Group members: Senhai Qu, Jian Wei Kang

1.) Running Trials and Tribulations

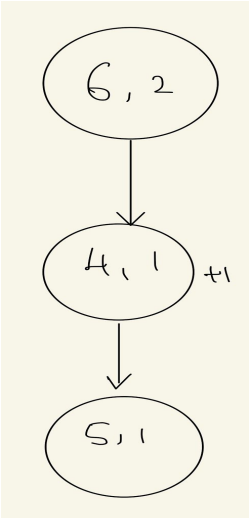
a) Describe the optimal substructure/recurrence that would lead to a recursive solution.

The optimal substructure/recurrence that would lead to a recursive solution is by recursively calling the “runTrialsRecur” function, and applying -2 on the “possibleSpeeds” parameter as well as applying -1 on “days” parameter, then, +1 on every time calling the function recursively. Which, will look like

“runTrialsRecur(possibleSpeeds -2, days-1) +1;”

To explain why we came up with this solution, it is because everytime the function recursively called, we will take one day off to simulate a day has passed, and the -2 on “possibleSpeeds” simulate the situation where the athlete is injured so we decrease the speed down by 2 and try again the next day, then the +1 on the recursively called function indicate the athlete has done the test one time if not injured.

c) Draw recurrence tree for given (#speeds = 6, # days = 2)



d) How many distinct subproblems do you end up with given 6 speeds and 2 days?

With the given 6 speeds and 2 days, there will end up with 2 subproblems, which is from (6 speeds, 2 days) to (4 speeds , 1 day) and +1 (to indicate the amount of speed test + 1) which we will get (5 speeds , 1 day).

e) How many distinct subproblems for N speeds and M days?

If there are N speeds and M days, the amount of distinct subproblems will be **(N - (M - 1) * 2) + (M - 1)**.

f) Describe how you would memoize runTrialsRecur.

The way of implementing memoization of “runTrialsRecur” will be if the value of **array[possible][days]** is updated, we return the array values, else we call “runTrialsRecur” recursively and keep updating the array value.

2.) Holiday Special - Putting Shifts Together

a)Describe the optimal substructure of this problem.

The optimal substructure of this problem is to be able to see which cook can do the most number of consecutive steps for the recipe, so that way we can assign the most steps to most consecutive steps cook and switch to another cook who know fewer steps.

b)Describe the greedy algorithm in plain words that could find an optimal way to schedule the volunteers for one recipe.

In order to do greedy algorithm to find the optimal way to make schedule, we have to find which cook is able to do the most consecutive steps as well as if the cook can do up to the last step. I have a nested for loop to run through the entire numCooks and numSteps to cover all the elements, then I use another for loop to indicate the next step of the step of the current cook, and I use variable called "indicator" to indicate the number of steps that can be done by the cook. When we know that the cook can do the current step then our next goal will be to check whether the cook can do the most consecutive steps or not. If we found a cook who can do the last step then that cook will be assigned to the scheduletable since it there will be no further steps to check. If the cook is not able to do the last step, then I use another loop to iterate the next step and store the data for the maximum amount of steps that the cook can do. If I found the cook that can do the most amount of steps consecutively I store the cook into the scheduleTable along with the steps and the steps ahead. I also skip the steps that are already been taken so will no be scheduled.

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

If we denote the number of shifts with n and number of cooks with m and max sequence step with l then the time complexity will be Q(n) + Q(m) + 2Q(log2(l)).

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

To proof that my greedy algorithm returns an optimal solution, let's go through the answer from part b. Since we know that the optimal solution should be having the cook assigned to a recipe with at least switches between cooks as possible. In the recipe for Homemade cranberry bread which has 4 cooks and 8 steps. Our first thing to do is to see if cook 1 contain the final step or not, in this recipe the cook 1 don't have step 8 so we see if cook 1 contain the most consecutive steps or not. Yes, cook 1 contain the most amount of consecutive steps, so register cook 1 into the table, and now go to the next cook. Cook 2 has 5, 7, 8 and 5 is taken but having 7 and 8 is not the most amount of consecutive steps. We skip cook 3 because it doesn't contain the most amount of consecutive steps as well. Cook 4 is actually what we are looking for, because it has 1, 5, 7, 8, and so far the steps pool didn't include steps 1, 7, 8, and having 3 steps is the most amount of consecutive steps other than cook 1's, so we put it into the schedule Table. The result will be Cook 1: 2 3 4 5 6 and Cook 4: 1 7 8. This is an optimal solution because there are only two switches. Cook 4 will do step 1, and switch to cook 1 for step 2 to 6, then switch back to cook 4 for step 7 and 8. 2 switches is the least amount of switch for this recipe. We can also prove by contradiction. Saying that the optimal solution will have a better solution than our greedy algorithm, which is also not possible because 2 switches is the best possible solution for any algorithm.

