Written problems:

1. Let $p \geq 3$ be a prime number, and let g be a primitive root modulo p. For any $h \in (\mathbb{Z}/p\mathbb{Z})^\times$, denote by $\log_g h$ the solution $x \in \{0, 1, \cdots, p-2\}$ to the congruence $g^x \equiv h \pmod{p}$.

   (a) Prove that this notation is well-defined, i.e. prove that for each choice of h there is a *unique* $x \in \{0, 1, \cdots, p-2\}$ solving this discrete logarithm problem.

   (b) Prove that $\log_g h$ is even if and only if h has a squre root modulo p (a "square root modulo p" is a solution to the congruence $x^2 \equiv h \pmod{p}$).

   (c) Prove that for all $h_1, h_2 \in (\mathbb{Z}/p\mathbb{Z})^\times$,

   $$\log_g(h_1 h_2) \equiv \log_g h_1 + \log_g h_2 \pmod{p-1}.$$

2. Textbook exercise 2.16 (Big $\mathcal{O}$ notation examples; six parts)

3. Let f, g, h be postitive-valued functions, and assume that $h(x) \geq 1$ for all x. Prove that if $f(x) = \mathcal{O}(h(x))$ and $g(x) = \mathcal{O}(1)$, then $f(x) + g(x) = \mathcal{O}(h(x))$ and $f(x)g(x) = \mathcal{O}(h(x))$.

4. For any positive integer n, denote by $B(n)$ the number of bits of n. In other words, $B(n)$ is the unique integer such that

   $$2^{B(n)-1} \leq n < 2^{B(n)}.$$

   (a) Let $f(n)$ be a function from positive integers to positive real numbers. Prove that $f(n) = \mathcal{O}(n)$ *if and only if* $f(n) = \mathcal{O}\left(2^{B(n)}\right)$.

   (b) Prove that $f(n) = \mathcal{O}(\sqrt{n})$ if and only if $f(n) = \mathcal{O}\left(\sqrt{2}^{B(n)}\right)$.

   (c) Let d be a positive integer. Prove that $f(n) = \mathcal{O}((\log n)^d)$ if and only if $f(n) = \mathcal{O}\left(B(n)^d\right)$.

5. Use the babystep-giantstep algorithm to solve each of the following discrete logarithm problems. Show your calculations, e.g. in the form of the table on page 83 of the textbook.

   (a) $10^x \equiv 13 \pmod{17}$

   (b) $15^x \equiv 16 \pmod{37}$

   (c) $5^x \equiv 72 \pmod{97}$

   (a) $x \equiv 1 \pmod{3}$
   $x \equiv 2 \pmod{5}$

   (b) $x \equiv 6 \pmod{11}$
   $x \equiv 2 \pmod{10}$

   (c) $x \equiv 2 \pmod{3}$
   $x \equiv 1 \pmod{10}$
   $x \equiv 3 \pmod{7}$

   (d) $x \equiv 6 \pmod{8}$
   $x \equiv 3 \pmod{9}$
   $x \equiv 0 \pmod{17}$

Programming problems:

1. Implement the Babystep-Giantstep algorithm, efficiently enough to solve the discrete logarithm problem for primes up to 40 bits in length. That is, write a function `bsgs(g,h,p)` that finds an integer x such that $g^x \equiv h \pmod{p}$. You may assume that p is prime, $1 \leq g, h < p$, and that a solution x exists.

   The test bank on Gradescope will be identical to the test bank for `disclog` on Problem Set 2, but now your code must solve all 40 cases for full credit.

   (It is fine to implement any algorithm you can devise to solve the discrete logarithm problem, but BSGS is probably the easiest to implement based on what we've discussed in class.)

Note. If implemented a certain way, your solution to this problem can be almost exactly identical to your solution to the following problem, and indeed can consist of a one-line call to the function you write for the following problem.

2. Write function `bsgsBoundedOrder(g,h,p,N)` to solve the discrete logarithm function $g^x \equiv h \pmod{p}$ under the assumption that the order of g is at most N. You may assume that p is a prime number, but unlike the previous set it will be quite large (256 bits). The integer N will be various sizes, up to 40 bits. Any correct solution x will be marked correct, even if it is not the smallest possible solution.

Note. The time limit is set to 1 second on Gradescope as usual, which may be slightly tight on the largest cases. Let me know if you are running into the time limit so that I can suggest improvements. I may extend the time limit if it seems that generally efficient implementations are not quite meeting it.

3. Suppose that you are given two integers m_1, m_2 with $\gcd(m_1, m_2) = 1$, and two integers a_1, a_2. Write a function `crt(a1,m1,a2,m2)` that efficiently determines the unique integer x such that

$$\begin{aligned} x &\equiv a_1 \pmod{m_1}, \\ x &\equiv a_2 \pmod{m_2}, \end{aligned}$$

   and $0 \leq x < m_1 m_2$. The fact that x exists and is unique comes from the Chinese Remainder Theorem.

4. (This will be an ingredient in the Pohlig-Hellman algorithm, to be discussed soon) Write a function `ppFactor(N)` which accepts an integer $N \geq 2$, and returns a list of the prime powers (all powers of different primes) factoring N, in any order. For example, if $N = 12$ the function should return either `[4,3]` or `[3,4]`. The integer N may be quite large (up to 1024 bits), but you may assume that all of the prime-power factors are 16 bits or smaller.

   Suggested approach: There are many ways to do this, and certainly many more efficient than what I'm about to describe, but here is one relatively quick-to-implement approach. Write a `for` loop to iterate through all numbers p from 2 to 2^{16}. For each number, check whether it divides N. If so, divide N by p repeatedly until it is no longer divisible by p (and replace N by the new value), then add the appropirate power of p to the list you will eventually return. As long as you shrink N as you go, you will never find that $p \mid N$ unless p is in fact prime, since any smaller factor would have already been found to divide N.