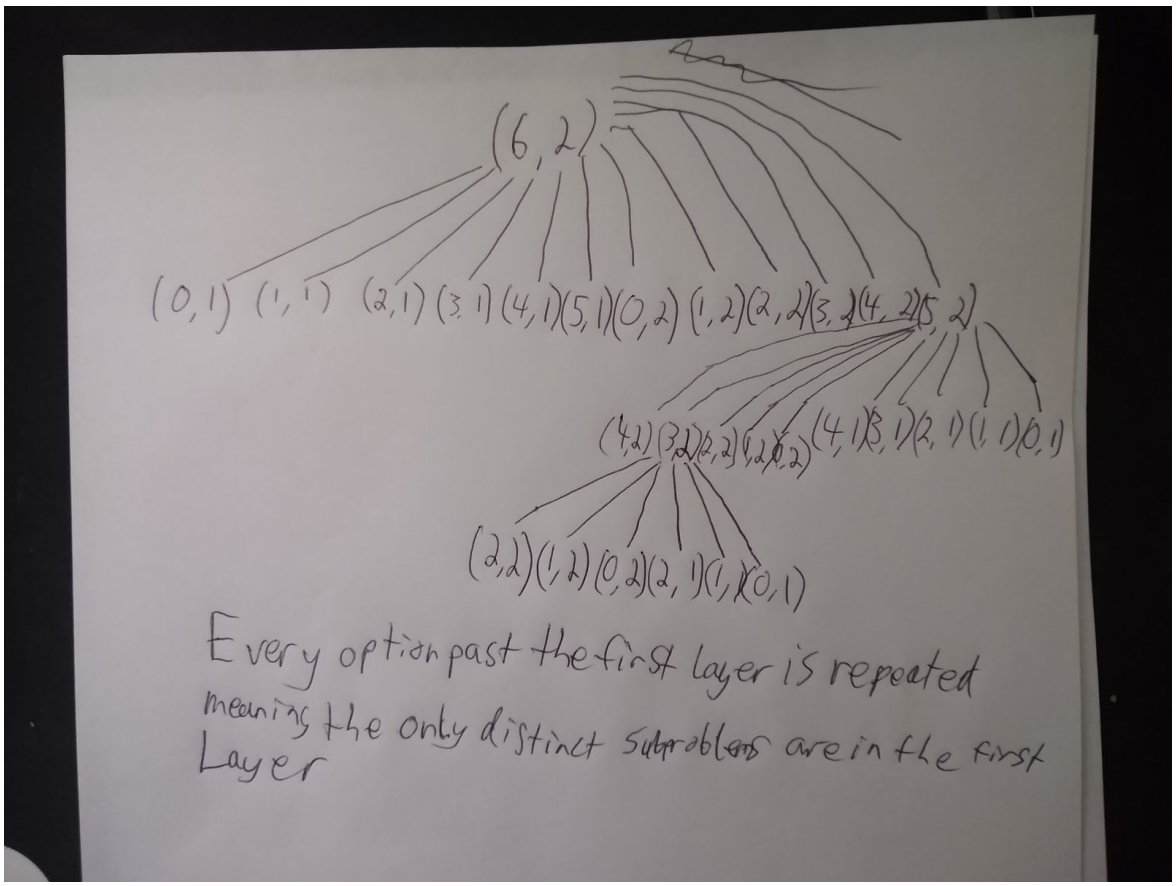
1) Running Trials and Tribulations

Given: M days, N paces

Find: What's the minimum number of speedtest runs the athlete needs to go on to figure out the fastest speed they can train for a race at without getting injured?

a) The optimal substructure is that for the number of paces as the result for a faster pace can be solved by breaking down N paces from n to 1 or 0 and then recursively finding the minimum number of trials needed for each pace.
b) See Code
c)



d) 12 distinct subproblems. 2 * n
e) There are 2n subproblems, as at each pace there are two options that you have, which are the runner was injured or they were not.
f) Create an array that would store the recursive call results, and then check if you have already calculated the recursive call in question, and if you have, you just look up the answer instead in the results array. This would cut out the need to recursively call overlying subproblems, as they would be stored in the memoized array.

2) Holiday Special

Given: n number of steps, m number of Cooks that give you a list of steps (sorted) they can participate in. Assume there's a lookup table where you can find if Cook X signed up for step Y in O(1), so no need to factor that into your runtime.

Find: An optimal way to schedule Cooks to steps such that there is the least amount of switching as possible.

a) The optimal substructure would be to find the cook that can do the most steps consecutively for the recipe, then switch to another cook and repeat. This is a greedy approach.
b) Greedy Algorithm: the greedy algorithm needed to be used is an optimal switching sequence. This can be done by finding the earliest largest sequence of steps. Starting at step 1, you find the largest sequence of steps that any single cook can take, then at the end of the sequence you look for the next largest sequence a cook can take. Repeat until all steps are completed.

d) The runtime complexity where n is the numSteps, and c is the numCooks, is that since there is the initial while loop to n, and inside of that while loop there is a second while loop that will iterate at most c times, meaning it is in sn^2 time, or $O(cn^2)$

e) Proof by contradiction: Let's assume that there is an optimal solution that produces a better solution than my greedy solution. The opt solution can produce a list of scheduling times for the given cooks that can achieve the same number of recipes, but with fewer switches than my solution produces while still finishing all recipes.

Let the number of switches that my algorithm produces be
$G = \{G_1, G_2, \ldots, G_k\}$
Let the number of switches that the optimal solution produce be
$O = \{O_1, O_2, \ldots., O_j\}$

Where $j < k$ and the num of cooks with the most consecutive scheduled steps are picked first. Let i be the first switch where G_i, != O_i By design of my greedy algorithm we can give a list with the least number of switches between the group of cooks that are registered for the recipe up until k switches. Therefore, using the cut and paste method to cut out O_i and paste in G_i. This would make the optimal solution look like

$O\{G_1\ G_2 \ldots.\ ,\ O_i + 1,\ \ldots.\ O_j\}$, where $G_1, G_2 \ldots., G_i = O_1, O_2 \ldots., O_i$

My greedy algo only ends when all possible switches are produced, but at j we haven't produced all the switches. This contradicts our earlier assumption that $j < k$, which means that my greedy algorithm does in fact provide an optimal solution.

3. League of Patience

a) I would solve this problem using Dijkstra's Algorithm that instead of finding the shortest path to all nodes given a source node, it would find the shortest path from the source node to the destination node.
b) The time complexity of the algorithm is V^2, as it is essentially just a slightly modified Dijkstra's that runs at V^2 time.
c) genericShortest is simply using Dijkstra's algorithm.
d) I modified the minutesBetween helper function to return the time in minutes instead of seconds to use instead of manually finding the difference in time based on milliseconds using the getTime function that the Date data structure has.\
e) The time complexity of genericShortest is $O(E + V^2)$ or $O(V^2)$ as V^2 dominates the runtime. If you were to use something like an adjacency list and a binary heap you could cut the time down to O(E log V)