LP2 (AI) Exp No.3

1. Problem Statement: - Implement Selection Sort Algorithm.

def selection\_sort(arr):

    n = len(arr)

    for i in range(n):

        min\_index = i

        for j in range(i+1, n):

            if arr[j] < arr[min\_index]:

                min\_index = j

        arr[i], arr[min\_index] = arr[min\_index], arr[i]

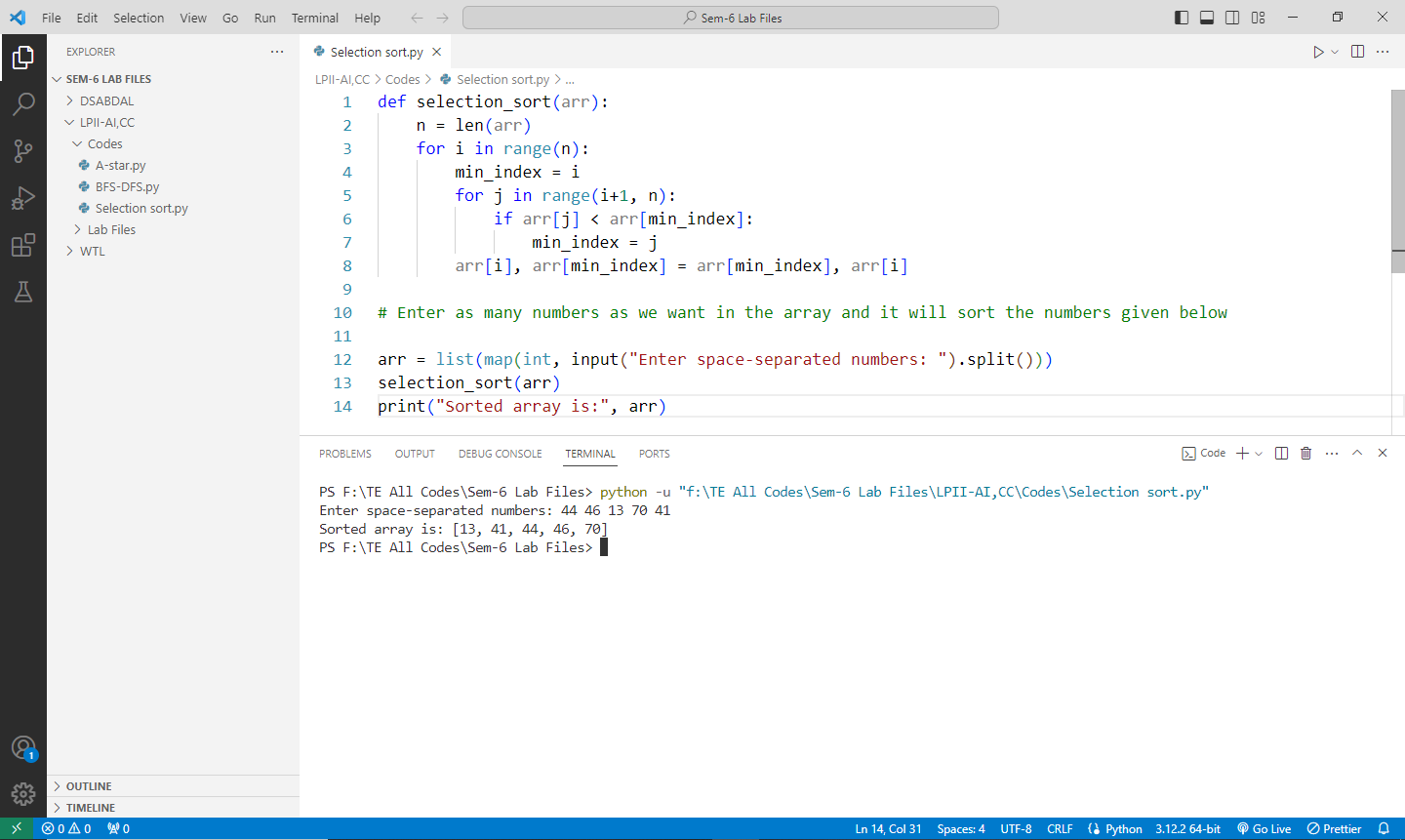
# Enter as many numbers as we want in the array and it will sort the numbers given below

arr = list(map(int, input("Enter space-separated numbers: ").split()))

selection\_sort(arr)

print("Sorted array is:", arr)

