Refer to the second page of the Course Survey for instructions on submitting written work on Gradescope, and to the instructions on Problem Set 1 for developing and submitting programming problems.

Written problems

1. Textbook exercise 1.24, part (a). (Solving two simultaneous congruences)

2. This problem shows one way in which congruences are compatible with exponentiation. The important point to notice is that **a different modulus is relevant in the exponent than in the base**. Suppose that p is a prime number, and $g \in (\mathbb{Z}/p\mathbb{Z})^\times$.

   (a) Suppose that n is any integer such that $g^n \equiv 1 \pmod{p}$, and a, b are two integers such that $a \equiv b \pmod{n}$. Prove that $g^a \equiv g^b \pmod{p}$.

   (b) Suppose now that n is the **order of** $g \pmod{p}$, and assume that g is a unit modulo p. Prove that for any two integers a, b,

   $$a \equiv b \pmod{n} \text{ if and only if } g^a \equiv g^b \pmod{p}.$$

   (For one direction of the "if and only if," you simply need to cite part (a). The other direction requires a separate argument, using specific properties of the order.)

3. Let p be prime, $y \in (\mathbb{Z}/p\mathbb{Z})^\times$, and let k be a positive integer such that $\gcd(k, p-1) = 1$. Prove that the congruence $x^k \equiv y \pmod{p}$ has a unique solution modulo p (here, "unique modulo p" means that if x_1, x_2 are both solutions then $x_1 \equiv x_2 \pmod{p}$).

   Hint: use the fact that k has an inverse modulo $p-1$, and apply Problem 2, part (a).

4. Textbook exercise 1.33, part (a). (a way to find order-q elements)
   Also read part (b) and think about it; you don't need to write up a solution, but the statement may help you think about how to solve Programming Problem 3).

5. Textbook exercise 1.34, parts (a), (b), and (c). (Identifying primitive roots by hand for some small primes)

Programming problems

1. Write a function `modpow(a,n,m)` that takes integers a, n, m and returns $a^n \pmod{m}$. The exponent n may be positive or negative (it will be positive in at least half of the test cases), but you may assume that if $n < 0$, then a is a unit modulo m (you'll need to use your code from the previous problem to handle the $n < 0$ case). **For this problem, please do not use Python's built-in function for modular powers. You should implement the fast-powering algorithm yourself to see how it works. However, you may look up and use the built-in function on all programming assignments after this one.** The test cases will range in size, with a and n 256-bit integers in the largest case.

2. Write a function `dh(g,p,B)` that performs Alice's role in Diffie-Hellman key exchange, using a randomly chosen secret number a. More precisely: you are given a prime number p, an element $g \in \mathbb{Z}/p\mathbb{Z}$, and Bob's transmission B. Your function should return a pair of two numbers A, S, where A is the number you transmit to Bob, and S is the shared secret. You should look up how to generate random numbers in Python (don't worry about finding a

"cryptographically secure" random number generator, just use the standard one in Python). The prime p will be between 16 and 256 bits long, and the autograder will run your code several times on each input to make sure that it appears to be choosing the secret number a randomly.

3. Write a function `findOrderQ(q,p)` that takes two prime numbers q and p and returns an element $g \in \mathbb{Z}/p\mathbb{Z}$ of order q if possible. See Written Problem 4 for a way to do this. If it is *n*ot possible to find such an element g, your function should `return None`.

4. Write a function `kthroot(k,y,p)`, that takes an integer y, a prime p, and an element $y \in \mathbb{Z}/p\mathbb{Z}$, and determines an element $x \in \mathbb{Z}/p\mathbb{Z}$ such that $x^k \equiv y \pmod{p}$. You may assume that $\gcd(k, p-1) = 1$ in all test cases. In the largest test cases, p will be 256 bits long, but a naive approach will receive partial credit.

   Hint: make use of Written Problem 2, part (b).