Note The due date for this assignment is **Friday** 12/6, so that it will not be due too soon after the midterm and break. However, the last problem set will be due soon after, on Wednesday 12/11 (the last day of class). It will be posted by Wednesday 12/4. So I would recommend doing as much of this one as possible by Wednesday to leave time to move on to the next.

Written problems:

1. This problem explores the reasons why the primes p and q is DSA can be chosen to have somewhat different sizes.

   Suppose that p, q, g are DSA public parameters (i.e. p, q are primes, and g has order q modulo p), and $A \equiv g^a \pmod{p}$ is Samantha's public (verification) key, while a is her private (signing) key. As we discussed in class, there are two main sorts of algorithms that Eve might use to extract a from A: collision algorithms (whose runtime depends on q), and the number field sieve (whose runtime depends on p). For simplicity, assume that Eve has a collision algorithm that can extract a in $\sqrt{q}$ steps, and an implementation of the number field sieve (a state of the art DLP algorithm; you do not need to know any details about it, but the textbook has a good overview) that can extract a in $e^{2(\ln p)^{1/3}(\ln \ln p)^{2/3}}$ steps (the true runtimes would involve a constant factor that would depend on implementation, and various other factors depending on the cost of arithmetic modulo p and of finding collisions).

   (a) Suppose that Samantha is confident that her private key will be safe as long as Eve does not have time to perform more than 2^{64} steps in either algorithm. How many bits long should she choose p to be? How many bits long should q be?

   (b) What if she instead wants to be safe as long as Eve doesn't have time for 2^{128} steps?

   (c) The NSA's recommendation for "Top Secret" government communications is to use 3072 bit values of p, and 384 bit values of q. How does this compare to your answers above? If the difference is significant, what might explain the discrepancy?

   For parts (a) and (b), it is sufficient to write a short script to find the minimum safe numbers of bits by trial and error (there are more efficient ways, of course).

2. Textbook exercise 6.1 (Elliptic curve arithmetic over $\mathbb{R}$)

3. Textbook exercise 6.5, parts (a) and (b) (Listing the points of an EC over $\mathbb{Z}/p\mathbb{Z}$)

   Hint. You can save some time by making two lists in advance: values of y^2 for various y and values of $x^3 + Ax + B$ for various values of x, then checking for numbers occurring in both lists)

4. Textbook exercise 6.6(a) (addition table for an elliptic curve over $\mathbb{Z}/5\mathbb{Z}$)

Programming problems:

1. Write a function `ecAdd(P,Q,A,B,p)` to compute the sum $P \oplus Q$ of two points on the Elliptic Curve over $\mathbb{Z}/p\mathbb{Z}$ defined by $Y^2 \equiv X^3 + AX + B \pmod{p}$. You may assume that P and Q are both valid points on the curve[1]. The points P and Q will be either pairs (x, y) of elements of $\mathbb{Z}/p\mathbb{Z}$, or the integer `0` (as a stand-in for the point $\mathcal{O}$ at infinity), and the function should return the result in the same format.

[1] Though of course if you were using this code in real life, you should add some error handling that checks this.

2. Write a function `ecMult(n,P,A,B,p)` that computes an integer multiple $n \cdot P$ of a point P on an elliptic curve $Y^2 \equiv X^3 + AX + B \pmod{p}$. Points will be formatted (x, y), with $0 \leq x, y < p$, while the point at infinity should be denoted simply as 0. Your code will need to be able to scale to very large values of n; I suggest adapting the fast-powering algorithm from modular arithmetic to elliptic curves.