Question 1)

Algorithm Idea: We will assume that each rally is sorted from the lowest ranked ( 1 ) to the highest rank (n). Given n rallies and a finish time fi, we will have an integer t = 0, which will keep track of the total time and ti = 0, which keeps track of the time it took to complete rally i. For every i in 1 … n do rally i. Once rally i is complete, you add the time it took complete rally i (t) to ti. We need to add the previous ti to the new ti plus t since we want the total time every since Serena started this. Now, you compare t to fi. If t < fi, you continue on to the next i. However, if t > fi, this schedule will not work. You print out, Serena cannot follow this schedule, and you end it right there. Once we exit the loop normally, we can assume that we finished each rally i within fi times. This means we can use this schedule since it saves the world in time.

Algorithm Detail:

Assume that the rallies 1 … n is sorted by lowest rank ( 1 ) to highest rank ( n )

Given n rallies and a finish time fi

ti = 0 (total time for rally i to finish)

t = 0 (current time for rally i to finish)

Assume that rally 1 is always the starting rally.

For i in 1 … n do

Rally i

t = time it took to complete rally i

Once rally i is complete

ti = ti + t

t = 0 (resets our current time)

Compare ti and fi

If ti < fi

Continue to next i

Else if ti > fi

Print Serena cannot follow this schedule

Return 0

Print Serena can follow this schedule

Proof Idea: My algorithm does every condition that the question is asking for. For each rally i, it must be finished with fi. Also, my algorithm ensures that there is no overlapping for each rally and each rally is completed before another one starts. I also ensure that we always start on rally 1 and that it tells Serena if this schedule is correct or not. I will also sum up each part of my algorithm to show that it runs in 0(nlogn).

Proof Detail: My algorithm fulfills all the necessary conditions that the problem gives us. My algorithm checks that each rally i with a time of ti is completed within the given fi. It checks if ti is < or > fi. If ti < fi, that means it works and we can move on to then next rally. However, if ti > fi, that means she went over the time limit and that schedule doesn’t work and we stop the algorithm right there. This check keeps going for as long as ti < fi or when we finished the last rally and the last rally goes through right. The next condition, only one rally at a time. My algorithm only allows one rally to go through. It sends in 1 rally and waits until that rally is done. Once that rally is done, it does the check and if the check works, it goes on to the next rally. Next condition, we are starting with rally 1. In my algorithm, I am explicating stating that we are starting at rally 1. So, no matter how many time the schedule changes and the algorithm is ran, we always have to start with rally 1. Finally, deciding of the schedule is correct or not. Within the check if ti is greater or less than fi, we have it check that, if ti > fi, we have it print out that the schedule is wrong and we end the algorithm right there. Once all the rallies have been finished and the final rally check goes through, we print out, this schedule works.

Run time Analysis: I will prove that my algorithm runs in 0(nlogn). The first part of the algorithm, sorting the n rallies, take up to O(nlogn). Any type of sorting that we do is done in O(nlogn). Next part, the for loop. Since the loop itself is running from 1 to n, the loop is going to be in n times. So that’s O(n). Now, for everything inside the loop. Running the rally and keep track of the time will only be a constant time, which is O(1). Keeping track of the total time with addition is also in constant time, O(1). Using if else statement to check if ti is less than or greater than fi is only O(1) and printing out the statement when ti > fi is only O(1). So, you multiple all of those run times within the for loop, and you get O(n) \* O(1) \* O(1) \* O(1) and you get O(n). Now, at the end of the algorithm, you have another print statement, which only runs in O(1) times. Now, to find the overall run time, we add up everything together, which we get O(nlogn) + O(n) + O(1), which becomes O(nlogn + n + 1), which becomes O(nlogn) since 1 is constant and nlogn is bigger than n, so we take the one that’s bigger.

Source: Piazza posts, lecture notes 20 and 21.

