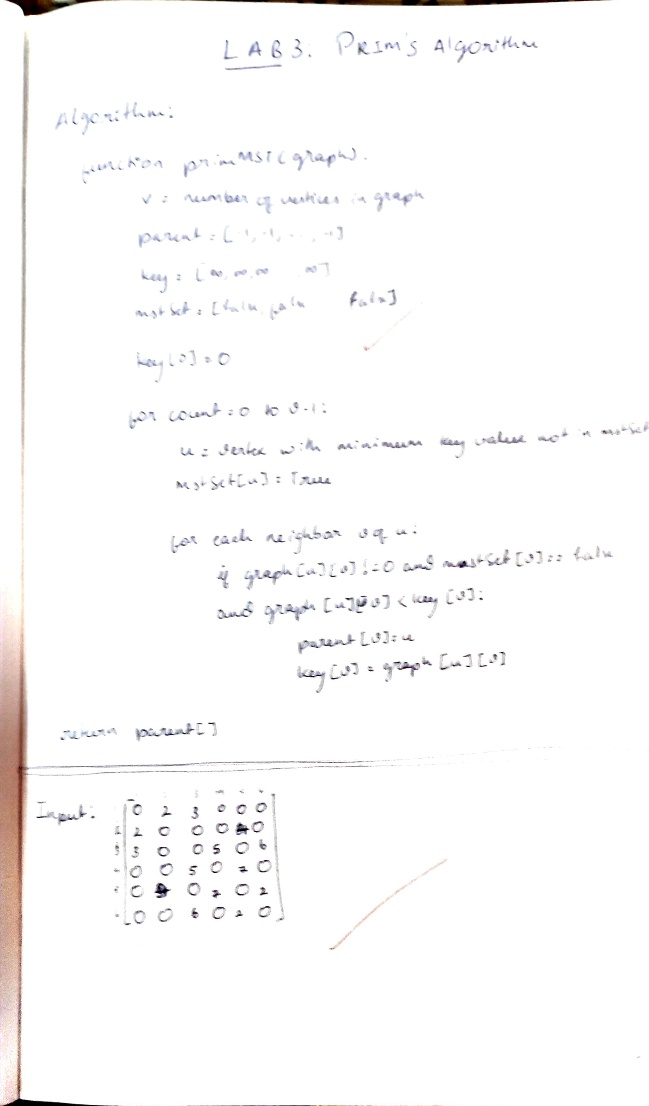
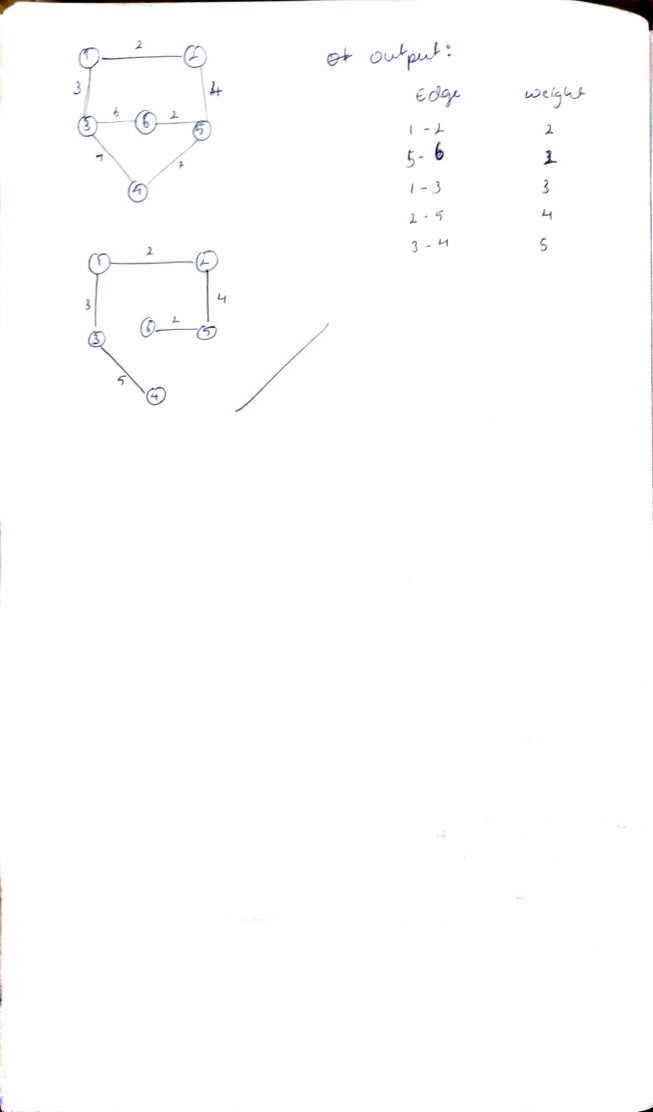
}

primMST(graph, V);

return 0;

}

**OUTPUT:**



**Lab program 3.2:**

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

**Code:**

#include <stdio.h>

#include <stdlib.h>

#define MAX 100

struct Edge {

int u, v, weight;

};

int parent[MAX];

int Find(int i) {

while (parent[i] != -1) {

i = parent[i];

}

return i;

}

void Union(int i, int j) {

int rootI = Find(i);

int rootJ = Find(j);

if (rootI != rootJ) {

parent[rootJ] = rootI;

}

}

void kruskals(int n, struct Edge edges[], int m) {

int count = 0;

int sum = 0;

struct Edge mst[MAX];

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

parent[i] = -1;

}

for (int i = 0; i < m - 1; i++) {

for (int j = i + 1; j < m; j++) {

if (edges[i].weight > edges[j].weight) {

struct Edge temp = edges[i];

edges[i] = edges[j];

edges[j] = temp;

}

}

}

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

int u = edges[i].u;

int v = edges[i].v;

int rootU = Find(u);

int rootV = Find(v);

if (rootU != rootV) {

mst[count] = edges[i];

count++;

sum += edges[i].weight;

Union(rootU, rootV);

if (count == n - 1) {

break;

}

}

}

printf("\nEdges in the Minimum Spanning Tree (MST):\n");

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

printf("%d - %d (Weight: %d)\n", mst[i].u, mst[i].v, mst[i].weight);

}

printf("Total weight of the MST: %d\n", sum);

}

int main() {

int n, m;

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

scanf("%d", &n);

printf("Enter the number of edges: ");

scanf("%d", &m);

struct Edge edges[m];

printf("Enter the edges (u v weight):\n");

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

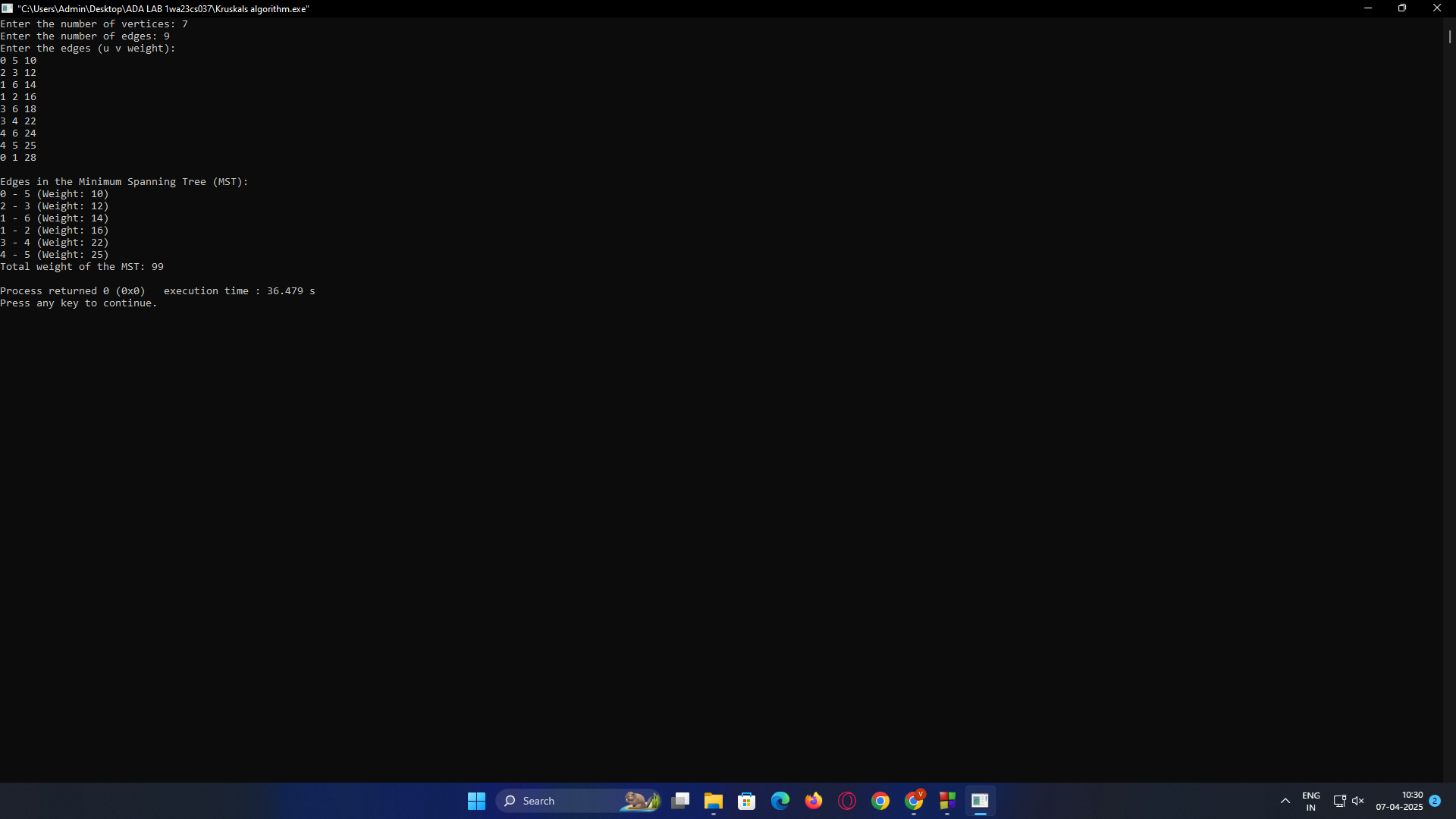
scanf("%d %d %d", &edges[i].u, &edges[i].v, &edges[i].weight);

