1. Running Trials and Tribulations

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

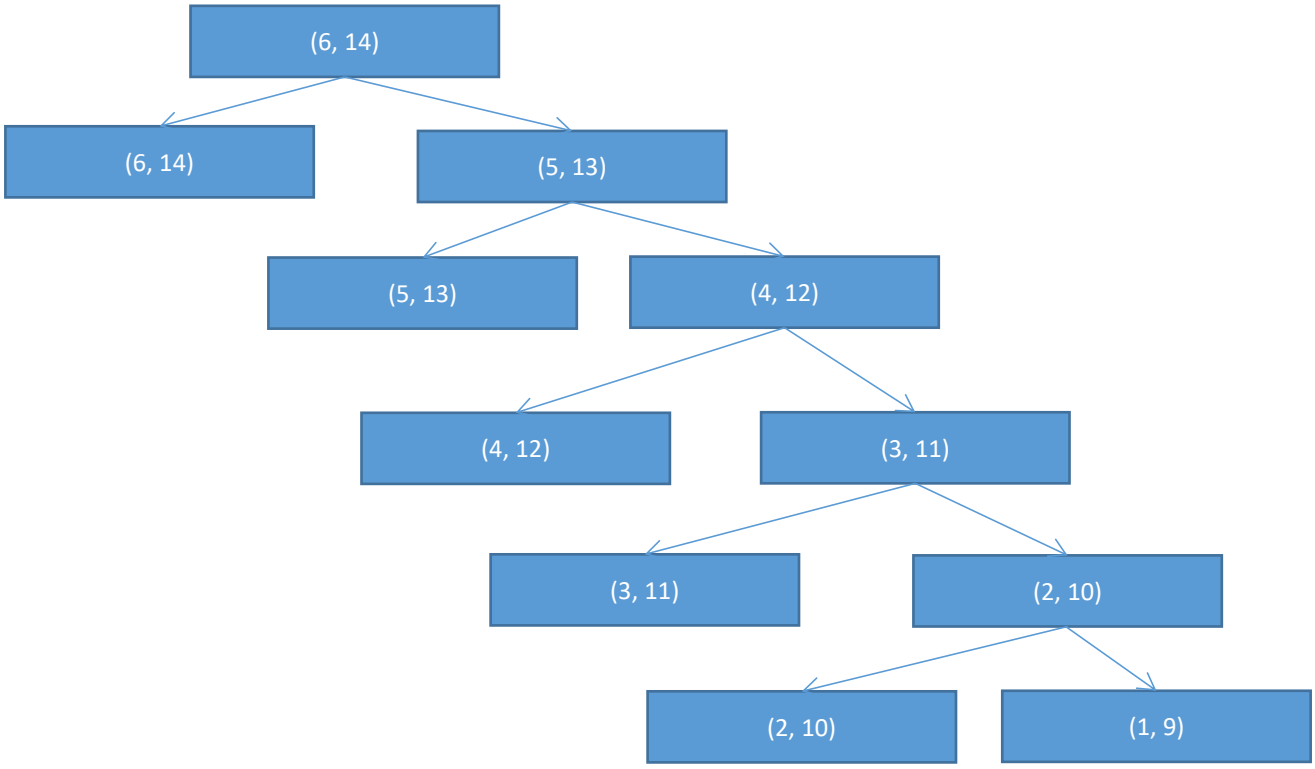
Recursion in computer science is a method of solving a problem where the solution depends on solutions to smaller instances of the same problem.

Such problems can generally be solved by iteration, but this needs to identify and index the smaller instances at programming time. At the opposite, recursion solves such recursive problems by using functions that call themselves from within their own code. The approach can be applied to many types of problems, and recursion is one of the central ideas of computer science.

When recursive is called, program save its states from top to ends and, when result is returned from end, it calculate results from end to top by sequence.

In top state, final result is returned.

(b) 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?

In my solution, there are 7 subproblems.
We can suppose injuries can be possible about each speed test.
And Whenever injuries is in, one day is lost, so we can make subproblems by decreasing speed range and days.
(In there, 2 means 2 weeks, we can suppose train days will be 14 days(2 * 7)).

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

If we suppose that day is enough to test all speed range, the number of subproblems is (N + 1).

(f) Describe how you would memoize runTrialsRecur.

e can suppose injuries can be possible about each speed test.
And Whenever injuries is in, one day is lost, so we can make subproblems by decreasing speed range and days.
(In there, 2 means 2 weeks, we can suppose train days will be 14 days(2 * 7)).
In each problem, it call recursive decreasing day and speed one by one and if day or speed is 0, bottom problem return result and get all results from bottom to top problem.

2. Holiday Special - Putting Shifts Together

(a) Describe the optimal substructure of this problem.

This problem is to find optimized switch number that satisfy to pass all steps.
Each cook has its own skills and they can pass steps with some of their skills.
The most important thing is to pass all steps and optimize switch number. For this, Greedy optimize algorithm is used for solution of this problem.

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

First, it finds cook who has greatest skills that can pass possible steps.
One Cook pass steps and program get rest steps.
For next, it search other cook who has greatest skills for rest steps.
Finally, it find cookies till to pass all steps by repeating above steps.
The switch number will be optimized.
This is Greedy search for this problem.

(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 step number is M, cook number is N, time complexity is O(M * N)

(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.

Greedy algorithm is always to search optimize solution for next state and it repeats this steps till to find final results.
In my algorithm, it always search cook who has skills that can be passed possible steps.
So, each selected cook will pass possible steps and reduce steps as possible as he can.
Finally, the number of switch between cookies is optimized.

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.

Form source, it finds all possible path and select node that has shortest path for next source.
By repeat this steps, it find shortest path from source to destination.

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

`T = O(VE) V: node number, E edge number`

(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 algorithm used in genericShortest is dijkstra's Algorithm.

(d) In the file LeagueOfPatience.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. Note the helper methods available to you, including one that simulates the game's API that returns the next quest time.

API is simulating waiting time, so we can suppose that completed time is duration + API time(wait time).
So, we can reference genericShortest method, but we have to modify time part.
Using getNextQuestTime method, we can get API time between node u and v and have to add this time to duration(graph[u][v]).
Next, it has to store of each node start date(real date) and for next step, if any node is selected, start time has to be replaces from stored date.
By repeating this steps, it find taken time from source to destination.

(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?

Time complexity is O(V, E).
There are lots of algorithms to find shortest path.
The one of faster algorithm than dijkstra can be Ant Colony Optimization algorithm.
This give very good result in short time , and user can set parameter how much it has to be run.
But it might not give us not perfect results by increase nodes and complexity.