3.) League of Patience

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

In the algorithm we need to be concerned about the time until the next quest time at each arrival at a location, so in the algorithm, we will update the time value of all adjacent vertices of the picked vertex, then mark current vertex that has the shortest time from source as visited. If the time from current vertex to each of all other vertices is shorter, then we keep updating the shorter time and add the time between start time and the time to get to the next quest, so that we will be taking the journey time into account here.

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

Although the time cost of journey from one to each of all other places is hard to determine, but the overall purpose is similar to the well-known Dijkstra's algorithm. The algorithm in (a) is to update the time value of all adjacent vertices first, then marked the current vertex as visited, it is the same idea as Dijkstra's algorithm. Then, look at all other adjacent vertices to determine which one has the shortest time and the shortest waiting time. The process so far is similar to Dijkstra's algorithm, so the runtime should be O((V+E) log V) as each operation takes O(log V) and there are V vertices and E edges. Then, the difference is that we added the waiting time value, but the time is constant, so the overall complexity would still be O((V+E) log V).

c) **See the file LeagueOfPatience.java, the method "genericShortest". Note you can run the LeagueOfPatience.java file and the method will output the solution from that method. Which algorithm is this genericShortest method implementing?**

The "genericShortest" algorithm is implementing Dijkstra's algorithm. As as the answer from (b) states, the overall implementation of solution (a) is very similar to Dijkstra's algorithm, and the only difference between "genericShortest" algorithm and algorithm in (a) is we take the waiting time into account. Therefore "genericShortest" algorithm is implementing using Dijkstra's algorithm.

d) **In the file LeagueOfPatience.java, how would you use the existing code to help you implementing your algorithm? The existing code only handles one piece of data per edge, so describe some modifications. Note the helper method available to you, including the one that simulates the game's API that returns the next quest time.**

The data that the existed code handle is the shortest time between vertex to vertex, and the goal for the "myFastestPlay" algorithm is to put the waiting time in concern. Therefore, on top of the existing code, we have to use other methods to find the time until the next quest time at the arrival at a location. Senhai made a method to simulates the start time as we get to the location before we know time for when the next quest is, called "getstartTime". Then we use "minutesBetween" method to return the time of the moment we start until we know the time for when the next quest start, and the value we get back will be the time until the next quest we have to wait in order to move to the next quest. We will then add value to the updated time on the current location. The helper methods we used are the "getNextQuestTime" and "minutesBetween".

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

The current complexity of "genericShortest" algorithm is O((V + E) log V), and there is another way to make the implementation faster. A* runs much quicker than Dijkstra's algorithm although both of them serve the same purpose. The main point of using A* algorithm is because it uses the heuristic function to guide its way towards the goal very quickly. In Dijkstra's algorithm, if it wants to go toward the goal it will have to save all the adjacent paths and check which path has the shortest path before starting to move. While, A* algorithm will mostly lean toward the goal, meaning it will only involve the with paths that are more toward the goal, so it doesn't need that much time to save all the adjacent paths and search through all of them to find the shortest path. It uses like Greedy Best-First-Search so that it uses a heuristic to guide it-self.

Summary:

1) We have divided up the work between each other fairly. Senhai is the one writing the codes for Question 1, “Running Trials and Tribulations” and Question 3, “League of Patience”, while Jian Wei Kang is the one writing the codes for Question 2, “Holiday Special - Putting Shifts Together”. In questions beside coding wise, we also take equally fair amount of responsibilities. We discussed the answers and think through them together, but Senhai Qu is responsible for answering most of the questions as well as the Extra Credit section for Question 1 (a, c, d, e, f) JianWei Kang answered most questions on Question 2 (a, b, d, e), and we both contributed equal amount of answers for Question 3, Senhai contributed for (a, d, e) and JianWei Kang contributed for (b, c). Also, most with the answers that are provided by Senhai, JianWei Kang takes most of those answers and wrote it into the writeup documents.

2) Experience
    a) In perspective of Jian Wei Kang:

        I truly appreciate the opportunity for us to be working together as a team. We learned how to cooperate together to come up with the answer that we both agree on. Working as a team is a great way to fill learn everything from others, so I will hope to contribute as a team member again.

    b) In perspective of Senhai Qu:

        Working together with Jian Wei is a great experience and I am happy to contribute my parts for this project. I also learned a lot of stuff that strengthen my skills and prepare me better for the upcoming final Exam.