Problem Set 8 - Computational Geometry

Question A

This question at first was quite easy to wrap my head around, the solution that I had thought of off my intuition was just to choose an initial set of three points that form a triangle. Then, I would iterate through every single one of the other points, check if it lies within the triangle, and if so, replace one of the points in the triangle with it. Because of this, the triangle will only ever shrink, and once we've checked every single point, we can guarantee we will be at a triangle that does not contain any others.

The harder part about this solution was trying to figure out how to check if a point lies within the triangle spanned by three points - probably will become a trivial question once I get more familiarised with Computational Geometry, but at the point of doing this question - I had watched the lectures a week ago, and then before that - hadn't touched geometry and cross products since high school…

Ultimately I decided the best and cleanest way to do this was to do a cross product between every edge and the point, and check that the point is always on the same side of every side in the triangle. Slightly more formally written:

```
// to check if something is in a triangle, define triangle ABC and point P
// AB X AP, BC X BP, CA X CP - all must be the same sign.
```

I also understood that this question could probably be done a hundred ways, so prior to coding this up, I had a chat with a friend to ask how he did it - and he outlined a WAY better solution, way easier to code up, which is the one I ended up using.

The solution was to sort all the points by x value, and then iterate through and find the first three non-collinear points, pretty genius. Unfortunately, life can never be that easy, and after coding it up, I was still failing a test case. What I realised was that the question was asking for all other points to be STRICTLY outside the triangle chosen. So, if my input points looked like:

0 0
1 -10
1 10
1 5
1 -5

My code would choose points 1, 2, 3, but should be choosing 1, 4, 5. Took me a while to come up with and actually discover this test case, but it wasn't too hard to solve. Just needed to - where the x values were equal - sort the y values increasing in terms of magnitude. After that, we were good, because it would always choose the smallest triangle for points with the same x value, guaranteeing no overlaps with other points of that same x value.

After this was solved, the solution was accepted :)