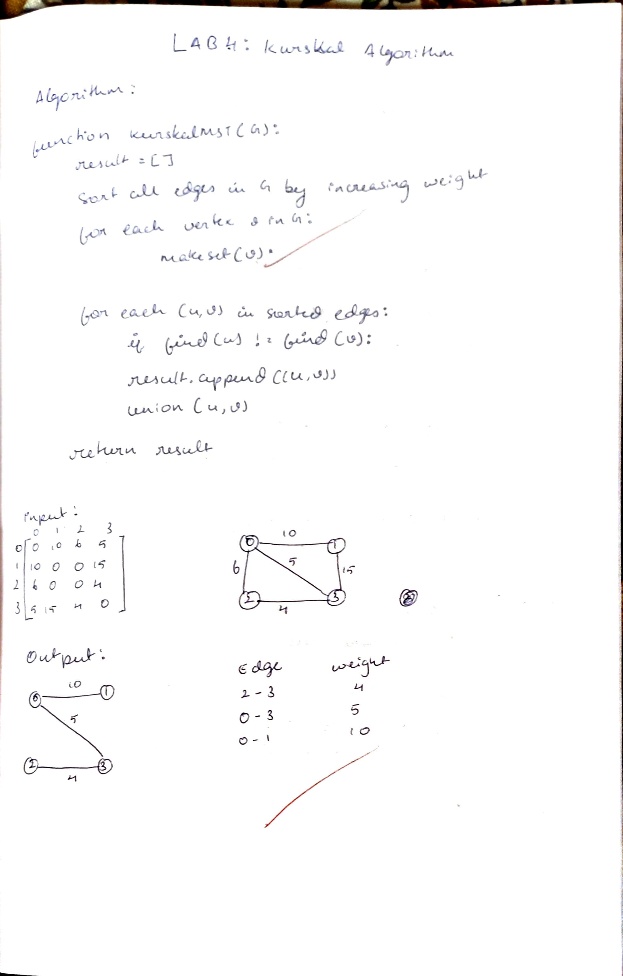
}

kruskals(n, edges, m);

return 0;

}

**OUTPUT:\**



**Lab program 4:**

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

**Code:**

#include <stdio.h>

#include <stdlib.h>

#define MAX 100

int adj[MAX][MAX];

int visited[MAX];

int stack[MAX];

int top = -1;

void dfs(int v, int n) {

visited[v] = 1;

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

if (adj[v][i] && !visited[i]) {

dfs(i, n);

}

}

stack[++top] = v;

}

void topologicalSort(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, e, u, v;

printf("Enter number of vertices: ");

scanf("%d", &n);

printf("Enter number of edges: ");

scanf("%d", &e);

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

printf("Enter edge (u v): ");

scanf("%d %d", &u, &v);

adj[u][v] = 1;

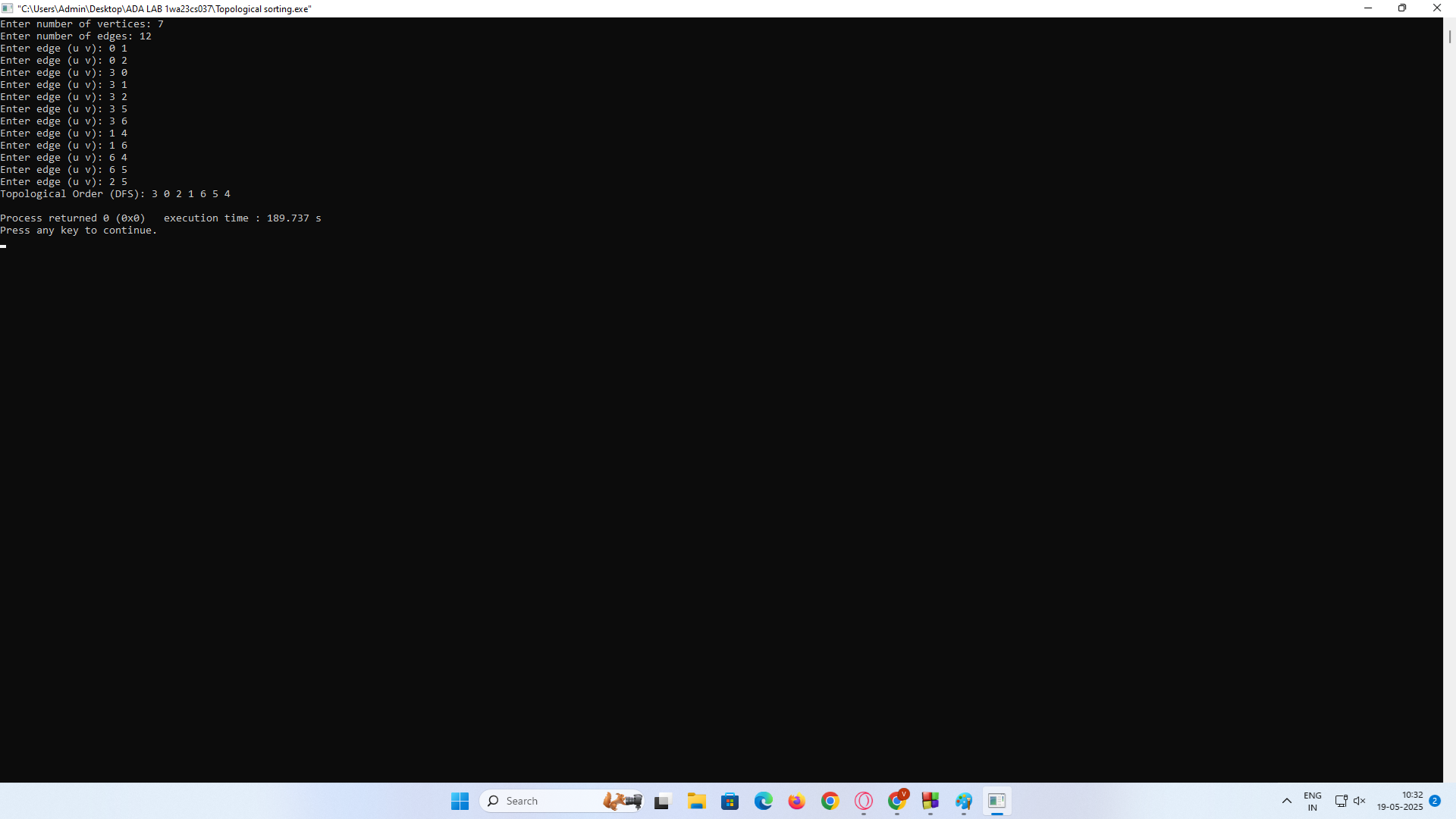
}

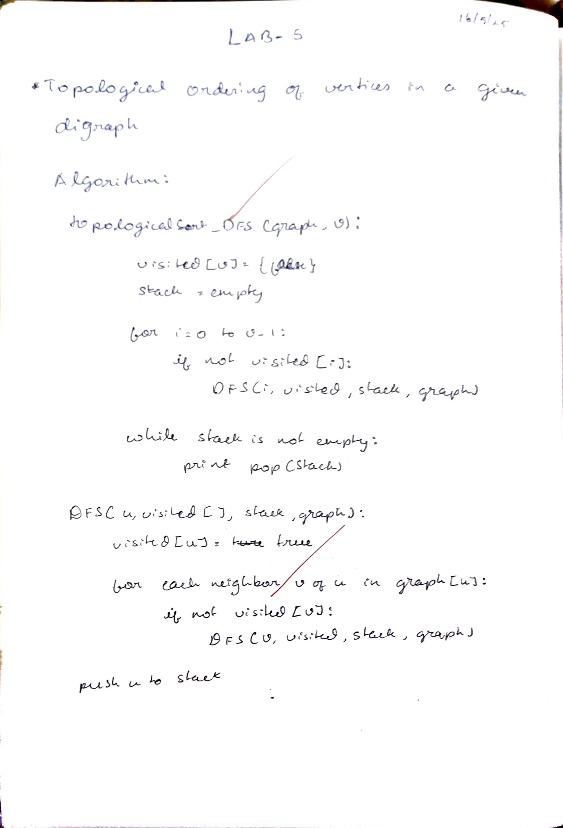
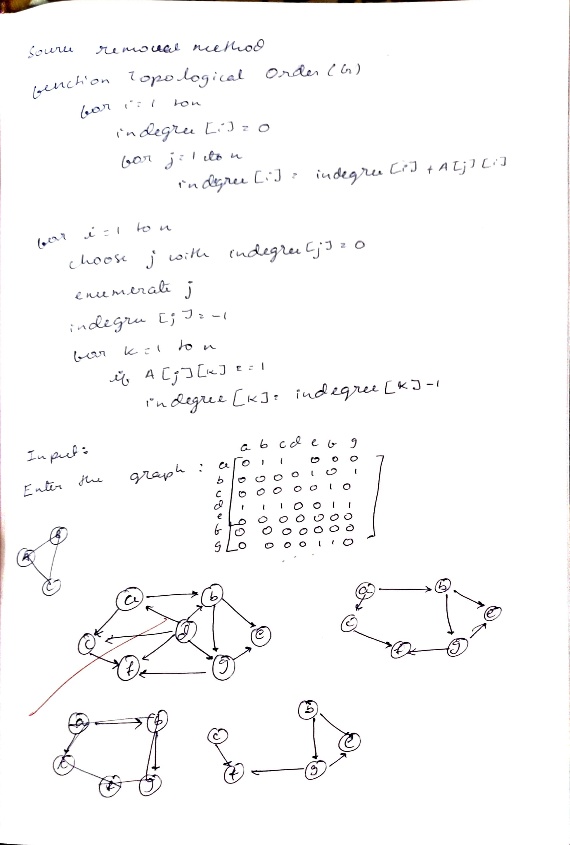
topologicalSort(n);

return 0;

}

**OUTPUT:**

****



**Lab program 5:**

Implement 0/1 Knapsack problem using dynamic programming.

