```
// Specifications

method GCD1(a: int, b: int) returns (r: int)
    requires 0 < a && 0 < b
    ensures r == gcd(a, b)
    decreases b // b is reduced each iteration which reduces a % b to reach the base case a % b == 0
{
    if a < b {
        r := GCD1(b, a);
    } else if (a % b == 0) {
        r := b;
    } else {
        r := GCD1(b, a % b);
    }
}

method GCD2(a: int, b: int) returns (r: int)
    requires 0 <= a && 0 <= b
    ensures r == gcd(a, b)
    decreases b
{
    if b == 0 {
        r := a;
    } else {
        r := GCD2(b, a % b);
    }
}

// Weakest Precondition Total Proof
// In the following weakest precondition proofs, [x.xx] references a law from Appendix A of Programming from Specifications by Carroll Morgan.

method GCD1(a: int, b: int) returns (r: int)
    requires 0 < a && 0 < b
    ensures r == gcd(a, b)
    decreases b
{
    // This method is totally correct as the given precondition implies the weakest precondition.
    { (0 < b && 0 < a) } // By law [A.01] (each event depends on (0 < b && 0 < a) being true)
    { ((0 < b && 0 < a) || ((0 < b && 0 < a) && (a < b)) || (0 < b && 0 < a)) } // By law [A.06]
    { (0 < b && 0 < a) || ((0 < b && 0 < a) && (a < b)) || ( (a >= b || (0 < b && 0 < a)) && (0 < b && 0 < a) ) } // By law [A.21]
    { (0 < b && 0 < a) || ( (a >= b || (0 < b && 0 < a)) && (a < b) ) || ( (a >= b || (0 < b && 0 < a)) && (0 < b && 0 < a) ) } // The modulus operation is defined for b > 0 and consequently a >= b or a > 0 ==> a, b > 0.
    { ( (a >= b || (0 < b && 0 < a)) && (a % b == 0) ) || ( (a >= b || (0 < b && 0 < a)) && (a < b) ) || ( (a >= b || (0 < b && 0 < a)) && (0 < b && 0 < a) ) } // By law [A.07]
    // In the lines above, spaces have been introduced [e.g. ( (...) )] in order to improve readability of grouped predicates.
    { (a >= b || (0 < b && 0 < a)) && (a % b == 0 || a < b || (0 < b && 0 < a)) } // By law [A.09]
    { ((a >= b || (0 < b && 0 < a)) && true && ((a % b == 0 || a < b) || (0 < b && 0 < a)) } // Since a % b ==> a == n*b ==> gcd(a, b) == gcd(n*b, b) == gcd(b, b) == b (n*b and b share a gcd of b). Then apply gcd property ii.
```

```
    { (a >= b || (0 < b && 0 < a)) && (a % b == 0 ==> gcd(a, b) == b) && ((a % b == 0 || a < b) || 0 < b && 0 < a) } // By law [A.22]
    { (a < b ==> (0 < b && 0 < a)) && (a % b == 0 ==> gcd(a, b) == b) && ((a % b != 0 && a >= b) ==> 0 < b && 0 < a) }
    if a < b {
        { 0 < b && 0 < a } // By law [A.09]
        { 0 < b && 0 < a && true } // By gcd property iii.
        { 0 < b && 0 < a && (gcd(b, a) == gcd(a, b)) } // By law [A.56]
        { 0 < b && 0 < a && forall r' :: (r' == gcd(b, a)) ==> r' == gcd(a, b) } // By law [A.74]
        { (forall r :: 0 < b && 0 < a) && forall r' :: (r' == gcd(b, a)) ==> r' == gcd(a, b) }
        r := GCD1(b, a);
        {r == gcd(a, b)}
    } else if (a % b == 0) {
        {b == gcd(a, b)}
        r := b;
        {r == gcd(a, b)}
    } else {
        { 0 < b && 0 < a } // By law [A.09]
        { 0 < b && 0 < a && true } // By gcd property iv.
        { 0 < b && 0 < a && (gcd(b, a % b) == gcd(a, b)) } // By law [A.56]
        { 0 < b && 0 < (a % b) && forall r' :: (r' == gcd(b, a % b)) && b > 0 ==> r' == gcd(a, b) } // By law [A.74]
        { (forall r :: 0 < b && 0 < (a % b)) && forall r' :: (r' == gcd(b, a % b)) && b > 0 ==> r' == gcd(a, b) }
        r := GCD1(b, a % b);
        { r == gcd(a,b) }
    }
    { r == gcd(a,b) }
}

method GCD2(a: int, b: int) returns (r: int)
    requires 0 <= a && 0 <= b
    ensures r == gcd(a, b)
    decreases b
{
    // The method is totally correct as the given precondition (0 <= a && 0 <= b) implies the weakest precondition.
    { (b == 0 || (0 < b && 0 <= a)) } // By law [A.1]
    // By the definition of the modulo operator in order for a % b to be >= 0,
    // b must be positive and a must be non-negative.
    { (b == 0 || (0 < b && 0 <= a && 0 < b)) }
    { (b == 0 || 0 < b && 0 <= (a % b)) } // By law [A.22]
    { true && (b != 0 ==> 0 < b && 0 <= (a % b)) } // By gcd property i.
    { (a == gcd(a, 0)) && (b != 0 ==> 0 < b && 0 <= (a % b)) } // By substitution
    { (b == 0 ==> a == gcd(a, b)) && (b != 0 ==> 0 < b && 0 <= (a % b)) }
    if b == 0 {
        { a == gcd(a,b) }
        r := a;
        { r == gcd(a,b) }
    } else {
        { 0 < b && 0 <= (a % b) } // By law [A.09]
        { 0 < b && 0 <= (a % b) && true} // By gcd property iv. and since b != 0 by else clause.
        { 0 <= b && 0 <= (a % b) && (gcd(b, a % b) == gcd (a, b))} // By law [A.74] and law [A.56]
        { (forall r :: 0 <= b && 0 <= (a % b)) && forall r' :: (r' == gcd(b, a % b)) && b > 0 ==> r' == gcd(a, b) }
        r := GCD2(b, a % b);
```

```
        { r == gcd(a,b) }
    }
    { r == gcd(a,b) }
}

/* Conclusion

As seen by the above specifications and formal (weakest precondition) proofs of total correctness, both colleagues are right.
*/
```