OUTPUT:-



1. Problem Statement: - Implement Greedy search algorithm for Dijkstra's Minimal Spanning Tree Algorithm.

import sys

class Graph:

    def \_\_init\_\_(self, vertices):

        self.V = vertices

        self.graph = [[0 for column in range(vertices)] for row in range(vertices)]

    def min\_distance(self, dist, spt\_set):

        min\_dist = sys.maxsize

        min\_index = 0

        for v in range(self.V):

            if dist[v] < min\_dist and spt\_set[v] == False:

                min\_dist = dist[v]

                min\_index = v

        return min\_index

    def dijkstra(self, src):

        dist = [sys.maxsize] \* self.V

        dist[src] = 0

        spt\_set = [False] \* self.V

        for cout in range(self.V):

            u = self.min\_distance(dist, spt\_set)

            spt\_set[u] = True

            for v in range(self.V):

                if self.graph[u][v] > 0 and spt\_set[v] == False and dist[v] > dist[u] + self.graph[u][v]:

                    dist[v] = dist[u] + self.graph[u][v]

        self.print\_solution(dist)

    def print\_solution(self, dist):

        print("Vertex \tDistance from Source")

        for node in range(self.V):

            print(node, "\t", dist[node])

V = int(input("Enter the number of vertices: "))

g = Graph(V)

print("Enter the adjacency matrix (space-separated entries, use 0 for no edge):")

for i in range(V):

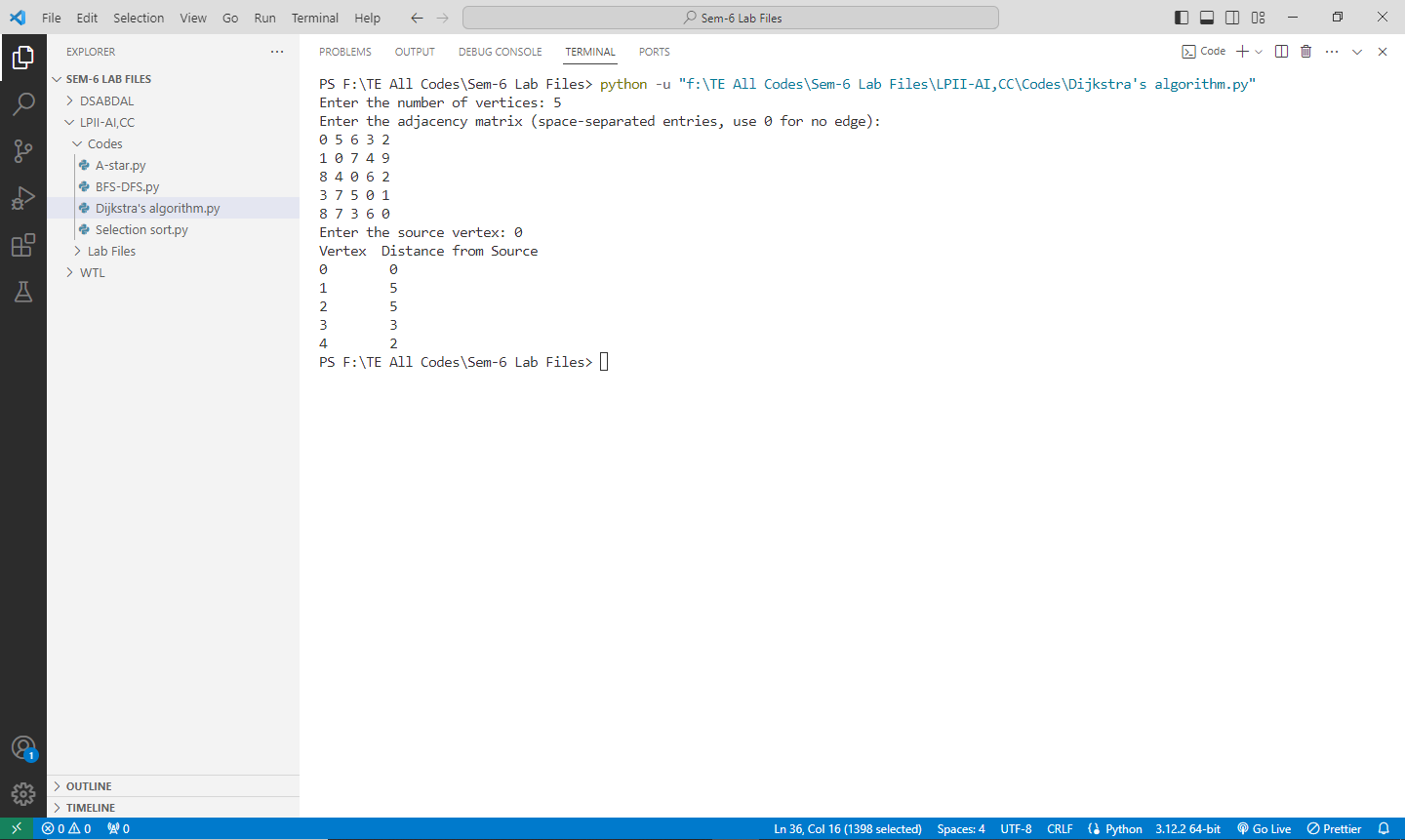
    row = list(map(int, input().split()))