**Code:**

#include <stdio.h>

int max(int a, int b) {

return (a > b) ? a : b;

}

int knapsack(int W, int wt[], int val[], int n) {

int i, w;

int K[n+1][W+1];

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

for (w = 0; w <= W; 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];

}

}

printf("\nDP matrix (rows: items, columns: capacities):\n");

printf(" ");

for (w = 0; w <= W; w++) {

printf("%4d", w);

}

printf("\n");

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

if (i == 0)

printf("0: ");

else

printf("%d: ", i);

for (w = 0; w <= W; w++) {

printf("%4d", K[i][w]);

}

printf("\n");

}

int res = K[n][W];

int capacity = W;

printf("\nSelected items (index: weight, value):\n");

for (i = n; i > 0 && res > 0; i--) {

if (res == K[i-1][capacity])

continue;

else {

printf("Item %d: weight = %d, value = %d\n", i, wt[i-1], val[i-1]);

res -= val[i-1];

capacity -= wt[i-1];

}

}

return K[n][W];

}

int main() {

int n, W;

printf("Enter the number of items: ");

scanf("%d", &n);

int wt[n], val[n];

printf("Enter the weights of the items:\n");

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

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

}

printf("Enter the values of the items:\n");

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

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

}

printf("Enter the maximum capacity of the knapsack: ");

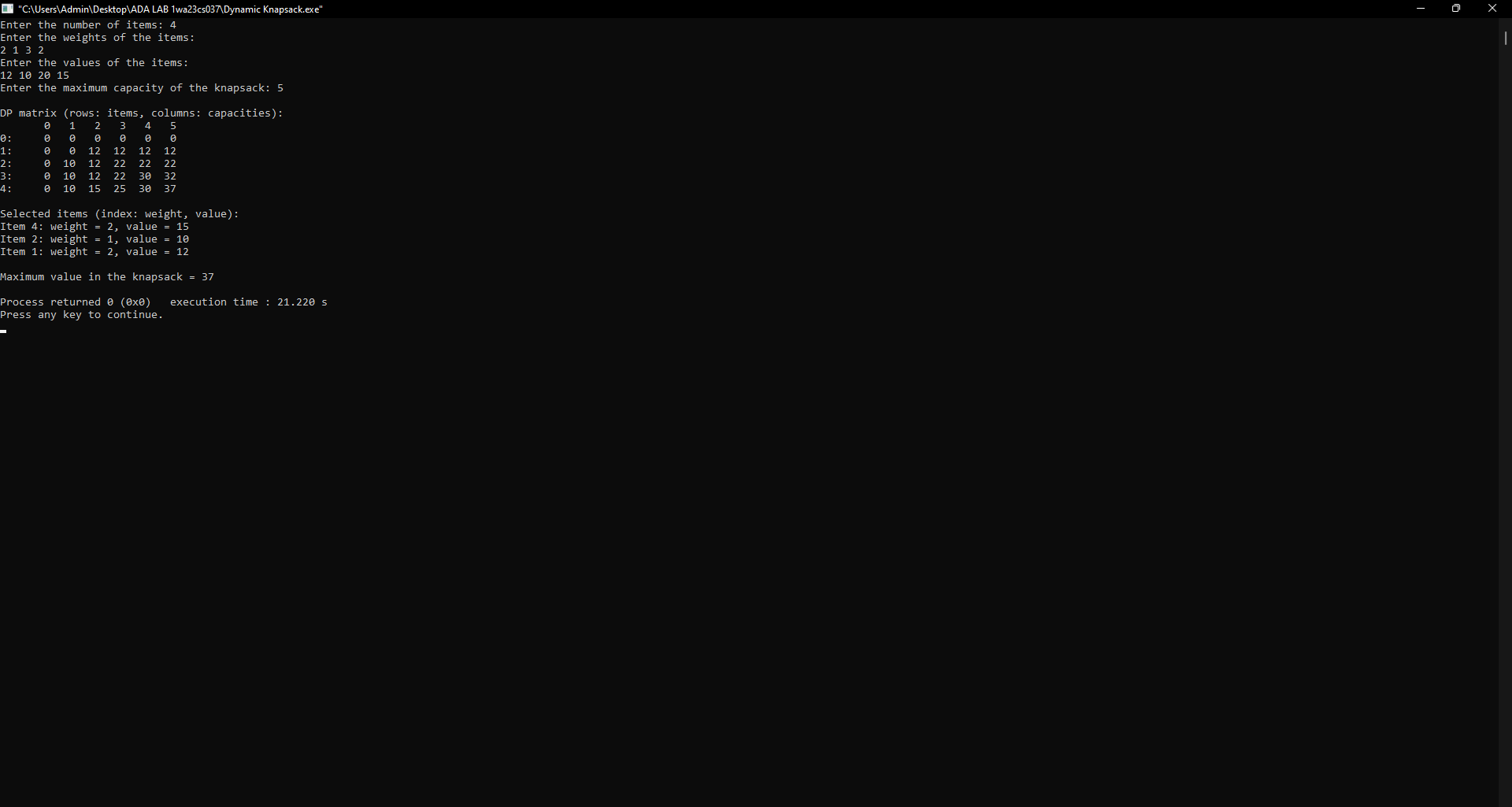
scanf("%d", &W);

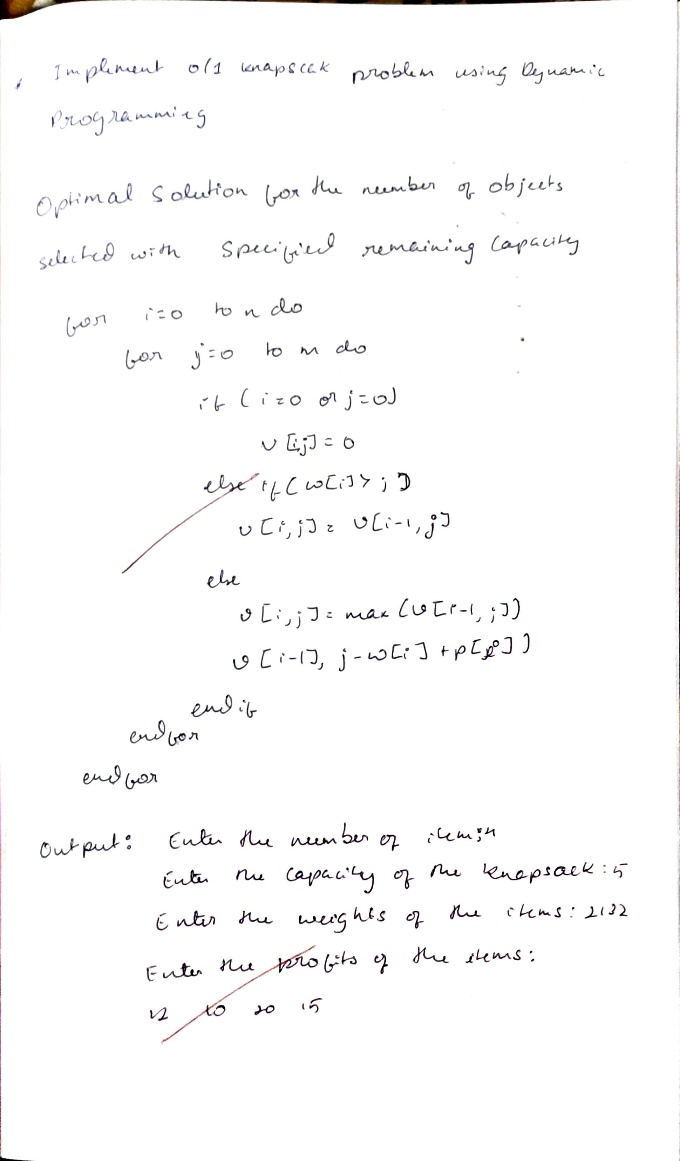
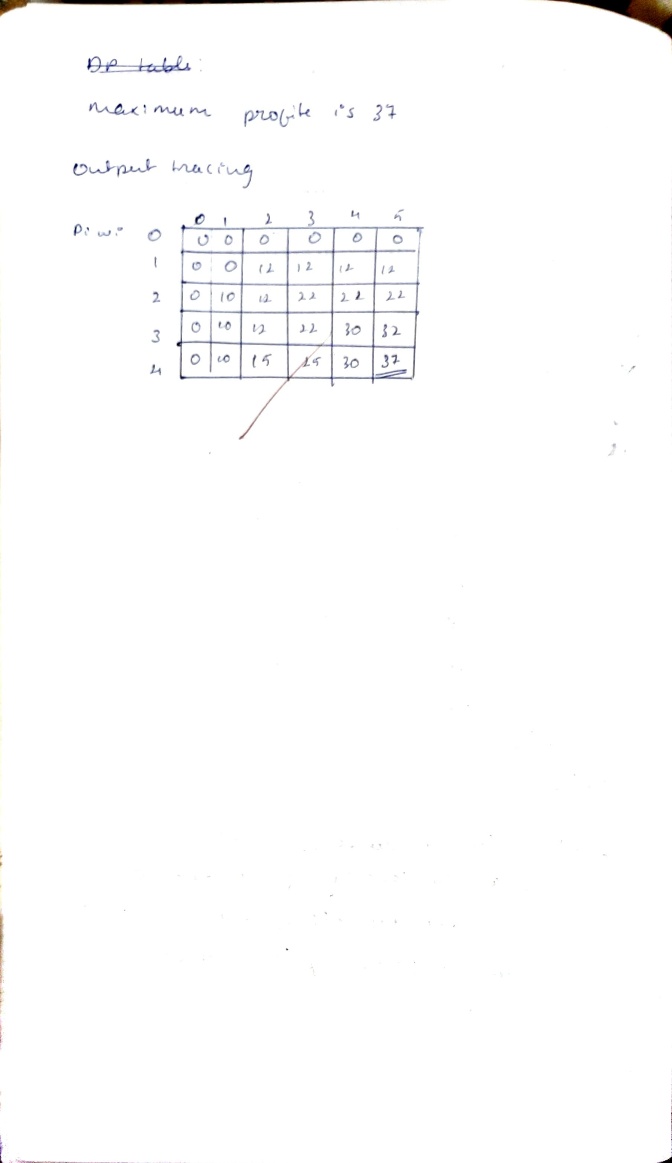
int max\_value = knapsack(W, wt, val, n);

