October JDCC Solutions + Comments

Problem A (Guessing Game):

Solution: This problem only requires a check of 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 every person and vote count is read in, 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 they're read 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 because the ratio either strictly increases or strictly decreases as the base is incremented.

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

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

Problem E (Estuary):

Solution: There are a number of ways one can approach this problem, I'll cover two of which here:

Solution 1 (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. Then run the following recursive method:

1) 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, marking them as visited.
2) The recursion terminates if it reaches (R-1, C-1) or if all of its adjacent points are visited.
3) The recursion returns the total volume found.

3) If the recursion reached the end (if (R-1, C-1) is marked visited), store the volume found, set the upper bound to be 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 the recursion found is the correct one.

Time Complexity: O(RC log E)
Space Complexity: O(RC)

Solution 4 (BFS with Priority Queue):

This solution is a more advanced, however it mimics the behaviour of the river perfectly and thus is more intuitive.

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

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 and repeat.
3) If the end is reached, don't add the adjacent points, but don't stop yet; stop once the next point in the queue has an elevation greater than the water level of the end point.

Time Complexity: O(RC log (RC))
Space Complexity: O(RC)