Written problems

Note: You may need to wait until Friday's class to do and comptutions involving adding an elliptic curve point to itself (or read about this case in the textbook).

1. Textbook exercise 6.1 (Elliptic curve arithmetic over $\mathbb{R}$)
2. 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)
3. Textbook exercise 6.6(a) (addition table for an elliptic curve over $\mathbb{Z}/5\mathbb{Z}$)
4. Textbook exercise 6.9 (listing all solutions n to an equation $Q = n \cdot P$ on an elliptic curve).
5. Textbook exercise 6.16. (A more concise way to send EC points; you should read Proposition 2.26 to do part (b))

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.
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.
3. Write a function `ecDLP(P,Q,A,B,p,q)` to solve the elliptic curve discrete logarithm problem, in cases where the prime p is up to 28 bits. Here, P, Q are points on the curve $Y^2 \equiv X^3 + AX + B \pmod{p}$, and you are given, for convenience, the number q of points on the curve (which you may assume to be prime). The function should return the minimum nonnegative n such that $n \cdot P = Q$. Note that if you find any such solution n', then you can find the minimum solution by computing $n' \pmod{q}$.

   A naive trial-and-error approach will earn partial credit, but to solve all test cases I recommend adapting the BSGS algorithm to the setting of elliptic curves. Note: one issue you may encounter is that it is not possible to place a "list" (e.g. `[2,3]`) into a Python `dict`. To resolve this, make sure all of your elliptic curve points (besides $\mathcal{O}$) are represented as "tuples" instead (e.g. `(2,3)`, with parentheses instead of brackets).

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