1. **Experimental Scheduling**
    a. The optimal substructure for this problem would be scheduling the student that can perform the maximum number of steps and requires the minimum number of switches. The problem can be structured as scheduling the student that has the most steps to begin with and then the scheduling the next student that has the most steps, that would not overlap the steps the previous student completed and keep moving forward. This will in turn maximize the number of steps completed and minimize the number of switches.
    b. The greedy algorithm that could find an optimal way to schedule the students is by scheduling the student with the greatest number of steps until there are no more steps to be assigned which would lead to the minimum number of switches.
    c. Code in PhysicsExperiment.java
    d. The runtime complexity of my greedy algorithm is O(n^3).
    e. Proof: Let ALG be the solution to my algorithm and OPT represent the optimal solution.

        ALG: <U1, U2, …, UX>
        OPT: <V1, V2, …,VY> where Y<X
        Our algorithm is designed to pick each student that can complete the maximum number of steps at each selection period. Let's say Si is the point in where we have students working on the same steps, after the ith index, we have students working on different ascending steps. Now, say we replace the V1 with U1, the optimality will not change. The optimal solution claims Y < X, which means less students which leads to less switches. After the reaching of index i, Ui != Vi, however this is not possible because the OPT solution must have missed some steps. We can come to this conclusion because the algorithm has to choose the student that can complete the most steps in each iteration. Thus, by contradiction it is proved that our algorithm is correct.

2. **Public, Public Transit**
    a. In order to solve this problem, I would use the Dijkstra algorithm. I would use it to find the shortest path and by modifying this algorithm, there would be a stored record that would track the path of each station that was stopped at, eventually leading to the shortest path. The wait time for the train at each stop from one station to the next would also be tracked to get the shortest path.

        https://developers.google.com/optimization/routing/vrp
        https://www.lancaster.ac.uk/media/lancaster-university/content-assets/documents/stor-i/interns-docs/HelenBlue_POSTER_small.pdf
        Links above used for reference
    b. The time complexity of the Dijkstra's algorithm would be $O(v^2)$.

        https://www.quora.com/What-is-the-complexity-of-Dijkstras-algorithm
        Link above used for reference.
    c. The algorithm being implemented is Dijkstra's algorithm.
    d. I would use the existing code to help implement my algorithm by creating a way to take into consideration the extra waiting time depending on whether or not the train comes at the current time. I would consider the total weight time and make an array to keep track of the information.

e. The current runtime complexity of "shortestTime" is $O(v^2)$. The runtime complexity of the optimal implementation is O(|E|+|V|log| V|), which can be achieved with a Fibonacci heap and if a Binary heap is used, the time complexity would be O((|E| + |V|) log |V|).
   https://www.quora.com/What-is-the-complexity-of-Dijkstras-algorithm
   Link above used for reference
f. Code in FastestRoutePublicTransit.java
g. Extra credit in FastestRoutePublicTransit.java

I also looked at the course repository to get an idea of how to go about coding and solving the problems.