Please submit your homework in two separate parts: problems 1-3 in the first part, and 4-6 in the second part. Make sure your name is on both parts. In the first part, please include the names of anyone (including course staff) you consulted with in connection with this assignment, as well as listing any resources (including course materials) you consulted.

**PART I**

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

   **Solution.** We show that for every positive integer $d$, $d$ is a divisor of $a$ and $b$ if and only if $d$ is a divisor of $b$ and $(a + b)$. Suppose $d | a$ and $d | b$. Then there exist integers $k$ and $\ell$ such that $a = k \cdot d$ and $b = \ell \cdot d$. Thus,
   \[ a + b = k \cdot d + \ell \cdot d = (k + \ell) \cdot d, \]
   so $d | b$ and $d | (a + b)$. Conversely, if $d | b$ and $d | (a + b)$, there exist integers $\ell$ and $m$ such that $b = \ell \cdot d$ and $(a + b) = m \cdot d$. Thus,
   \[ a = (a + b) - b = m \cdot d - \ell \cdot d = (m - \ell) \cdot d, \]
   so $d | a$ and $d | b$. Since the set of common divisors of $a$ and $b$ is equal to the set of common divisors of $b$ and $(a + b)$, their greatest common divisors are equal.

2. (15 points) We recursively define the function $f : \mathbb{N} \to \mathbb{N}$ by $f(0) = 1$, $f(1) = 3$, and for all $n \geq 2$, $f(n) = f(n - 1) + f(n - 2)$. Prove by mathematical induction on $n$ that $\gcd(f(n), f(n + 1)) = 1$ for all $n \in \mathbb{N}$. (Hint: you may assume the claim in problem 1.)

   **Solution.** Simple mathematical induction suffices for this proof. The predicate $P(n)$ is the assertion that $\gcd(f(n), f(n + 1)) = 1$. The base case is $n = 0$, for which we claim that $\gcd(f(0), f(1)) = 1$. This is true because, by the definition of $f$, we have $f(0) = 1$ and $f(1) = 3$, and $\gcd(1, 3) = 1$. The inductive hypothesis is that $P(n)$ is true for some $n \geq 0$. Thus, $\gcd(f(n), f(n + 1)) = 1$ by the IH. Then because $n \geq 0$, $f(n + 2) = f(n + 1) + f(n)$, by the definition of $f$. By the IH and the result of problem 1, we know that
   \[ 1 = \gcd(f(n), f(n + 1)) = \gcd(f(n + 1), f(n + 2)). \]
   Because $f(n + 2) = f(n) + f(n + 1)$, this shows that $\gcd(f(n + 1), f(n + 2)) = 1$, that is, $P(n+1)$ is true. Thus we conclude by mathematical induction that $P(n)$ is true for all natural numbers $n$.

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 $n > 1$, $g(n) = 1 + g(\lfloor (n/2) \rfloor)$. Make a table of the function $g(n)$ for $1 \leq n \leq 8$, and prove by mathematical induction that for all positive integers $n$,
   \[ 2^{g(n)} \leq n < 2^{g(n)+1}. \]

   **Solution.**
The proof is by strong induction. The predicate $P(n)$ is $2^{g(n)} \leq n < 2^{g(n)+1}$. The base case is $n = 1$. We have $2^0 \leq 1 < 2^1$, so $P(1)$ is true. The inductive hypothesis is that $P(1), \ldots, P(n)$ are all true, for some $n \geq 1$. To prove $P(n+1)$, we consider two cases, depending on whether $(n+1)$ is even or odd.

