George Harrison HW3

7.1

```
        // Use Euclid's algorithm to calculate the GCD.
        // The algorithm finds the GCD of two integers by repeatedly taking the remainder of the
division.
        private long GCD( long a, long b )
        {
           a = Math.abs( a );
           b = Math.abs( b );

           for( ; ; )
           {
              long remainder = a % b;
              If( remainder == 0 ) return b;
              a = b;
              b = remainder;
           };
        }
```

7.2

The two conditions for writing bad comments are either writing comments as you code or writing all the code without comments then going back and adding them. The problem is that these methods involve you explaining what the code literally does instead of a more higher level comment explaining what the code should do.

7.4

You could apply offensive programming by checking that the inputs are non-negative, as well as non-zero

```
// Use Euclid's algorithm to calculate the GCD.
// The algorithm finds the GCD of two integers by repeatedly taking the remainder of the
division.
private long GCD(long a, long b)
{
   Debug.Assert(a >= 0 && b >= 0);
   checked
   {
      if (a == 0) return b;
      if (b == 0) return a;

      a = Math.Abs(a);
      b = Math.Abs(b);

      while (true)
      {
         long remainder = a % b;
         if (remainder == 0) return b;
         a = b;
         b = remainder;
      }
   }
}
```

7.5
Yes you should add an error handler, otherwise the program will crash

7.7

Drive to nearest supermarket
- Get in the car
- Get directions to the nearest supermarket
- Start driving, following directions to the supermarket
- When you've arrived, park

Assumptions
- You have a car and license
- You are capable of driving safely
- You have a phone or some device with a map
- The supermarket is open
- You have enough gas in the car

8.1

```
public static void Main(string[] args)
   {
      TestIsRelativelyPrime(21, 35); // Not relatively prime
       TestIsRelativelyPrime(21, 22); // Relatively prime
       TestIsRelativelyPrime(-1, 100); // Relatively prime
       TestIsRelativelyPrime(-1, 0); // Relatively prime
       TestIsRelativelyPrime(1000001, 3); // out of range error
   }
private bool TestIsRelativelyPrime(int a, int b)
{
  if (a < -1000000 || a > 1000000 || b < -1000000 || b > 1000000)
   {
       Console.WriteLine("out of range");
       return false;
   }

  if (IsRelativelyPrime(a, b))
   {
         Console.WriteLine($"{a} and {b} are relatively prime”);
         return true;
   }
   else
   {
         Console.WriteLine($"{a} and {b} are not relatively prime”);
         return false;
   }
}
```

8.3

This is a black-box test because it does not use the internal implementation of the method, and only focuses on the output and input behavior. Black-box testing is useful to test the functionality from the perspective of the end-user. Exhaustive testing is most comprehensive, however it is not always feasible to test every possible combination of inputs. It is best used on smaller or simpler systems. In this case it's probably not an option since there would be too many integers to test. Grey-box and white-box could be used if we had some knowledge of the internal structure of the method.

8.5
The code worked for all the test cases I tested. I just had to make all the methods static for them to run properly. Testing the code helped to see that the code worked as expected for the few test cases I had. It would probably be more beneficial if I had a more comprehensive list of test cases.

8.9
Exhaustive testing has to be black box, because you can't know the internal details of how the code works. Exhaustive testing must cover every possible input to the code. This is not possible if you are doing white box testing, because you would be limited by the conditions known from the internal implementation.

8.11
Using the lincoln index, L = (5 * 4 * 5) / 5 = 20. This means there are 20 bugs total, and about 6 remaining to be found.

8.12
If there are no bugs in common then the denominator will be 0, leading to an undefined result. This means there is probably a lot of bugs that have not been found. The lower bound would have to at least be the amount of total bugs found by both testers.