1: Physics experiment:

a)Describe the optimal substructure of this problem

- To minimize the switching between students the optimal substructure of this problem would have to consist of the students with the maximum number of consecutive steps. Let's say the problem started with a universal set U as a union of all subsets makes up the universal set. So after the first iteration, the algorithm will look for any student(s_i) with most consecutive steps. After choosing the students the U' will consist of steps U' - S_i. So now the algorithm will look for the next possible students with most possible matching consecutive to make up a subpart of U' thus yielding U''. Thus choosing the student with the most number of consecutive steps ensures the least number of steps left for other students to do ensuring the optimal solution.

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

-

If we were to schedule the student that can do the most consecutive steps in a row, then we would be using the greedy algorithm that will be able to find the optimal solution. From the start of the algorithm, it will look for any student who can do the most number of consecutive steps. If a student were able to do this, then this would mean that the amount of switching between students would decrease when they try to do all the steps. I.e: so if student one does (x_n, x_{n+1},..., x_{n+i}) then the algorithm will take this one and then it will continue finding the next student with next longest set with consecutive steps starting/including the step i+1; or the student with next longest consecutive steps from the initial step up to x_{n-1} if applicable, until all the steps are

discovered. This algorithm code potentially finds a student who by any chance can do all the steps at once without doing any switches.

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.

- total runtime would be O(m x n) where m is the number of students and n is the number of comparisons/operations.

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

- Proof by contradiction: Let's assume s1:s1,s2,s3...sn is an optimal solution. There exists a t:t1,t2,t3...tm such that t is more optimal than s. So by our assumption m<n. Thus there is a point where t does less switching than s. Let's assume up to s1,s2,s3 and t1,t2,t3, both are equal. But at s4 and t4 both differ. Theoretically, t4 would do more consecutive steps(longer sets of consecutive steps) than s4 does result in less switching than the optimal solution given by s. But by our definition s4 always picks the set with the longest consecutive steps performed by a student at any given steps, contradicting the definition of optimality.

2: Public Transit:

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

The algorithm implements the usual Dijkstra's shortest path algorithm with few modification. The first modification would be finding a path from a given vertex to another compared to Dijkstra which looks for the shortest path from the source vertex to every other vertex. Another most important modification would be accounting for the wait time. In Dijkstra's algorithm, each edge represented a weight. But in this specific problem, we also have to account for each weighted edge(time is taken from u-v) + the wait time at each station.

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

At each step for each modification, this algorithm takes a constant amount of time to computes the next train arrival time. So it preserves the actual runtime of the Dijkstra's algorithm. So Dijkstra's algorithm takes O(v^2 + e log v) as it is represented by a matrix and by using a priority queue to keep track of the vertices.

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?

- Dijkstra’s shortest path algorithm

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.

- As mentioned in part a, the given(shortestPath) code preserves the algorithm of Dijkstra's shortest path algorithm with the exception of accounting the wait time at every station. So from vertex u to adjacent vertex v(station), the algorithm

calculates the edge weight, its need to be modified to account for the weight time for the next train at all adjacent vertex v from u. Implementing the ideas of the shortest path; also shortest wait time at the next adjacent vertex(station) in the path of the destination vertex.

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?

- the current runtime for the “shortestTime” time is O(v^2) as there are 2 nested for loops and each loop executed v times at worst case. But this runtime can be improved by implementing some changes to the data structures. Min-heap can be implemented to keep track of the vertices that are visited rather than visiting all the vertices. This process takes O(lgv) time at every execution. To check the neighbour vertices adjacency list can be implemented rather than a matrix to improve runtime. At worst case number of e(# of edges) may equal to v resulting O(elogv) runtime. This can be further brought down to O(E + VLogV) using Fibonacci Heap. The reason is, Fibonacci Heap takes O(1) time for decrease-key operation while Binary Heap takes O(Logn) time.