Problem Set 8 - Computational Geometry

<u>Question B</u>

This question wasn't a super difficult one algorithm-wise. I think one of the first thoughts I had was pretty much the correct algorithm. If we take an arbitrary set of 3 non-collinear points called ABC, and then test each edge of the spanning triangle. For each edge (take AB), draw a line through AB and mark off every point that is on that line. And then check if there is a line through C that contains every other point that wasn't marked off in the first go.

While implementing though, this algorithm did change slightly, at first, I pretty much coded exactly what I outlined above, looking for a set of three non-collinear points by taking the first two points, and cycling through the rest, marking the first that wasn't collinear.

Then I wrote a check function that takes in these three points, drawing a line through the first two, and then checking if a line through the third and any other point picks up the rest of the points. However, the implementation was getting a little bit messy this way, and as I was coding and debugging, I realised that I didn't really even need to go through the step of finding the set of three non-collinear points at the beginning. If I take the 0'th and 1st index points first and second point, draw a line through those, and then take the third as the first point that wasn't hit by that line, we've achieved the same task. I can then do this for the 1st and 2nd index, and then the 0'th and 2nd index as starting points, to achieve the same effect as I did above.

If the first three points were collinear, they would just get crossed off by the drawing of the first line, and we would find another distinct third. But if they weren't, it would be the same as getting the triangle as in the original algorithm, and would still work.

This new implementation method ALMOST got me there. But I was still failing a test case for some odd reason. Looked through all the logic and had not a clue why it was failing… asked a friend and they mentioned rounding issues - and that since the points given were all lattice points, we could use long longs to handle the cross product, as they would never have a decimal component. Changed the functions and points to pairs of long longs, and it worked…

I remember at first when thinking about the algorithm, thinking that lattice points had something to do with the approach/algorithm, otherwise why else would they specify something like that. Turns out they were simply for fixing the rounding issues? Or maybe there was an alternate algorithm that took advantage of them idk.

Regardless, it worked so we take those :)