Written problems:

1. Evaluate the discrete logarithm $\log_{40} 33$ in $\mathbb{Z}/73\mathbb{Z}$ using the Pohlig-Hellman algorithm, according to the following steps (see the statement of Theorem 2.31 in the textbook for details on the notation). You may use, without proof, the fact that 40 is a primitive root modulo 73.

   (a) Let N be the order of 40 $\pmod{73}$. Factor N into prime powers as $N = q_1^{e_1} \cdots q_t^{e_t}$.

   (b) Determine the numbers g_i and h_i for each i from 1 to t inclusive. For each i, what is the order of g_i modulo 73?

   (c) For each i, evaluate the discrete logarithm $y_i = \log_{g_i} h_i$ in $\mathbb{Z}/73\mathbb{Z}$, using a method of your choice.

   (d) Solve the system of congruences $x \equiv y_i \pmod{q_i^{e_i}}$ to obtain the discrete logarithm $x = \log_{40} 33$.

2. Suppose that p is a large prime (e.g. 1024 bits), g is a primitive root modulo p, and Alice has an Elgamal public key A corresponding to a private key a (that is, $A \equiv g^a \pmod{p}$, and Alice knows the number a). Bob does not believe that Alice actually knows the private key a corresponding to A, so she asks her to solve the following challenge to prove it. Bob will give Alice a positive integer d of his choosing. Alice must return (in a reasonable amount of time) *two* integers b, c such that

$$g^b \cdot b^c \equiv A^d \pmod{p}.$$

   If she succeeds, Bob will be convinved that Alice really does know her private key.

   (a) Describe a procedure that Alice can use to solve Bob's challenge efficiently.
   Hint. Choose an integer e at random, and choose b to be $g^e \pmod{p}$. Then find a choice of c.

   (b) Explain briefly why Bob should be convinced that Eve (or anyone else who doesn't know the private key) would not be able to carry out the procedure you describe in part (a).

   Note. This exercise prefigures the basic idea behind Elgamal "digital signatures," which we will discuss soon. You can solve this problem without knowing anything about signatures, however.

3. This problem considers a slightly more general form of a fact proved in the course of develoepd the Pohlig-Hellman algorithm. You may want to refer to your notes from that discussion for ideas on how to proceed.

   (a) Suppose that a, x, y, m are integers such that $a \mid m$. Prove that $ax \equiv ay \pmod{m}$ if and only if $x \equiv y \pmod{\frac{m}{a}}$.

   (b) Suppose that p is a prime number, g, h are units modulo p, and x is a solution to the discrete logarithm problem $g^x \equiv h \pmod{p}$. Let $n = \operatorname{ord}_p(g)$ and let q be an integer such that $q \mid n$. Prove that for any integer y, the congruence

$$(g^{n/q})^y \equiv h^{n/q} \pmod{p}$$

   holds if and only if $x \equiv y \pmod{q}$.

   (c) Briefly explain how the fact above was involved in our discussion of the Pohlig-Hellman algorithm.

4. In this problem, you will empirically investigate a famous theorem called the *Prime Number Theorem*, and some of its variations. We will discuss this theorem, these variations, and their relevance to cryptography soon.

   (a) Using any method you wish, determine the number of primes exactly n bits long for each of the following values of n: $4, 8, 12, 16, 20$. (Note: a quick way to do this without writing any code is to use Wolfram Alpha; it can correctly answer questions of the form "number of primes between a and b". You can also use your Miller-Rabin code (one of this week's coding problems), or implement the Sieve of Eratosthenes, or use any other method you can think of to count primes).

   (b) The simplest of of the Prime Number Theorem says that the number of primes less than or equal to n is approximately $\frac{n}{\ln(n)}$. Use this approximation to give a formula estimating the number of primes in a closed interval $[a, b]$ (where $a, b \in \mathbb{Z}$), and determine the number of exactly n-bit primes this predicts for the five values of n in part (a).

