1.

a. An optimal substructure, by definition, is, if the problem in question, can be broken down into sub-problems to find an optimal solution to its subproblems, whilst, efficently. The optimal solution in this problem is to complete all the steps in the lowest number of switches. The optimal substructure by virtue of being done by the same student and their ability to have a consecutive scheduled list, we can select a plan with minimum switches in n steps

b. A greedy algorithm that can be applied here is to schedule those students with the greatest amount of consecutive steps. If we list in non-descending order, we'll have a method to minimize the instances of switches between the students that, of course, fulfill the requirements, that does the basics of an greedy algorithm: make the best instant/local decision in hopes of a globally optimized solution. Pick students with the longest consecutive-scheduled number of jobs that have the first job scheduled. After so, look for the longest consecutive jobs.

c. Coded

d. My algorithm at the beginning starts with a while loop. "0 < steps..." Once, steps has decremented to 0 then it will stop. At the worst case, my algorithm should take O(n^3), The code relating to student that finish the greatest number of steps. Taking into account both instances, the worst case is O(n^3).

e. Assertion: Greedy Algorithm (GAB) described in B, returns an optimal solution.

Proof by contradiction:

Assume that there exists an optimal solution (OPT) that is a better solution than the greedy algorithm (GAB) that I proposed. OPT can produce a list of scheduling times for the given students with less switches relative to GAB and the same number of experiments as GAB.

Let the switches that GAB produces be:

$GAB = \{ G_1, G_2, G_3, \ldots, G_T \}$ where T is arbitrary

Let the switches that OPT produces be: where K is arbitrary

$$OPT = \{P_1, P_2, P_3, \ldots, P_K\}$$

where $K < T$, and the requisites - number of students having the greater number of scheduled jons ~~picked~~ the condition of being picked first.

Let w be the initial switch where G_w is not similar to P_w. Using the greedy algorithm, GAB, we can produce a list with the least number of switches between the group of students that are registered for the experiment up until T switches. By using the cut and paste methodology, to cut ~~and~~ O_w and paste in G_w, to the extent of K would produce:

$$OPT = \{G_1, G_2, G_3, \ldots, P_{w+1} \ldots, P_K, \ldots\}$$

where $G_1, G_2, G_3, \ldots, G_i = P_1, P_2, P_3, \ldots, P_w$

GAB now is the result of all the switches except at P we havent produced the entirety of switches that will ensure the fulfillment of all experiments; therefore, we have reached a contradiction of the assumption that $P < K$. Thus our greedy algorithm will provide us with the optimal solution to produce a list with the least number of switches for any number of students, and still have all of our experiments conducted

2. a. I would use Dijkstra's algorithm. Dijkstra's algorithm, most often, is used to find the shortest path to a target node from a source node. Dijkstra's algorithm, using the source node, builds a set of node with the least distance to the source node. This is reminiscient of greedy algorithm by virtue of finding the shortest path from a set node; producing, a globally optimized solution.
One advantage, although, a double edge sword, is a faster run time than Bellman-Ford's algorithm but an inability to deal with negatively weighed edges.

To implement Dijkstra's algorithm some changes are needed.
The waiting time at a station needs calculations and to produce a minimum cost journey.
The helper matrix is referenced to ensure minimum cost from source node to destination node.
To find the next train that can be caught to travel to the next station is given by this formula:
The time it takes to get between two adjacent stations u and v added to how frequently the train stops at u on its way to v multiplied by a arbitrary index subtracted by the start time.
I would initialize 2 arrays for shortest time and boolean array.
I would initialize the greatest value and false for these arrays
After calculating the number of vertices from given time in adjacency vertices, find the subsequent vertex to run calculations
Check its neighboring nodes and calculate the cost of those nodes. If minimum cost is undermined, then update min. cost, and repeat

b. It is $O(V^2)$.

c. It implements Dijkstra's algorithm.

d. Since the existing code handles one piece of data per edge, I could do this to help me implement my algorithm: Take into account the waiting time at station to get a connecting train and the time of the connecting train.

e. Current complexity is $O(v^3)$. It can be optimized to a complexity of $O(E+V \log V)$ by implementation of minimum priority queue.

Sources of inspiration:

Nods,

brilliant.org/wiki/shortest-path-algorithms/