(a) Describe the optimal substructure of this problem

(b) Describe the greedy algorithm that could find an optimal way to schedule the students

(d) What is the runtime complexity of your greedy algorithm? Again, you don't need to factor in the setup of the lookup table, just your scheduling algorithm.

(e) In your PDF, based on your answer to part b, give a full proof that your greedy algorithm returns an optimal solution.

a) Optimal Solution is the solution with the smallest number of switches.

Optimal Substructure would be having a student do as many consecutive tasks before being switched off until all task are done.

Student i does {m,…,n} where m < n and have consecutive numbers from m to n. other students perhaps the same on will have to do {1...m-1} and {n+1...end}. For those subsections you can do the same algorithm.

b) find the highest consecutive task a student is willing to do. If it is highest of all students schedule him. This will make the amount of switching minimum.

d) worse case for finding the student that can finish most consecutive tasks would take at most O(n^2). Also in a while loop that runs O(n) thus the worst case is O(n^3).

e) ALG <S1,S2,…,Sk>

OPT<S'1,S'2,…,S'm> m<k

Up until some point lets say Si we have the same students doing the same tasks. After the ith index we have different students doing different consecutive steps. If we replace S'i with Si our optimality would not change because our algorithm takes the student that can perform the longest consecutive tasks. We can use the cut and paste method for the rest of the task/students. The optimal solution claims m<k thus less students which means less switches. If the optimal solution does have less that means we must have missed one or several steps. This is a contradiction because it claims all task will be done. Therefore, our algorithm is the optimal solution.

(a) Describe an algorithm solution to this problem. Feel free to talk about how you would adapt an algorithm we covered in class.

(b) What is the complexity of your proposed solution in (a)?

(c) See the file FastestRoutePublicTransit.java, the method "shortestTime". Note you can run the file and it'll output the solution from that method. Which algorithm is this implementing?

(d) In the file FastestRoutePublicTransit.java, how would you use the existing code to help you implement your algorithm? The existing code only handles one piece of data per edge, so describe some modifications.

(e) What's the current complexity of "shortestTime" given V vertices and E edges? How would you make the "shortestTime" implementation faster? Describe any algorithm changes or data structure changes. What's the complexity of the optimal implementation?

a) using a modified Dijkstra's algorithm to find the shortest path to the station. And using this with freq and first to calculate the time from start to target. For each station you would add the freq time of the train leaving (waiting time). F

b) from https://www.geeksforgeeks.org/dijkstras-shortest-path-algorithm-greedy-algo-7/) the worst time would be O(n^2). Worse case for calculating the time would be O(n). Then running time is O(n^2 + n) -> O(n^2).

c) Dijkstra algorithm

d) maybe add another array to keep track of the stations along the way. You can use this array to backtrack easier.

e) The current run time is O(n^2) where n is the number of stations. Not sure how to make it faster.