February JDCC Solutions + Comments

Problem A (Palindrome):

Solution: Loop through the string, checking if the **i**th char equals the **length – 1 - i**th char.

Time Complexity: O(N)
Space Complexity: O(N)

Problem B (Ants on a Log):

Solution: The key here is to notice that the ants turning around is the same as the ants walking through eachother. Then, the answer is simply the distance of the farthest ant from the end of the log.

Time Complexity: O(N)
Space Complexity: O(1)

Problem C (Sorting Trains):

Solution: We first note that we don't actually have to take the cars out of the tracks to solve the problem. We want to figure out how to optimally position the cars to use the least amount of tracks. The solution happens to be placing each car into the track whose last car is as large as possible and less than the car you're trying to place. If all the last cars are bigger, then you need to use another track. Note that we only need the last car in each track, so we use an array to store the last cars.

Time Complexity: $O(N^2)$
Space Complexity: $O(N)$

Better Solution: If we use an ordered set instead of an array, we can get the largest car value less than the current one in log(n) time.

Time Complexity: $O(N \log N)$
Space Complexity: $O(N)$

Note: This problem is a restatement of the longest decreasing subsequence problem, so any solutions for this problem work for that problem and vice versa.

Problem D (Roadwork):

Solution: This problem is asking for the longest path between any two nodes in the tree, which is also known as the tree's diameter. For the first 50% of test cases, it is enough to run a search (BFS or DFS) on each building to find the building that is farthest away, then print the maximum distance you find.

Time Complexity: O(N^2)
Space Complexity: O(N)

There is also a faster way to find the diameter. Start at any building and find the one that is farthest away. Then start at that building and find the longest path. The length of that path will be your answer.

Time Complexity: O(N)
Space Complexity: O(N)

Problem E (Pocket Change):

Solution: This problem is a combination of the famous coin dynamic programming problem and knapsack problem.

We first calculate the minimum number of coins that the grocer needs to use to give any amount of money between 0 and 10,000. Our dp state stores the minimum number of coins required for some value and our transition state is to try using any of the coins as the current one.

We then calculate the maximum number of coins that Rosie can use to make every value between 1 and 10,000. We do this using a knapsack algorithm which is nearly identical to the coin dp. We loop through all the coins and all the values and store the best of using or not using the current coin. Since at every point we only need the dp states of the previous coin, we can use some clever overwriting to only use a 1 dimensional array.

Once we do this, we simply loop between K and 10,000 and figure out at which value Rosie would lose the most amountof coins by subtracting the knapsack result and adding the coin dp result at each value.

Time Complexity: O($NL + ML$), where L = 10,000 is the maximum sum of the coins.
Space Complexity: O($N + M + L$)