printf("\nMaximum value in the knapsack = %d\n", max\_value);

return 0;

}

**OUTPUT:**



**Lab program 6:**

Implement All Pair Shortest paths problem using Floyd’s algorithm

**Code:**

#include <stdio.h>

#define INF 9999

#define MAX 10

int main() {

int V, i, j, k;

int graph[MAX][MAX], dist[MAX][MAX];

printf("Enter number of vertices: ");

scanf("%d", &V);

printf("Enter adjacency matrix (use %d for INF):\n", INF);

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

for (j = 0; j < V; j++)

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

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

for (j = 0; j < V; j++)

dist[i][j] = graph[i][j];

for (k = 0; k < V; k++)

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

for (j = 0; j < V; j++)

if (dist[i][k] + dist[k][j] < dist[i][j])

dist[i][j] = dist[i][k] + dist[k][j];

printf("Shortest distances:\n");

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

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

if (dist[i][j] == INF)

printf("INF ");

else

printf("%3d ", dist[i][j]);

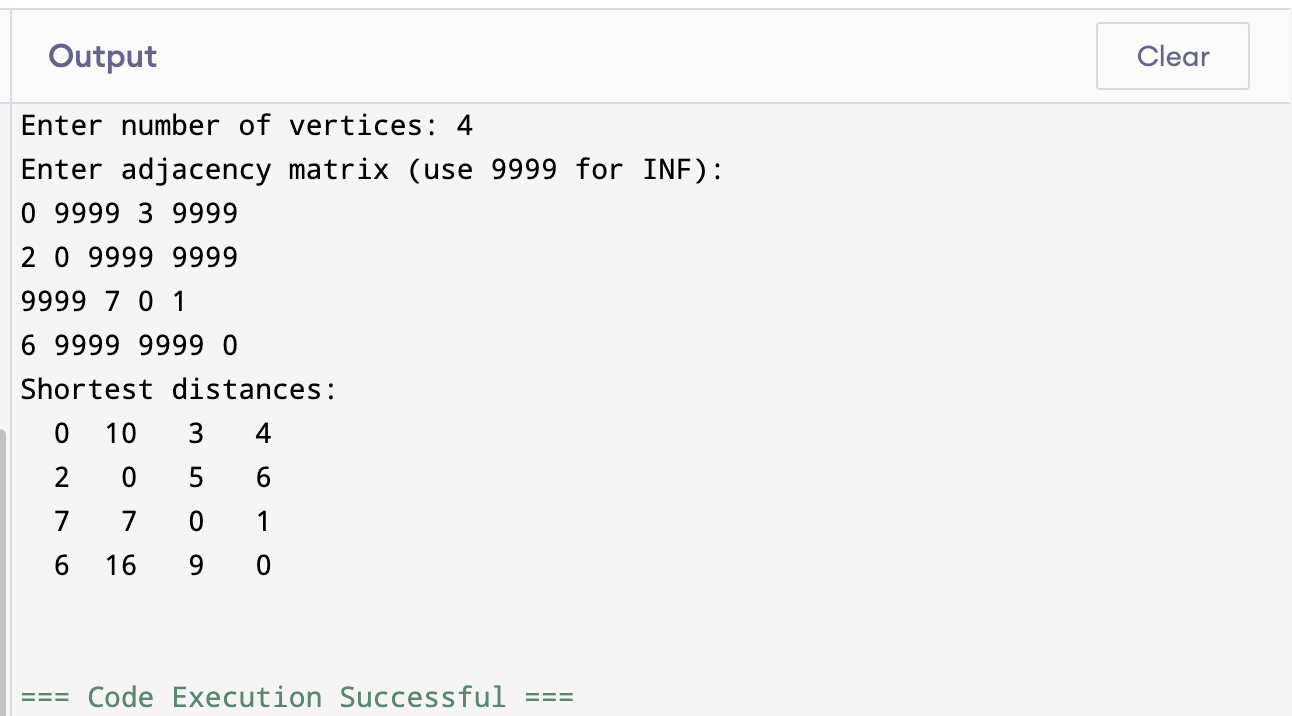
}

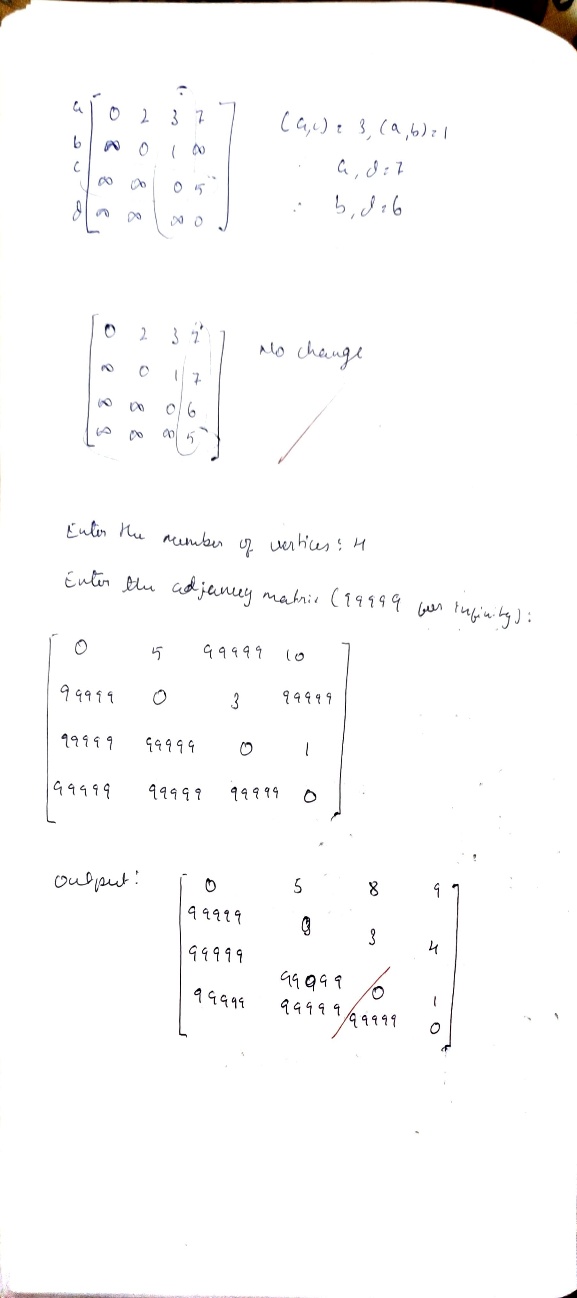
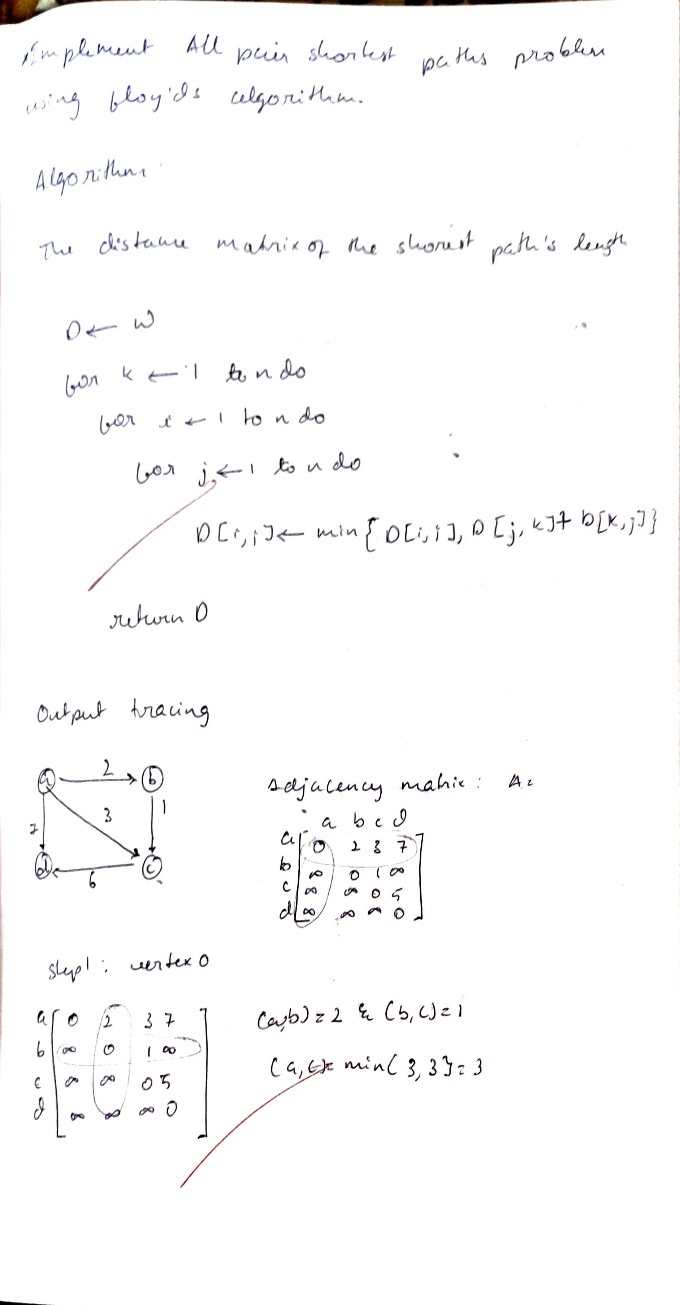
printf("\n");

