CAS CS 330. Problem Set 3
Name: Kebi Hong

Problem 1.

In order to find the minimum subset B $\subseteq$ S, the optimal choice would be that having a interval, which is big enough to include all other intervals. for example, there are several intervals :(1,2), (3,4), (5,6),(8,9). if there is an interval (1,10), then this one interval would be the minimum subset B we are looking for.

Algorithm:

First, we need to label all the intervals and sort the list by labels in decreasing order. Then we pick out the one with the most overlaps, and then delete other intervals in the list, who are overlapped with this selected interval. After this, we pick the next one in the list who has largest label, and repeat the algorithm until we reach the end of the list.

1. use partitioning.py code to get all the intervals labeled
2. for interval in intervals:
    solutionList += interval
    for i in intervals:
        if i != interval:
            if i is intersected with interval:
                then delete i in the list
3. After this double for loop, we will get our answers in the solutionList

Proof:

Case: we have intervals: A=(1,7), B = (8,10), C = (11,12), D = (4,9), E =(6,9)

The algorithm will first use partitioning code to label all the intervals and we get the following:

labelA = 3, labelB = 3, labelC = 1, labelD = 4, labelE = 4

After sort the list by their labels, we get [D,E,A,B,C]. At this time, D and E both have the same amount the labels, but it doesn't matter which one we pick.

The algorithm then picks the first interval in the list, D, and adds it to the new list S = [D], then deletes D from the old list and deletes all intervals that intersect with D. Therefore the old list would be [C]. Next, the algorithm picks C and puts it in S. Because there is nothing in the old list, the algorithm stops.

In this way, we get the solution:[D,C], which is correct.

Analyze:

In order to get all intervals labeled, we can use the partitioning.py program from the last assignment, and this will give us approximately theta(nlgn). Then we will sort the list by their labels, and this will take about theta(n). After this, we will go through every interval in the list, and for everyone of them, we will compare them with the rest of the list, and this will cost theta(n^2). So the total time will be theta(n^2)

Problem 2.

Algorithm (Dijikstra's):

1. Assign to every node a tentative distance value: set the initial node (start point) to be 0 and all other nodes to be ∞.
2. Create two sets : visited set S and unvisited set U. S only contains the initial node, and U contains all the other node. Set the initial node to be the current node. For every element u in U, if u is the neighbor of the initial node, set its value to be the distance, otherwise, set its value to be ∞.
3. For every element in U, find the element k, which is closest to the current node. set k to be the current node. Add k to S and delete k from U
4. For the current node, compute the distance between the current point and the connecting neighbors in U plus the cost of entering the nerghbor. Compare the newly calculated distance to the current assigned value and assign the smaller one. for example, if the current node *A* is marked with a distance of 6, and the edge connecting it with a neighbor *B* has length 2, and the cost of the vertex B is 3. Then the distance to *B* (through *A*) will be 6 + 2 + 3= 11. If B was previously marked with a distance greater than 11 then change it to 11. Otherwise, keep the current value.
5. Repeat steps 3 and 4 until S contains all the nodes.

Proof:

If we make the cost of edge equals as following, it will just be the same case that Dijikstra is talking about: cost-of-edge = cost-of-edge + cost-of-entering-vertices

Make up case: input a matrix[6][6]. If two nodes are neighbors,the value of each element in the matrix equals to the cost of edge plus the cost of entering to a vertex, otherwise, the value will be 999

	0	1	2	3	4	5
0	999	5	7	999	999	999
1	5	999	999	20	15	999
2	7	999	999	3	999	10
3	999	20	3	999	6	999
4	999	15	999	6	999	8
5	999	999	10	999	8	999

After applying the Dijikstra's algorithm, visited set S=[0], unvisited set U = [1,2,3,4,5]

1. We find that 0 -> 1 is the shortest. S = [0,1] U = [2,3,4,5]
2. 0 -> 2 = 7 is the shortest, S = [0,1,2] U = [3,4,5]
3. 0 -> 2 -> 3 = 10 is the shortest, S = [0,1,2,3], U = [4,5]
4. 0 - >2 -> 3 -> 4 = 16 is the shortest, S=[0,1,2,3,4], U = [5]
5. 0 ->2 -> 5 = 17 is the shortest, S = [0,1,2,3,4,5], U = []

In this way, we correctly update all the distances between vertices, and it is not worse than other algorithm, therefore, it is a good algorithm.

Analyze:

If there are n nodes in set U, and we need to visit all of them, which is theta(n). And for each time we visit, we first use a for-loop to find the best vertex (minimum path), this will take be theta(n^2). After that, we need to go through all the node and update the distance, this will take theta(n^2). Therefore, the total time would be theta(n^2)

Problem 3

Algorithm:

the algorithm is simple, we are going to drop off the executive that cost most and drop off everyone that is on the way.

first we need to define the class executives, and it should have three fields as following:

dist = the point (bus station) that executive is going to get off

cost = the amount of charge of every unit of time before the executive is dropped off

total = dist * cost

1. first load all the datas into a list U and create an empty list S.
2. set valuable previous = 0
3. for ex in U:

    sort everything in U by their total field, with decreasing order

    put ex in to list S # record how far the bus has traveled

    DeltaD = ex.dist - previous

    # then we are going to drop off executives that are going to the same direction and lives closer

        for ex2 in U that are not ex:

            if the field dist for every ex2 is like this previous <ex2 < ex:

                then add it to S and delete it from U

    #then we are going to updates the total field for everything left in U

        for ex3 in U that are not ex:

            if ex3 and ex have same sign (negative or positive):

                then ex3.total =ex3.cost * (ex3.dist - DeltaD)

            else (they have different sign):

                then ex3.total = ex3.cost* (ex3.dist + DeltaD)

        set previous = ex and delete it from U

4. After we add everything into S, we finish the algorithm, and then if we print out S by order, we will get the order we want to drop off executives.

Proof:

Because every time, we try to drop off executives that cost most as soon as possible, therefore, we will save a lot of money. This is no worse than other algorithms.

Analyze:

Because we used double for loop to updates total cost for every executive for every stop, this program will run under theta(n^2)