If $(n+1)$ is even, then for some positive integer $m$ we have $(n+1) = 2m$, and $n+1 > 1$. Then $g(n+1) = 1 + g(m)$. By the IH, because $1 \leq m \leq n$,

$$2^{g(m)} \leq m < 2^{g(m)+1}.$$  

Multiplying by 2, we have

$$2^{g(m)+1} \leq 2m < 2^{g(m)+2},$$

which, because $(n+1) = 2m$ and $g(n+1) = 1 + g(m)$, yields

$$2^{g(n+1)} \leq (n+1) < 2^{g(n)+1}.$$

In case $(n+1)$ is odd, because $n \geq 1$, there is a positive integer $m$ such that $(n+1) = 2m + 1$. Then $g(n+1) = 1 + g(m)$, and because $1 \leq m \leq n$, we have by the inductive hypothesis that

$$2^{g(m)} \leq m < 2^{g(m)+1}.$$  

If we multiply by 2,

$$2^{g(m)+1} \leq 2m < 2^{g(m)+2}.$$  

Because $g(n+1) = 1 + g(m)$, this implies

$$2^{g(n+1)} \leq 2m < 2^{g(n+1)+1}.$$  

Because $2m + 1$ is odd and $2^{g(n+1)+1}$ is even, we also have

$$2^{g(n+1)} \leq 2m + 1 < 2^{g(n+1)+1},$$

which implies

$$2^{g(n+1)} \leq (n+1) < 2^{g(n+1)+1}.$$  

Thus, in either case $P(n+1)$ holds, and we conclude by strong induction that $P(n)$ is true for all positive integers $n$.

---

**PART II**

4. The goal of this problem is to prove (as claimed in lecture) that when $0 < m < n$, Euclid's greatest common divisor algorithm computes $\gcd(m, n)$ using a number of recursive calls bounded by $O(\log(n))$, thus making it a "Good" algorithm on numbers.
(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 8.1.1 and inequality reasoning rather than induction.)

Solution. We consider cases based on the value of \( m \). If \( m > n/2 \), then the remainder when we divide \( n \) by \( m \) is \((n - m)\), which is less than \( n/2 \). If \( m \leq n/2 \), then the remainder when we divide \( n \) by \( m \) is less than \( m \) (by Theorem 8.1.1), and therefore is less than \( n/2 \).

(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 \( \text{gcd}(m,n) \), it makes a recursive call with \( \text{gcd}(m',n') \), which in turn makes a recursive call with \( \text{gcd}(m'',n'') \). 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 \( \text{gcd}(m,n) \) is in \( O(\log(n)) \).

Solution. Given that these two recursive calls happen, we know that \( n' = m \) and \( m' = (n \mod m) \), and also that \( n'' = m' \) and \( m'' = (n' \mod m') \), according to Euclid’s gcd algorithm. Thus, \( n'' = (n \mod m) \), where \( 0 < m < n \). By part (a), this means that \( n'' < n/2 \).

Also, we know that \( m' = (n \mod m) \) is less than \( m \), but is not 0 (because the algorithm did not return after the first recursive call.) Then \( m'' = (m \mod m') \), where \( 0 < m' < m \). Again by part (a), this means that \( m'' < m/2 \).

Thus, for every two recursive calls in Euclid’s gcd algorithm, both arguments are reduced by more than half. Since both arguments are natural numbers, and the algorithm returns when \( m = 0 \), and \( m < n \), the number of recursive calls is bounded above by \( \lceil 2 \log_2(n) \rceil \), which 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) \((11, 18)\)

Solution.

\[
\begin{align*}
gcd(11, 18) &= gcd(7, 11) = gcd(4, 7) = gcd(3, 4) = gcd(1, 3) = 1 \\
7 &= 18 - 11 \\
4 &= 11 - 7 = 11 - (18 - 11) = 2 \cdot 11 - 18 \\
3 &= 7 - 4 = (18 - 11) - (2 \cdot 11 - 18) = 2 \cdot 18 - 3 \cdot 11 \\
1 &= 4 - 3 = (2 \cdot 11 - 18) - (2 \cdot 18 - 3 \cdot 11) = (5 \cdot 11) - (3 \cdot 18)
\end{align*}
\]

The inverse of 11 modulo 18 is 5:

\[ 5 \cdot 11 \equiv 1 \pmod{18} \]