   (c) Another version of the Prime Number Theorem says that the number of primes in a closed interval $[a, b]$ is approximated by the sum

   $$\sum_{m=a}^{b} \frac{1}{\ln(m)}.$$

   (Informally, you can pretend that "the probability that m is prime is $\frac{1}{\ln(m)}$." This is nonsense if taken literally, but it is a useful fiction: the sum above would then be the expected value of the number of primes between a and b inclusive, since it the sum, for each m in that interval, of the probability that m is prime.)

   Compute the number of exactly n-bit primes predicted by this estimate for the same five values of n. (You can compute this however you like; it can, for example, be done with a few lines of Python code, or using Wolfram Alpha. State in your write-up exactly how you computed it.)

   (d) Summarize the numbers you found in parts (a) through (c) in a table. Discuss the relative accuracy of the two different estimates in parts (b) and (c).

   Note If you're interested (not part of the course), you can try to prove that the two estimates, despite looking different, are the same asymptotically (that is, the limit of their ratios converges to 1 as $b \to \infty$). This can be done by approximating the sum with an integral, applying integration by parts, and finding some bounds on the result.

   (e) Modify the approximation methods from parts (b) and (c) to instead approximate the number of primes $p \equiv 1 \pmod 5$ of each of the five bit lengths, and compute the exact number of such primes (Wolfram Alpha can do this as well; ask me if you're having trouble getting it to understand the question). Construct a table like in part (c) to compare the true value and the two estimates. Briefly discuss any observations you can make from this table.

5. (Solve the Miller-Rabin programming problem first, so that you have code that you can use to count Miller-Rabin witnesses) For each integer n between 1,000,000 and 1,000,009 inclusive, determine the proportion of the numbers from 1 to $n - 1$ inclusive that are Miller-Rabin

witnesses. Which of these numbers are prime? (The figures you obtain should convince you that the 75% figure from Rabin's theorem is rather conservative, and explains why most people are not worried about using only a few Miller-Rabin trials to test primality).

Programming problems:

1. Write a function `crtList(ls)` that takes a list `ls` of pairs (a_i, m_i) of integers, with any two of the values m_i relatively prime, and returns a pair (a, m) such that the system of congruences $x \equiv a_i \pmod{m_i}$ is equivalent the single congruence $x \equiv a \pmod{m}$, and $0 \leq a < m$ (i.e. a is reduced modulo m).

   For example, `crtList( [(2,3), (3,5), (0,2)] )` should return $(8, 30)$, since the system of three congruences $x \equiv 2 \pmod 3, x \equiv 3 \pmod 5, x \equiv 0 \pmod 2$ is equivalent to the single congruence $x \equiv 8 \pmod{30}$.

   The integer a should be reduced modulo m, i.e. $0 \leq a < m$. The moduli m_i will be integers up to 256 bits in length, and the list will contain up to 128 entries.

2. Implement the Pohlig-Hellman algorithm. That is, write a funciton `ph(g,h,p)` that solves the discrete logarithm problem $g^x \equiv h \pmod{p}$ under the assumption that p is a "weak" prime, in the sense that $p - 1$ factors into small prime factors. More specifically: you may assume that $p - 1$ factors into prime powers, all 16 bits or smaller, but p will be 64 bits in length.

3. Implement the Miller-Rabin primality test (or another primality test of your choice): write a function `isPrime(n)` that returns `True` or `False` according to whether or not n is prime. The starter code will also define a function `checkList` that applies your function to a list of integers; you do not need to modify that part. Each test case will give your function ten integers of the same size; to pass the test case your function must give the correct answer for all ten.

4. Write a function `makeQP(qbits,pbits)` that generates two primes numbers q and p such that q is exactly `qbits` bits long, p is exactly `pbits` bits long, and $p \equiv 1 \pmod{q}$. Recall that p is "exactly n bits long" means that $2^{n-1} \leq p < 2^n$.

   Take a moment to remind yourself why it is useful to construct primes this way, even if it's only the prime p that you want (e.g. for Diffie-Hellman parameters).