    g.graph[i] = row

src = int(input("Enter the source vertex: "))

g.dijkstra(src)

OUTPUT:-



1. Problem Statement: - Implement Greedy search algorithm for Kruskal's Minimal Spanning Tree Algorithm.

class Graph:

    def \_\_init\_\_(self, vertices):

        self.V = vertices

        self.graph = []

    def add\_edge(self, u, v, w):

        self.graph.append([u, v, w])

    def find\_parent(self, parent, i):

        if parent[i] == i:

            return i

        return self.find\_parent(parent, parent[i])

    def union(self, parent, rank, x, y):

        x\_root = self.find\_parent(parent, x)

        y\_root = self.find\_parent(parent, y)

        if rank[x\_root] < rank[y\_root]:

            parent[x\_root] = y\_root

        elif rank[x\_root] > rank[y\_root]:

            parent[y\_root] = x\_root

        else:

            parent[y\_root] = x\_root

            rank[x\_root] += 1

    def kruskal\_mst(self):

        result = []

        i, e = 0, 0

        self.graph = sorted(self.graph, key=lambda item: item[2])

        parent = []

        rank = []

        for node in range(self.V):

            parent.append(node)

            rank.append(0)

        while e < self.V - 1:

            u, v, w = self.graph[i]

            i += 1

            x = self.find\_parent(parent, u)

            y = self.find\_parent(parent, v)

            if x != y:

                e += 1

                result.append([u, v, w])

                self.union(parent, rank, x, y)

        print("Edges in the constructed MST:")

        for u, v, weight in result:

            print("%d -- %d == %d" % (u + 1, v + 1, weight))

V = int(input("Enter the number of vertices: "))

E = int(input("Enter the number of edges: "))

g = Graph(V)

print("Enter the edges as 'source destination weight':")

for i in range(E):

    u, v, w = map(int, input().split())

    g.add\_edge(u - 1, v - 1, w)

g.kruskal\_mst()

OUTPUT:-