}

return 0;

}

**OUTPUT:**



**Lab program 7:**

Implement Fractional Knapsack using Greedy technique.

**Code:**

**Fractional Knapsack**

#include <stdio.h>

#define MAX 10

struct Item {

int index;

int value;

int weight;

float ratio;

};

void sortItems(struct Item items[], int n) {

struct Item temp;

for (int i = 0; i < n - 1; i++) {

for (int j = i + 1; j < n; j++) {

if (items[j].ratio > items[i].ratio) {

temp = items[i];

items[i] = items[j];

items[j] = temp;

}

}

}

}

int main() {

struct Item items[MAX];

int n, capacity;

printf("Enter the number of items: ");

scanf("%d", &n);

printf("Enter knapsack capacity: ");

scanf("%d", &capacity);

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

printf("Enter value and weight for item %d: ", i + 1);

scanf("%d %d", &items[i].value, &items[i].weight);

items[i].ratio = (float)items[i].value / items[i].weight;

items[i].index = i + 1;

}

sortItems(items, n);

float totalValue = 0.0;

int remaining = capacity;

printf("\nSelected weights:\n");

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

if (items[i].weight <= remaining) {

remaining -= items[i].weight;

totalValue += items[i].value;

printf("Item %d: %d (full)\n", items[i].index, items[i].weight);

} else {

float fraction = (float)remaining / items[i].weight;

totalValue += items[i].value \* fraction;

printf("Item %d: %.2f (fractional)\n", items[i].index, (float)remaining);

remaining = 0;

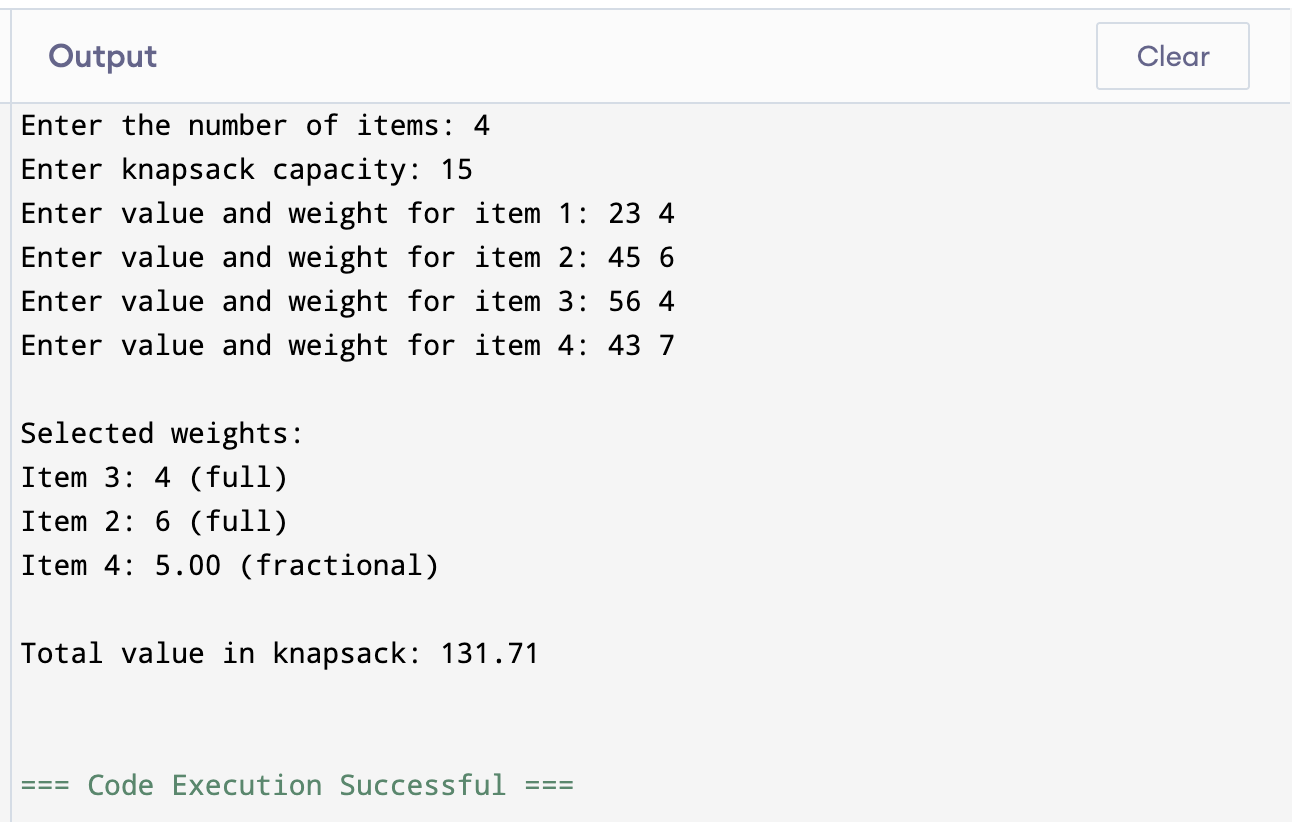
}

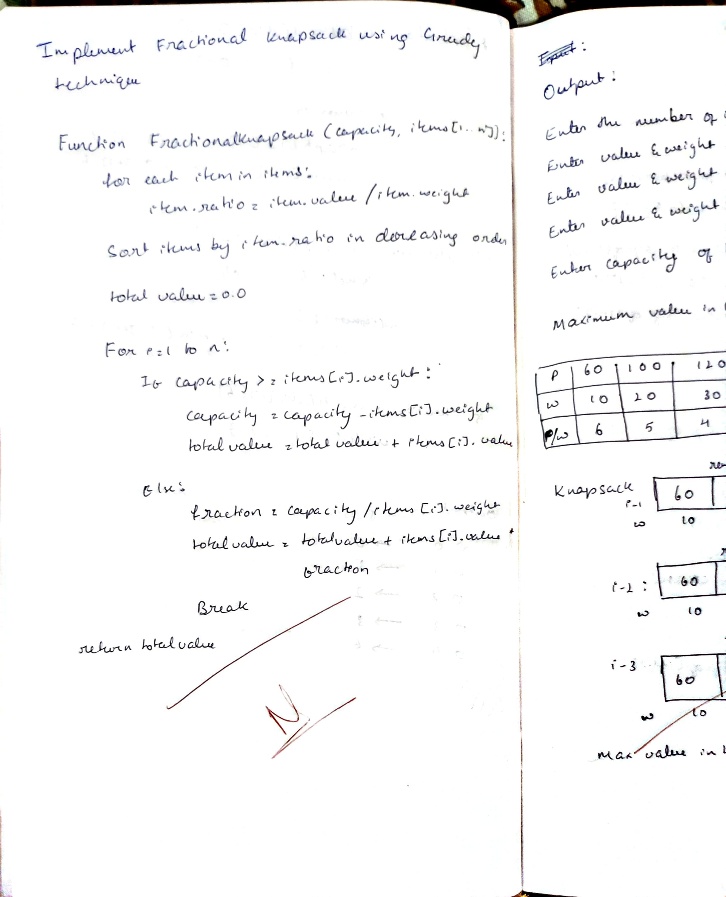
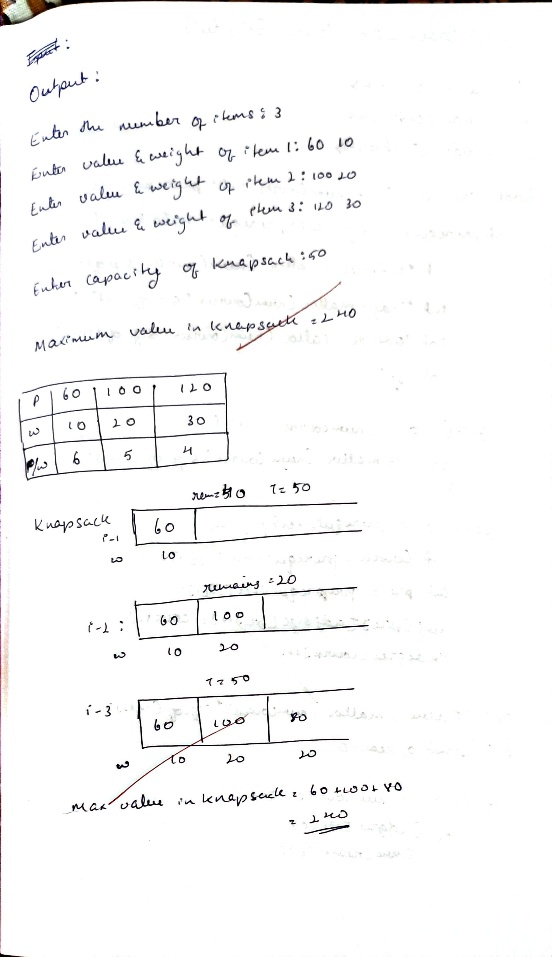
}

