NEW SUBMISSION INSTRUCTIONS. Please turn in your homework in THREE parts, each part stapled together, with your name, the homework number and the problem numbers on the front of each part. The three parts will be placed in three separate labeled boxes:

• Problems 1 and 2
• Problems 3 and 4
• Problems 5 and 6

Please state (with problem 1) the names of any persons and references to any resources (including online) you consulted with in connection with this assignment, or state that none were consulted. (There will be 2 points off for failure to do so.)

For this assignment, while you may consult outside resources, any facts that you use in your proofs must be cited from the course text, Notes on Discrete Math, rather than outside resources.

For this homework, log(x) denotes the logarithm to base 2 of x. For the algorithms discussed below, assume that inputs that are numbers (a, b, m, n) are represented as strings of bits in binary. Note that the number of bits to represent n ∈ N is O(log(n)), and the time to multiply or divide (with remainder) two numbers with b1 and b2 bits is bounded above by a polynomial in b1 and b2.

1. (15 points) Let a and b be arbitrary positive integers. Prove that gcd(a, b) = gcd(b, a + b).

2. (15 points) We recursively define the function f : N → N by f(0) = 2, f(1) = 5, and for all n ≥ 2, f(n) = f(n − 1) + f(n − 2). Prove by mathematical induction on n that gcd(f(n), f(n + 1)) = 1 for all n ∈ N. (Hint: you may assume the claim in problem 1.)

3. (20 points) We recursively define the function g from the positive integers to the natural numbers as follows: g(1) = 0, and for all n > 1, g(n) = 1 + g(⌊n/2⌋). Make a table of the function g(n) for 1 ≤ n ≤ 8, and prove by mathematical induction that for all positive integers n,

\[2g(n) ≤ n < 2g(n)+1.\]

4. The goal of this problem is to prove that when 0 < m < n, Euclid’s greatest common divisor algorithm computes gcd(m, n) using a number of recursive calls bounded by O(log(n)).

(a) (5 points) Prove that for all positive integers m and n, if m < n, then when we divide n by m, the remainder is strictly less than n/2. (Hint: use Theorem 8.2.1 and inequality reasoning rather than induction.)

(b) (10 points) Suppose that m and n are positive integers such that 0 < m < n and that when we call Euclid’s greatest common divisor algorithm with gcd(m, n), it makes a recursive call with gcd(m', n'), which in turn makes a recursive call with gcd(m'', n''). Use part (a) to show that m'' < m/2 and n'' < n/2. Give an informal argument that this shows that the number of recursive calls made in the computation of gcd(m, n) is in O(log(n)).
5. (15 points) Use the extended version of Euclid’s gcd algorithm to find the multiplicative inverse of \( a \) modulo \( m \), for the following pairs \((a, m)\). That is, find the *unique* integer \( b \) such that \( 0 \leq b < m \) and \((a \cdot b \mod m) = 1\). Show the relevant steps of the extended Euclidean algorithm in each case.

(a) \( (11, 19) \)

(b) \( (16, 39) \)

(c) \( (10, 33) \)

6. This problem considers an algorithm for exponentiation modulo a positive integer \( m \). We would like an algorithm \( e(a, n, m) \) that takes in natural numbers \( a \), \( n \), and \( m \), where \( m > 0 \), and returns the natural number \((a^n \mod m)\), that is, the remainder on dividing \( a^n \) by \( m \).

One approach would be to compute the integer \( a^n \) by multiplying together \( n \) copies of \( a \) (that is, \((n - 1)\) multiplications), and then dividing by \( m \) to find the remainder \((a^n \mod m)\). This would take time \( \Omega(n) \), but we can do better!

(a) (5 points) One improvement would be to find the remainder modulo \( m \) after every multiplication; this doubles the number of operations, but keeps the numbers from getting “too big”. Use results in Chapter 8 to explain why this gives a correct answer.

(b) (5 points) An additional improvement when \( n \) is a power of two, say \( n = 2^t \), would be to compute \((a^{2^t} \mod m)\) by repeatedly squaring \( a \), that is, \( a, a^2, a^4, a^8 \), and so on, up to \( a^{2^t} \). Use results in Chapter 8 to explain why this gives a correct answer.

(c) (10 points) Using the results in (a) and (b), give an algorithm for \( e(a, n, m) \) for arbitrary natural numbers \( a \), \( n \), and \( m \), where \( m > 0 \) and prove that it runs in time bounded above by a polynomial in \( \log(a) \), \( \log(n) \), and \( \log(m) \).

(Hints: (1) \( 2^{a+b} = 2^a \cdot 2^b \); (2) when we write a number like 11001 in binary, we are writing it as a sum of powers of two: \( 2^4 + 2^3 + 2^0 \) )