Problem Set 8 - Computational Geometry *(Sorry for the length, there was a lot to talk about…)*

Question D

This question was the biggest dumpster-fire of a problem that I have done in the whole course - and how appropriate that it was the last question that I would do in this course. As a disclaimer, I worked very closely with a friend for this solution, sharing code-snippets, ideas, working through test-cases, and ultimately sharing the same approach to solve the question. The initial approach we discussed was SO clean in theory. But after handling edge case, after edge case - it just became a mess of if statements and special cases that ultimately led us to the final answer. Usually, as the solution starts to become too messy, the intuitive thing to do is to pivot - but sunken cost fallacy kept us in the game, and just tenaciously kept on digging up edge cases, solving them until it finally worked, and boy was it satisfying when the 'Accepted' finally popped up. The initial algorithm was built off the same premise as the trapezoidal rule - precompute, for every pair of red dots, the number of black dots below it. If we're going from left to right, then the precomputation will be positive, from right to left, the precomputation will be negative. Finally, we run the same trapezoidal rule algorithm, summing up our pre-computed values as we track around the circle. After I had initially implemented this, came a whole slew of unexpected edge cases to handle.

In no particular order, some things that needed to be handled were:

- When checking if a black dot was under a line, two checks needed to be made - if it was between the x values (boundaries not included), and if it was under the line (using a counterclockwise/clockwise check, depending on the direction of the line).
- When going from left to right (inclusion), we needed to include everything strictly under the line, but from right to left (exclusion), we needed to also exclude stuff on the line, so we needed to relax the ccw check on the right to left check to also allow collinear points to be counted.
- Additionally, the 'between the x values' check also would fail to count black points that lie directly below the every red value. So we needed to include one endpoint of each red-red segment as well. The way to do this was to precompute how many black dots are below each (strictly below for left to right and softly below for right to left, similarly to prior), and add/subtract in the black dots below for ONLY the **end vertex** of every segment (to avoid double counting).
- A special case for this occurred at the furthest right and furthest left points of the polygon, where - if the rightmost point had a single vertex - we would add the dots under this vertex, but fail to subtract it (as we've only subtracting from the end vertex of the segment going to the left from this point, and not the start). This was solved by ignoring the additions/subtractions of black dots directly under red dots if they were at the leftmost or rightmost value in a polygon.
- Another edge case that needed to be handled was vertical line segments that occurred (not on the rightmost/leftmost x values of the polygon, as they would be ignored). To keep it brief, after doing some testing, we found out when going right, we want to subtract the black dots under the starting point, and add back the ending point of the vertical segment. And when going left, add the dots under the starting point, and subtract the dots under the end.