Question 2)

Algorithm Idea: We are given n number of charging stations that are between Buffalo and Seattle. We can also assume that the route between each charging station are given in miles. We will assume that she will charge her Tesla to full at each charge station she stops at. Since we’re not sure if the list of charging stations are in order, we will sort out the charging stations from the closest charging station to Buffalo to the farthest one. We then put the list of charging stations into an array C. As long as C isn’t empty, we will have an integer m which will be initial 0. We will take the first three charging stations that are in the array and add them up together. We are adding three at a time because we are given the fact that the distance between two charging stations will be no more than 299m. Now, we will take m and compare that to 300, since the car can only go for 300m without charge. If m is greater than or equal to 300, we have to remove the third distance, and we print out the first two charge stations as that is the distance of charge stations she can travel without charge. Then we remove the two charge stations from the array C, since we know the distance between two charging stations will never be over 299m. However, if m is less than 300, we first remove the three charging stations from the array and we can add one more charging station distance to m. Now, we check again if m is less than or greater than or equal to 300. If m >= 300, we remove the distance of the charge station that we just added, and we print out the first charge station we added and the last charge station we added. If m < 300, we remove the charge station we added previously from the array and add in another one. We repeat this process until we get m > 300.

Algorithm Detail:

Given n number of charging stations.

Each n is given a distance between the next charging stations ahead of it.

Assume car is charged full at each charge station

Sort out list of n by closest charging station (Buffalo) to farthest charging stations (Seattle)

Place n into an array C.

While C isn’t empty

int m = 0

int x = 3 where x will represent the charge station number after the third charge station

ci will be the charge stations and will always point to the first entry in C

Take the first three charge stations in C: ci, ci+1, and ci+2

m = ci + ci+1 + ci+2

Check if m is < or >= 300

If m >= 300

m = m – ci+2

Remove ci and ci+2 from C

Print, you can travel from ci to ci+1 without charge and must charge at ci+1

Else if m <= 300

Remove ci, ci+1, and ci+2 from C

1 If ci+x exists

2 Take the next charge station in C: ci+x

3 m = m + ci+x

4 Check if m is < or >= 300

5 If m > 300

6 m = m – ci+x

7Print, you can travel from ci to ci+x-1 added without charge and must charge at ci+x-1

8 Else if m <= 300

9 Remove ci+x charge station from C

x = x + 1

Repeat steps 1 - 9 from above until we hit and do if m >=300 or we reach Seattle

Proof Idea: My algorithm does every condition that the question is asking for. My algorithm checks the minimum amount of times to charge to travel from Buffalo to Seattle. My algorithm also checks that she doesn’t get stuck between two charge stations. I will also sum up each part of my algorithm to show that it runs in 0(nlogn).

Proof Detail: My algorithm fulfills all the necessary conditions that the problem gives us. Since we know that the car can only go up to 300 miles max without charge, we can sum up the distance of a certain amount of charge stations to see if they are less than or greater than or equal to 300. If it’s less than or equal to 300, we can assume that she can travel more, so we add in another distance. If it’s greater than 300, she can’t travel from the starting charge station to the last charge station because we know she can only travel 300 max. We then remove the last charge station added from the total distance added. Each time we hit the max she can go, it prints out that she can travel from starting charge station to the last charge station added without charge and must charge at the last charge station added. This process repeat itself until C runs out, which means she has hit Seattle. This also checks that she is never stuck between two charge stations.

Run time Analysis: I will prove that my algorithm runs in 0(nlogn). The first part of the algorithm, sorting the list of n charge stations, take up to O(nlogn). Any type of sorting that we do is done in O(nlogn). Next part, inputting the list into an array will take O(n) times since we are putting a list of n into an array. For the while loop, it is running until C runs out, so the worst case it can be is O(n) because it will always hit every entry in C at least once. For everything inside the while loop, we have a couple of if else statements, removing element from array, print statements and some simple addition and subtraction. If-else statement run in O(1), as well as the simple addition and subtraction and print statement. Removing each charge station from the array will only be O(1) since we are removing at most 3 charge stations at once, not the whole array. So the full runtime for the while loop will be O(n) \* O(1) \* O(1) \* O(1) and you get O(n). Now, to find the overall run time, we add up everything together, which we get O(nlogn) + O(n), which becomes O(nlogn + n), which becomes O(nlogn) since nlogn is bigger than n, so we take the one that’s bigger.

Source: Piazza, lecture notes 20 and 21.