printf("\nTotal value in knapsack: %.2f\n", totalValue);

return 0;

}

**OUTPUT:**

**Lab program 8:**

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 V 100

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

int min = INT\_MAX, min\_index;

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

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

min = dist[v], min\_index = v;

return min\_index;

}

void printSolution(int dist[], int n, int src) {

printf("Vertex\t\tDistance from Source %d\n", src);

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

printf("%d -> %d\t\t%d\n", src, i, dist[i]);

}

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

int dist[V];

int visited[V];

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

dist[i] = INT\_MAX, visited[i] = 0;

dist[src] = 0;

for (int count = 0; count < n - 1; count++) {

int u = minDistance(dist, visited, n);

visited[u] = 1;

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

if (!visited[v] && graph[u][v] && dist[u] != INT\_MAX &&

dist[u] + graph[u][v] < dist[v])

dist[v] = dist[u] + graph[u][v];

}

printSolution(dist, n, src);

}

int main() {

int n;

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

scanf("%d", &n);

int graph[V][V];

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 src;

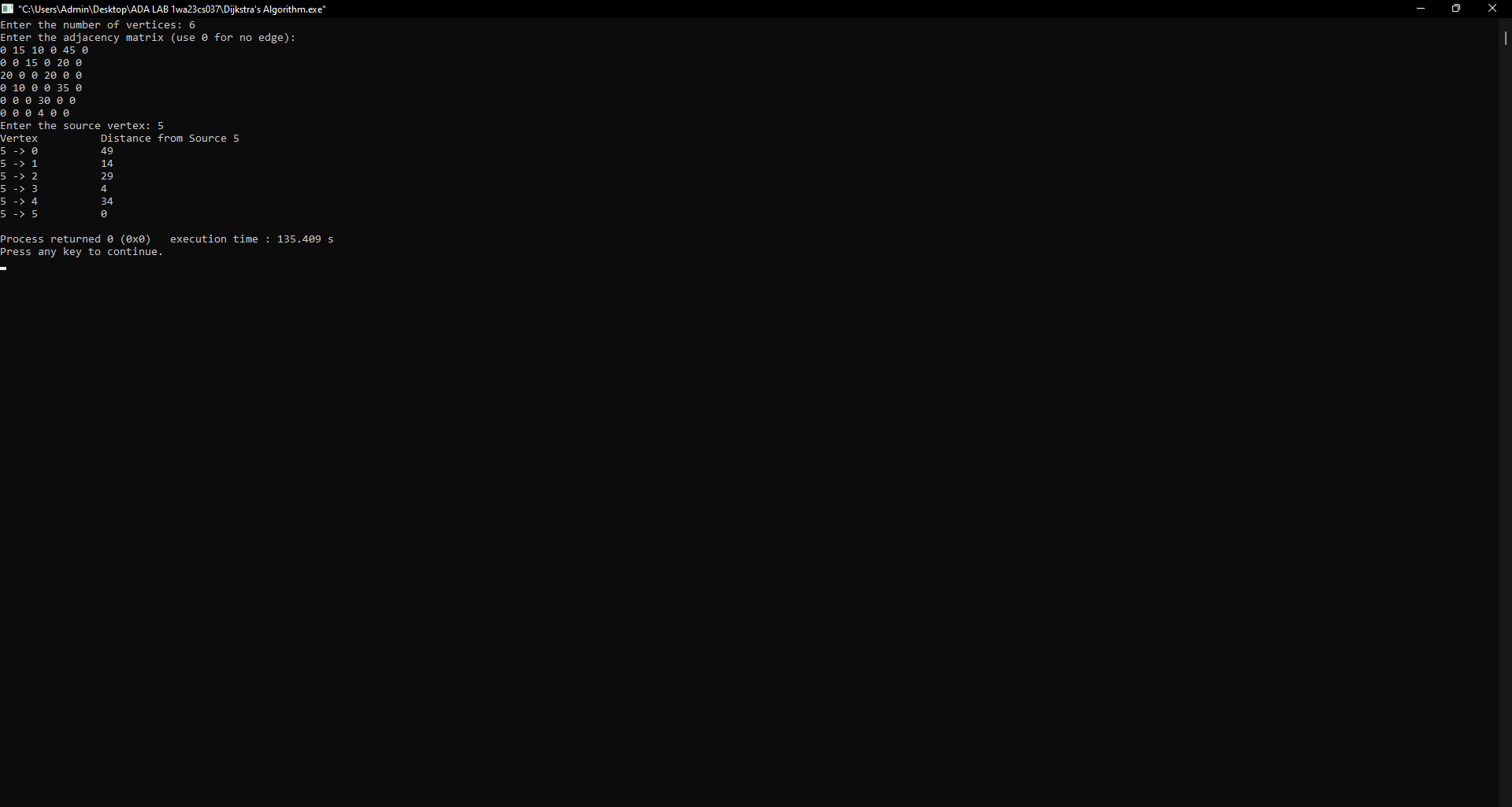
printf("Enter the source vertex: ");

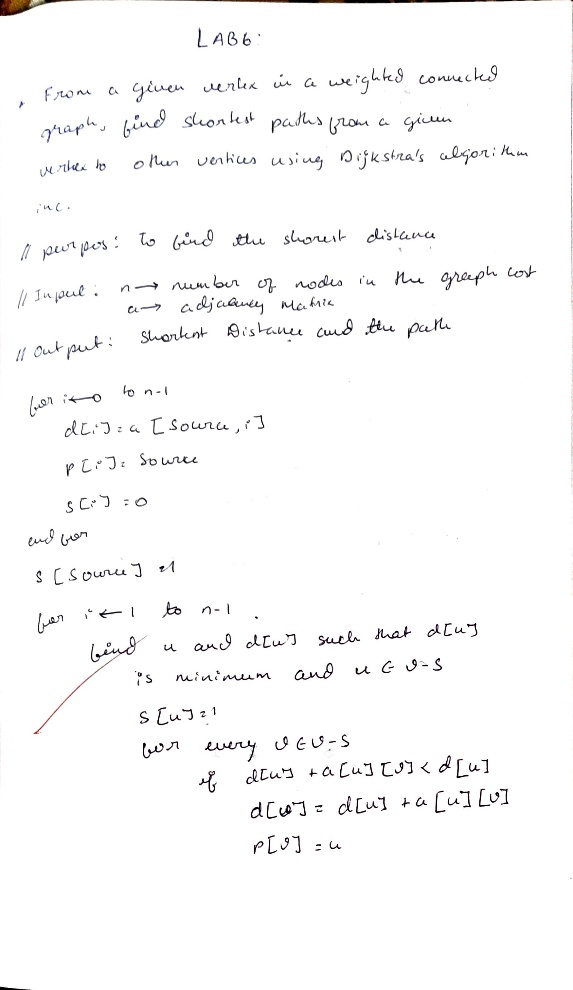
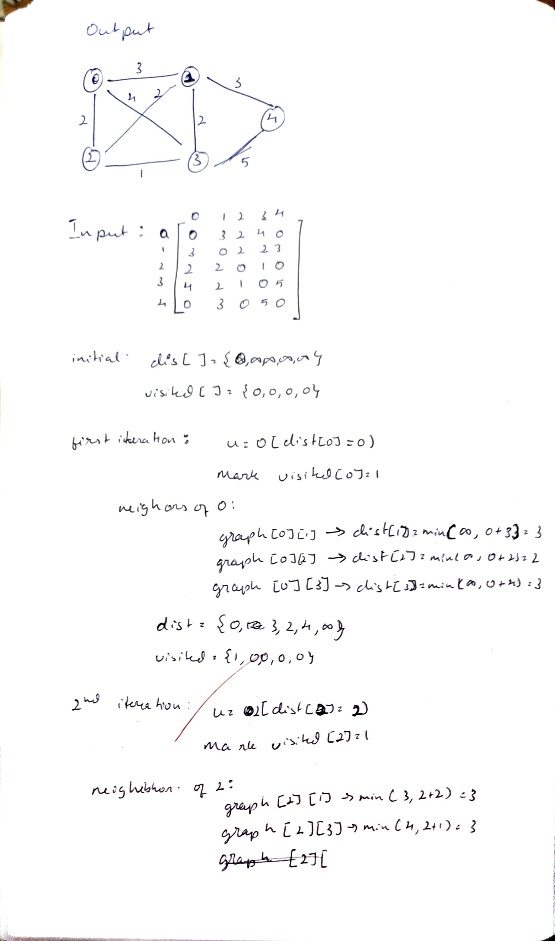
scanf("%d", &src);

