Name: Ashutosh Raj Gupta

Roll no: TECO2223A045

Batch: A3

Assignment no: 3

Implement Greedy search algorithm for any of the following application:

1. Selection Sort
2. II. Minimum Spanning Tree
3. III. Single-Source Shortest Path Problem
4. IV. Job Scheduling Problem
5. V. Prim's Minimal Spanning Tree Algorithm
6. VI. Kruskal's Minimal Spanning Tree Algorithm
7. VII. Dijkstra's Minimal Spanning Tree Algorithm

Code:

1.Selection sort:

def Selection\_Sort(array):

    for i in range(0, len(array) - 1):

        smallest = i

        for j in range(i + 1, len(array)):

            if array[j] < array[smallest]:

                smallest = j

        array[i], array[smallest] = array[smallest], array[i]

array = input('Enter the list of numbers: ').split()

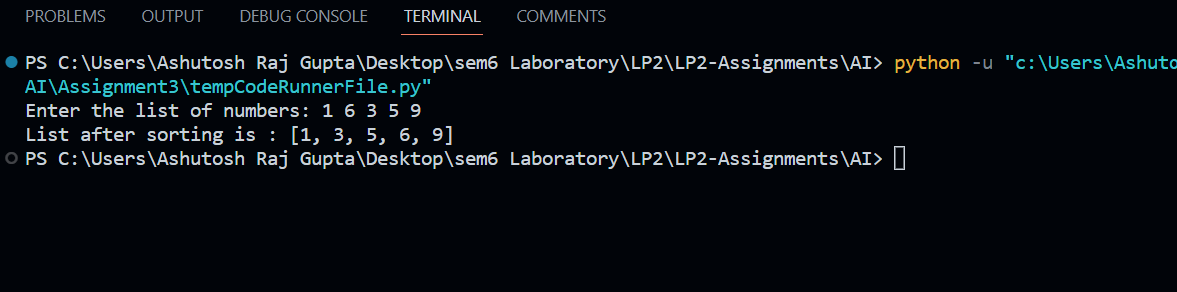
array = [int(x) for x in array]

Selection\_Sort(array)

print('List after sorting is : ', end='')

print(array)

output:



Code 2: Job scheduling

# Jobs, Profit, Slot

profit = [15,27,10,100, 150]

jobs = ["j1", "j2", "j3", "j4", "j5"]

deadline = [2,3,3,3,4]

profitNJobs = list(zip(profit,jobs,deadline))

profitNJobs = sorted(profitNJobs, key = lambda x: x[0], reverse = True)

slot = []

for \_ in range(len(jobs)):

    slot.append(0)

profit = 0

ans = []

for i in range(len(jobs)):

    ans.append('null')

for i in range(len(jobs)):

        job = profitNJobs[i]

        #check if slot is occupied

        for j in range(job[2], 0, -1):

            if slot[j] == 0:

                ans[j] = job[1]

                profit += job[0]

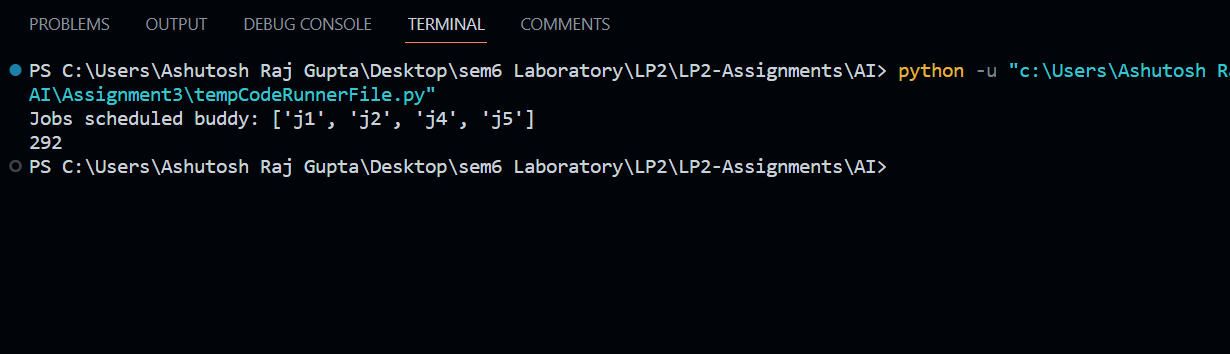
                slot[j] = 1

                break

print("Jobs scheduled buddy:",ans[1:])

print(profit)

output:



Code 3: Kruskals

# Kruskal's algorithm in Python

class Graph:

    def \_\_init\_\_(self, vertices):

        self.V = vertices

        self.graph = []

    def add\_edge(self, u, v, w):

        self.graph.append([u, v, w])

    # Search function

    def find(self, parent, i):

        if parent[i] == i:

            return i

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

    def apply\_union(self, parent, rank, x, y):

        xroot = self.find(parent, x)

        yroot = self.find(parent, y)

        if rank[xroot] < rank[yroot]:

            parent[xroot] = yroot

        elif rank[xroot] > rank[yroot]:

            parent[yroot] = xroot

        else:

            parent[yroot] = xroot

            rank[xroot] += 1

    #  Applying Kruskal algorithm

    def kruskal\_algo(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 = i + 1

            x = self.find(parent, u)

            y = self.find(parent, v)

            if x != y:

                e = e + 1

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

                self.apply\_union(parent, rank, x, y)

        for u, v, weight in result:

            print("%d - %d: %d" % (u, v, weight))

g = Graph(6)

g.add\_edge(0, 1, 4)

g.add\_edge(0, 2, 4)

g.add\_edge(1, 2, 2)

g.add\_edge(1, 0, 4)

g.add\_edge(2, 0, 4)

g.add\_edge(2, 1, 2)

g.add\_edge(2, 3, 3)

g.add\_edge(2, 5, 2)

g.add\_edge(2, 4, 4)

g.add\_edge(3, 2, 3)

g.add\_edge(3, 4, 3)

g.add\_edge(4, 2, 4)

g.add\_edge(4, 3, 3)

g.add\_edge(5, 2, 2)

g.add\_edge(5, 4, 3)

g.kruskal\_algo()

output-