(Alternatively, the algorithm given in 8.2.2 can be used, if the relevant intermediate steps are included. Similarly for parts (b) and (c).)
(b) (15, 38)
Solution.
\[
gcd(15, 38) = gcd(8, 15) = gcd(7, 8) = gcd(1, 7) = 1
\]
\[
8 = 38 - 2 \cdot 15
\]
\[
7 = 15 - 8 = 15 - (38 - 2 \cdot 15) = 3 \cdot 15 - 38
\]
\[
1 = 8 - 7 = (38 - 2 \cdot 15) - (3 \cdot 15 - 38) = 2 \cdot 38 - 5 \cdot 15.
\]
The inverse of 15 modulo 38 is \(-5\), which is 33 modulo 38:
\[
33 \cdot 15 \equiv 1 \pmod{38}
\]

(c) (8, 13)
Solution.
\[
gcd(8, 13) = gcd(5, 8) = gcd(3, 5) = gcd(2, 3) = gcd(1, 2) = 1
\]
\[
5 = 13 - 8
\]
\[
3 = 8 - 5 = 8 - (13 - 8) = 2 \cdot 8 - 13
\]
\[
2 = 5 - 3 = (13 - 8) - (2 \cdot 8 - 13) = 2 \cdot 13 - 3 \cdot 8
\]
\[
1 = 3 - 2 = (2 \cdot 8 - 13) - (2 \cdot 13 - 3 \cdot 8) = 5 \cdot 8 - 3 \cdot 13
\]
The inverse of 8 modulo 13 is 5:
\[
5 \cdot 8 \equiv 1 \pmod{13}
\]

6. This problem considers a “Good” 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 is a “Bad” algorithm for numbers because the number of multiplications depends linearly on the value of the input \(n\) instead of being polynomial in the number of bits representing all the inputs.

(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. Why is this still a “Bad” algorithm for numbers?
Solution. The result after Theorem 8.4.2 states that if \(x \equiv_m x'\) and \(y \equiv_m y'\), then \(x \cdot y \equiv_m x' \cdot y'\). If we start with \(x_1 = a\) and repeatedly compute \(x_{k+1} = ((x_k \cdot a) \mod m)\), then we can show by induction on \(k\) that \(x_k \equiv_m a^k\) for \(k = 1, 2, \ldots, n\). Thus, the final result will be the correct value of \((a^n \mod m)\). This is still a “Bad” algorithm because the number of multiplications is still linear in \(n\).

(b) (5 points) Another improvement, at least for \(n\) a power of two, would be to compute \((a^{(2^k)} \mod m)\) by repeatedly squaring \(a\), that is, \(a, a^2, a^4, a^8, \text{ and so on, up to } a^{(2^k)}\).
Explain how this idea and the improvement in part (a) can give a “Good” algorithm for $e(a, n, m)$ when $n$ is a power of two.

Solution. To compute $(a^{(2^t)} \mod m)$, we let $x_0 = a$ and $x_{k+1} = ((x_k \cdot x_k) \mod m)$. Then $x_k = (a^{(2^k)} \mod m)$ for $k = 0, 1, \ldots, t$ for the same reason as in part (a). The number of multiplications to compute $(a^{(2^t)} \mod m)$ is $t$, which is linear in the number of bits of $n = 2^t$. This is a “Good” algorithm on numbers because the total cost of the operations is polynomial in the number of bits representing the inputs $a$, $n$, and $m$.

(c) (10 points) Using the results in (a) and (b), give a “Good” algorithm for $e(a, n, m)$ for arbitrary natural numbers $n$. (Hint: when we write a number like 11001 in binary, we are writing it as a sum of powers of two: $2^4 + 2^3 + 2^0$.)

Solution. For the general algorithm, we express the number $n$ in binary. If $2^t$ is the highest power of two that appears in $n$, then we can use $t$ multiplications and remainders modulo $m$ to compute $(a^{(2^k)} \mod m)$ for all $k = 0, 1, \ldots, t$ as in part (b). With at most $t$ additional multiplications and remainders modulo $m$, we multiply together the values of $(a^{(2^k)} \mod m)$ for those powers $2^k$ corresponding to 1’s in the binary representation of $n$. For example,

$$(a^{25} \mod m) = ((a^{(2^4)} \cdot a^{(2^3)} \cdot a^{(2^2)}) \mod m).$$

This is still a “Good” algorithm – the cost of the operations is polynomial in the number of bits representing the inputs $a$, $n$, and $m$, and correctly computes $e(a, n, m)$. 