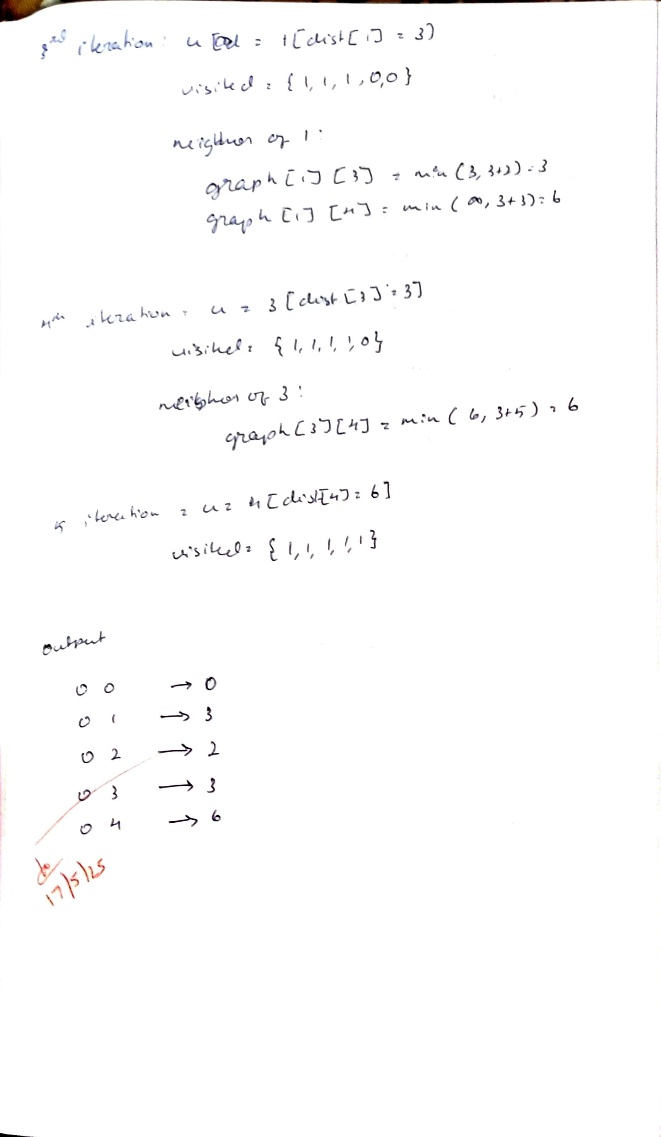
dijkstra(graph, src, n);

return 0;

} **OUTPUT:**

****



**Lab program 9:**

Implement “N-Queens Problem” using Backtracking.

**Code:**

#include <stdio.h>

#include <stdlib.h>

#define MAX 10

int board[MAX], n;

int isSafe(int row, int col) {

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

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

return 0;

}

return 1;

}

void printSolution() {

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

for (int j = 1; j <= n; j++) {

if (board[i] == j)

printf("Q ");

else

printf(". ");

}

printf("\n");

}

printf("\n");

}

void solve(int row) {

for (int col = 1; col <= n; col++) {

if (isSafe(row, col)) {

board[row] = col;

if (row == n)

printSolution();

else

solve(row + 1);

}

}

}

int main() {

printf("Enter number of queens: ");

scanf("%d", &n);

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

printf("Please enter a valid number between 1 and %d.\n", MAX);

return 1;

}

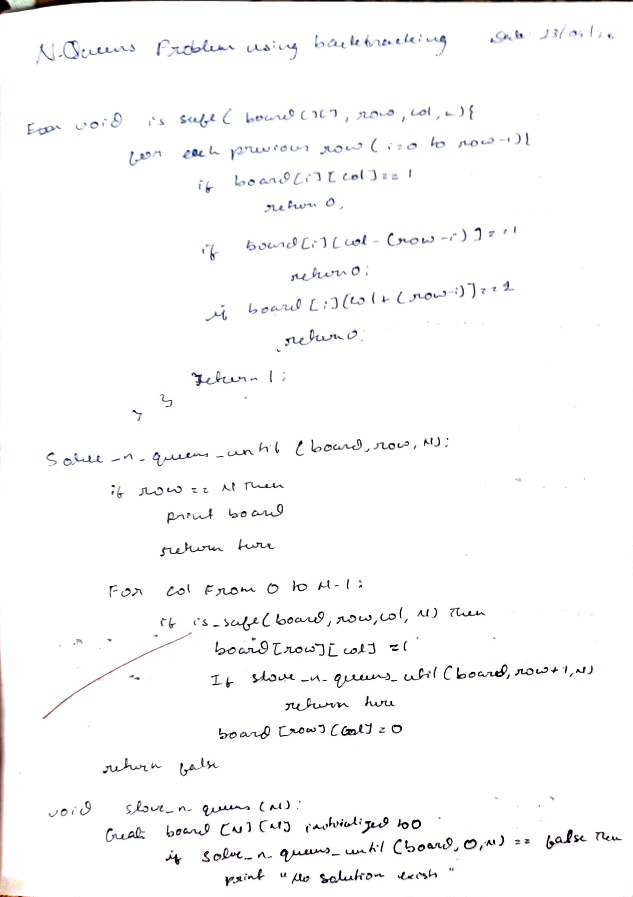
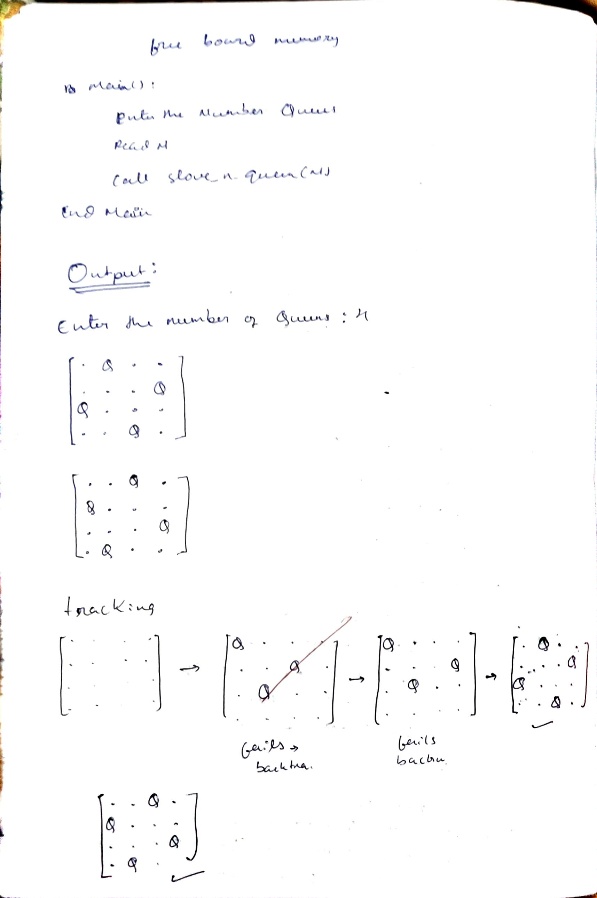
solve(1);

return 0;

}

**OUTPUT:**

****

**Lab program 11:**

Implement Johnson Trotter algorithm to generate permutations.

**Code:**

#include <stdio.h>

#include <stdlib.h>

#define LEFT -1

#define RIGHT 1

typedef struct {

int value;

int dir;

} 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;

int mobileIndex = -1;

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

int next = i + perm[i].dir;

if (next >= 0 && next < n && perm[i].value > perm[next].value) {

if (perm[i].value > mobile) {

mobile = perm[i].value;

mobileIndex = i;

}

}

}

return mobileIndex;

}

void generatePermutations(int n) {

Element \*perm = (Element \*)malloc(n \* sizeof(Element));

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

perm[i].value = i + 1;

perm[i].dir = LEFT;

}

printPermutation(perm, n);

while (1) {

int mobileIndex = getMobile(perm, n);

if (mobileIndex == -1)

break;

int next = mobileIndex + perm[mobileIndex].dir;

Element temp = perm[mobileIndex];

perm[mobileIndex] = perm[next];

perm[next] = temp;

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

if (perm[i].value > temp.value)

perm[i].dir = -perm[i].dir;

printPermutation(perm, n);

}

free(perm);

}

int main() {

int n;

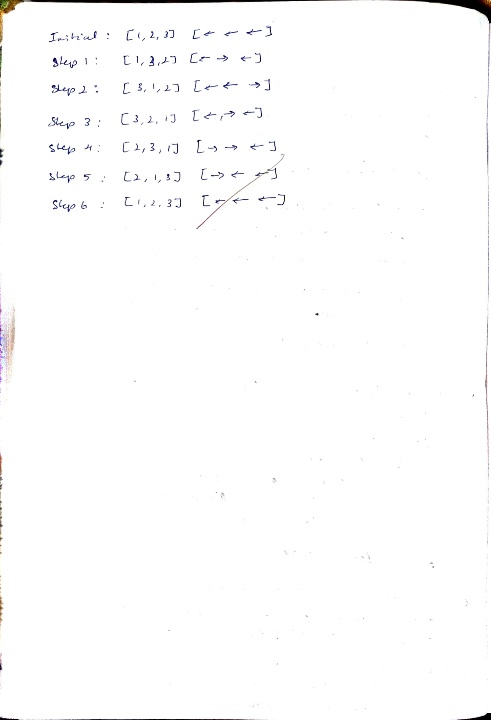
printf("Enter number of elements to permute: ");

