October JDCC Solutions + Comments

Problem A (Guessing Game):

Solution: This problem only requires you to check two possible cases for the person's number, so check both and see which meets the requirements.

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

Problem B (Programming Elections):

Solution: As you read in every name and vote count, check if the person has received more votes than the best result so far. If so, make them the best result, otherwise continue.

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

Problem C (Test Candy):

Solution: Read in all the values and store them in an array, finding their average in the process (add values to a sum as you read them in, then divide it by n). Then, add (50 – avg) to each value, and see how many are greater than or equal to 50%.

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

Problem D (Basically Right):

Naïve Solution: Try converting both numbers to every valid base and seeing what their ratio is, if it's greater than the best ratio found so far, then update the ratio.

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

Best Solution: After some analysis, one can notice that only the smallest and largest possible bases need to be considered, as the ratio either constantly increases or decreases.

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

Note on rounding: In java, you can use [out.printf ("%.6f", doubleVar);] to automatically round to 6 decimal places.

Problem E (Estuary):

Solution: There are a number of ways you can approach this problem, with some being faster than others. I'll cover a few of them below.

Solution 1 (Sweep Line):

1) Start with the water level being the elevation of (0, 0) + 1, and then mark the start as visited.
2) Iterate through the grid, figuring out the elevations of all points adjacent to visited ones, storing the minimum.
3) Update the water level to be [1 + minimum found in 2)], update the volume to be the change in water level * the # of visited locations and repeat step 2), making sure you update adjacent points less than the water level to be visited.
4) Stop once you reach the end.

Complexity: $O((RC)^2)$

Solution 2 (Recursion on fixed water level):

1) Start with the water level being the elevation of (0, 0) + 1.
2) Run a recursive method on (0, 0) with the current water level. This method recurses on adjacent points which have an elevation less than the current water level, stops once it reaches the end point, and returns the total volume.
3) If the recursion doesn't manage to get to the end, increment the water level by 1, then run 2) again.
4) Once the recursion reaches the end point, the volume it returns is the answer.

Complexity: $O(RCE)$, where E is the maximum elevation.

Solution 3 (Recursion + Binary Search):

1) Set a lower bound (e.g. 1) and an upper bound (1,000,000) on the water level.
2) Set the water level to be the midpoint (average) of the lower and upper bound. The rest of the step is the same as the second solution.
3) If the recursion reached the end, store the volume found, mark upper bound as the midpoint - 1, then go to step 2). If the recursion didn't reach the end, mark the lower bound as the midpoint + 1, then go to step 2)
4) Break once the lower bound isn't less than the upper bound, the last volume that your recursion found is the correct one.

Complexity: $O(RC \log E)$

Solution 4 (BFS with Priority Queue):

This solution is a much more advanced version of the first solution, I'm adding it since it mimics the behaviour of the river perfectly and thus sort of intuitive.

Setup: Point class storing coordinates and elevation of a point, PriorityQueue of points (prioritizes lowest elevation), 2d array storing visited.

1) Add the start to the queue, mark it as visited.
2) Pull the next point from the queue, update the water level and volume based on this point's elevation, then add all adjacent nodes to the queue.
3) If you reach the end, don't add the adjacent points and break out of the while loop once the next point in the queue has an elevation greater than the water level.
4) Once the while loop breaks, you have your volume.

Complexity: O(RC log (RC))