Part 1

a) The Optimal Substructure is at each step (i) there is a list of students who can complete the tasks at i.

b) Loop through each step
    if studentCanGoConsecutively
        UpdateScheduleTable
        Subtract 1 from consecutive number
        continue for loop
    Loop through each student
        if studentHasStep
            increase counter for student
            Loop through all steps starting from current one
                if student can do next step
                    increase counter for student
                else
                    break loop
    Find student with most consecutiveSteps
    Store them in booleran array (index is student number, value is true if used)
    Update table
    decrease counter for student

d) $O(mn+O)$ m = steps O = check n = students

e) Loop invariant:
Initialization: A 2D Array that has an integer value set to 1 if student can do a step and 0 if they cannot. An integer value denoting the total amount of steps.
A second 2D Array that can store the steps and amount of students initialized to 0. A 1D Array the size of number of students, initialized to 0. An integer that holds the index for a student initiall set to -1;

Maintenence: At the i^{th} step, the loop will check if the the validStudent index is greater than 0 and if there is any other turns left for that student. If there is, it will update the table, subtract 1 from the consecutiveStudent[validStudent]. If there is no other consecutiveStudents[validStudent] is 0, reset the remaining consecutive student values. Then continue through the loop.
If the valid student is less than 0 and the consecutiveStudent is 0 loop through every student:
At the j step of students, the loop will checks if the student has the current i^{th} step. If he does increase the consecutie count by 1 and loop through the remaining steps starting at i
At the k step, the loop will check if the student can perform the next step and contiue through the loop
The loop will then select a student who has the most amount

of consecutive turns, giving preference to those who already went store the student in a was used array. Update the table and reduce the consecutive turns by 1.

Termination: The loop will terminate once it has reached the final step.

Part 2

a) Run Djikstra's algorithm using the length matrix. While the algorithm calculates the minimum distance, I would search for the next node that needs to be processed and the looks throught the remaining nodes and check the value of my current node against the others. I would store the value of the first train in a variable and if the current time is less than the first time the train comes, simply checks if a temporary min value (default set to highest integer) is less than the first time the train comes - the current time + the length it takes from the current node to the next node. If the current time is greater than the first, add the freq of the train to the value of the first until the value is greater or equal to the current time. Store the value of the minimum node based on the above calculation and check if the node value is less than whatever is already stored in the times array and store the value. Store the value of the vertices in a seperate array. After checking the current node against the others increase the total time to the minimum value found. Checks against all nodes to find the shortest time.

b) $O(V^2)$

c) Djikstra's Shortest Path

d) In the current code, I plan to store the calculations found for each edge using the inner for loop. The calculation would checks if the user has and edge from u to v and the total up the amount of time it would take if you use when the train first arrives or if you need to continuosly increase based on the frequency. I would then add the length to that value and check in the if statement if the current node is less than the value stored in the times array. If it is, store the value. Then add to the totalMinutes after leaving the inner loop.

e. $O(V^2)$. It can become $O(E \log V)$ with a binary heap.
With a binary heap I will be able to traverse the data structure a lot faster and more efficient than a 2D array.