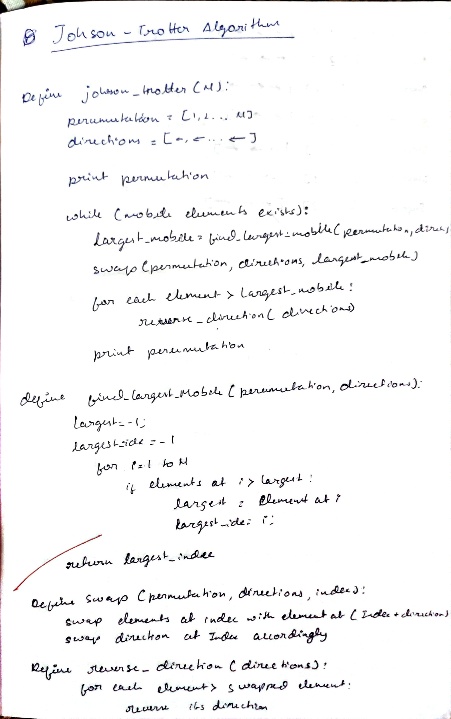
scanf("%d", &n);

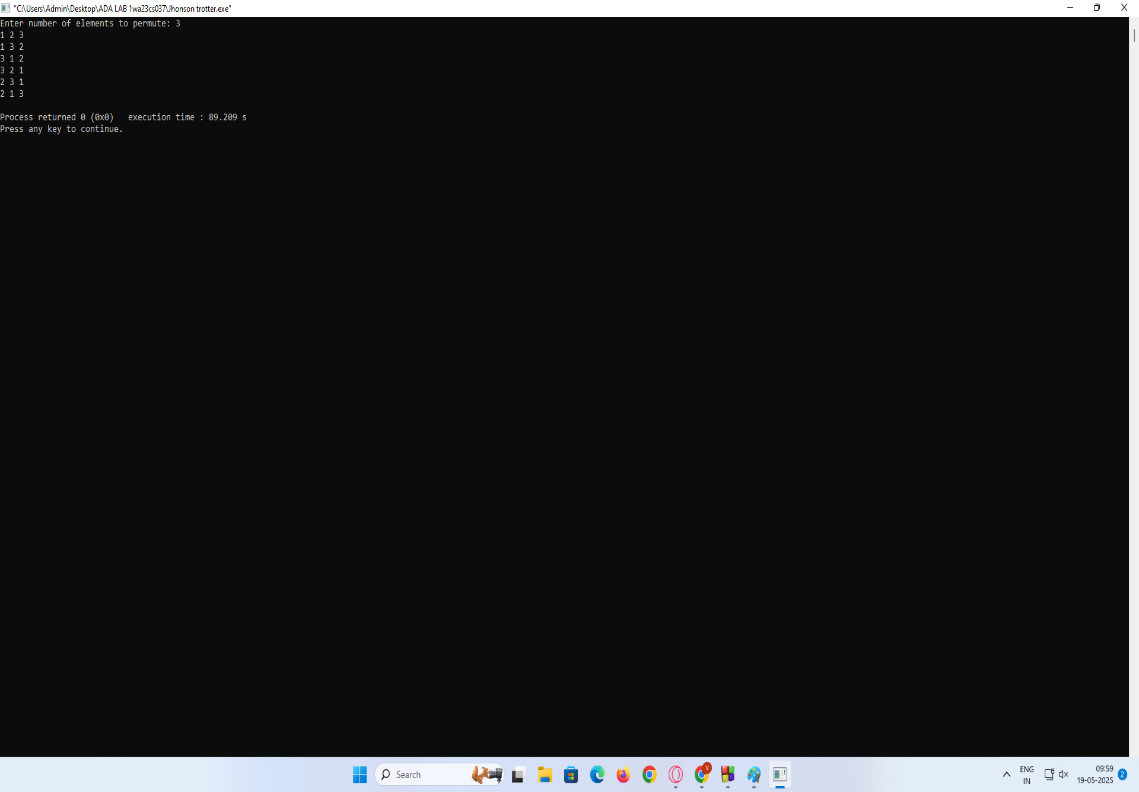
generatePermutations(n);

return 0;

}



**OUTPUT:**

****

**Lab program 12:**

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

**Code**

#include <stdio.h>

#include <stdlib.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);

}

}

void printArray(int arr[], int n) {

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

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

printf("\n");

}

int main() {

int n;

printf("Enter number of elements: ");

scanf("%d", &n);

int \*arr = (int \*)malloc(n \* sizeof(int));

printf("Enter %d elements:\n", n);

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

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

clock\_t start = clock();

heapSort(arr, n);

clock\_t end = clock();

double time\_taken = ((double)(end - start)) / CLOCKS\_PER\_SEC;

printf("Sorted array:\n");

printArray(arr, n);

printf("Time taken for Heap Sort: %.6f seconds\n", time\_taken);

free(arr);

return 0;

}

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;

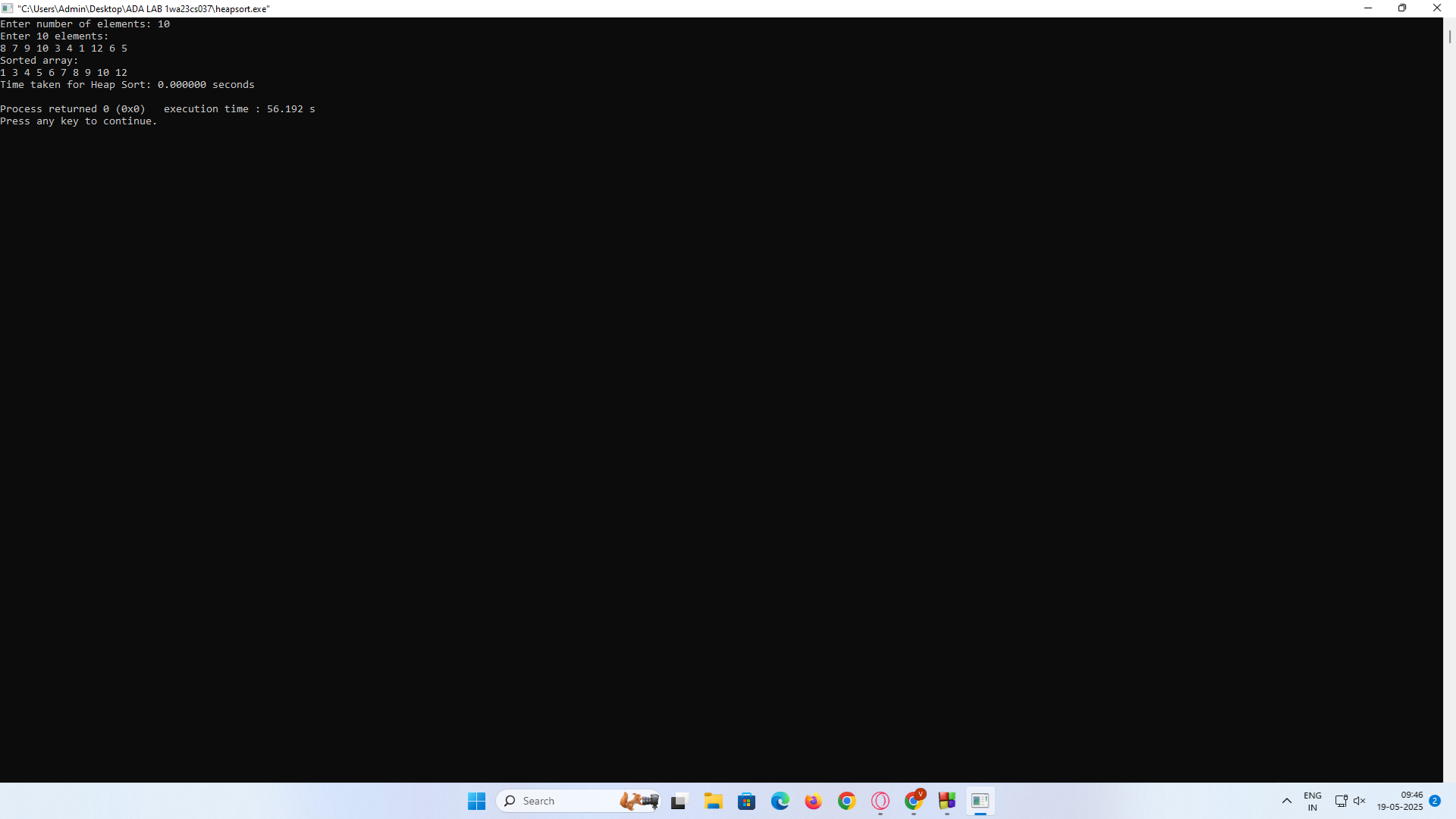
for (int i = 1; i < fact(n); i++) printOnePerm(a, dir, n);

}

int main() { int n = 4;

printPermutation(n); return 0;

}

**OUTPUT:**

