Ece Karasu
29467

a)

$$\delta(s,v) = \begin{cases} 0 & \text{if } s = v \\ \text{maxsize} & \text{if } v \text{ does not exist} \\ \min[\, \delta(s,u) + w(u,v) \,] & \text{otherwise} \end{cases}$$

where:

- $\delta(s,v)$ is the quickest itinerary from s to v
- u is each of the neighbors of v
- $w(u,v)$ is the direct itinerary from u to v

b)

```
for each v in the twod_list
    set goal_idx = v

    if δ(s,v) is not in twod_list_memo
        if dep_idx is equal to goal_idx
            δ(s,v) = 0
            add it to twod_list_memo

        else if twod_list(dep_idx, goal_idx) > 0
            initialize neighbors_list for goal_idx with maxsize number
            add twod_list(dep_idx, goal_idx) to neighbors_list
            for each neighbor u of goal_idx
                find δ(s,u) + w(u,v)
                add it to neighbors_list
            δ(s,v) = minimum of neighbors_list

        else if twod_list(dep_idx, goal_idx) = -1
            initialize neighbors_list for goal_idx with maxsize number
            for each neighbor u of goal_idx
                find δ(s,u) + w(u,v)
                add it to neighbors_list
            δ(s,v) = minimum of neighbors_list

        else
            δ(s,v) = maxsize - 1
            add it to twod_list_memo

    else
        δ(s,v) = twod_list_memo(dep_idx, goal_idx)
```

Ece Karasu
29467

c) Let's say there are n cities in the set of cities. So, we assume there are 2n stations.

Time complexity:
1- For the function call (for every station) → O(2n) = O(n)
2- In the function (it may run the while loop in the worst case) → O(2n) = O(n)
3- 1 and 2 brings O(n)*O(n) = O(n^2)

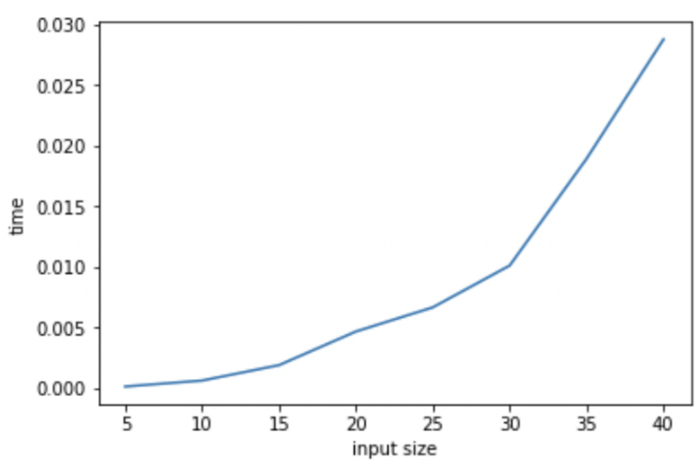
(I ignored the time required to initialize the memoization matrix and print operations)

⇒ In total, the time complexity of my algorithm is O(n^2)

Space complexity:
1- For creating the itineraries matrix (twod_list) → O((2n)*(2n)) = O(n^2)
2- For creating the memoization matrix → O((2n)*(2n)) = O(n^2)
3- For values list → O(2n) = O(n)
4- For visited list → O(2n) (in the worst case) = O(n)
5- For neighbors list → O(2n) (in the worst case) = O(n)

⇒ In total, the space complexity of my algorithm is O(n^2)

d)



For the graph, I used a random matrix generator piece of code and tried different sizes of input. According to the graph, my algorithm runs in quadratic time which matches the result that I have found in part c. So, the graph worked as expected.