Mitchell Simon Homework 3

*Problem 7.1*

The comments were pretty much useless, more useful comments would have explained the concept of the algorithm (hence linking to the wiki).

The following shows an example of the improved version of the previous code:

       // Use Euclid's algorithm to calculate the GCD (see <https://en.wikipedia.org/wiki/Euclidean_algorighm> for more details on Euclid’s algorithm)  
         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;  
            };  
         }

*Problem 7.2*

Bad comments happen either when the programmer writes comments after fully writing the code, which leads to comments that just describe the line they’re on rather than giving an idea of how the method works, or they write comments while they’re coding that have superfluous detail and end up confusing people that try to read it.

*Problem 7.4*

Offensive programming would validate the inputs and the result by asserting them as greater than 0. A debug assert would throw an exception if it sees a problem.

*Problem 7.5*

Error handling is there to catch anything that might crash the program, so it would always be best to have some form of error handling.

*Problem 7.7*

1. Go to garage
2. Find car
3. Open door
4. Get in car’s driver seat
5. Turn on car engine
6. Drive forwards and take a right
7. Take another right
8. Open garage door
9. Continue straight then turn right
10. Turn left onto La Tijera and stop at the stop sign
11. Turn right and then take another right at Loyola Blvd.
12. Turn left into Ralphs’ parking lot
13. Find parking spot
14. Turn off the car engine and get out
15. Lock car and go to supermarket

*Problem 8.1*

private bool Validate\_AreRelativelyPrime(int a, int b)

{

// Use positive values.

a = Math.Abs(a);

b = Math.Abs(b);

// If either value is 1, return true.

if ((a == 1) || (b == 1)) return true;

// If either value is 0, return false.

// (Only 1 and -1 are relatively prime to 0.)

if ((a == 0) || (b == 0)) return false;

// Loop from 2 to the smaller of a and b looking for factors.

int min = Math.Min(a, b);

for (int factor = 2; factor <= min; factor++)

{

if ((a % factor == 0) && (b % factor == 0)) return false;

}

return true;

}

*Problem 8.3*

It’s a blackbox test because we don’t know how the method works. If we were given how it works we could write white box and grey box tests for it. Writing an exhaustive test ranging from negative 1 million to positive 1 million would be a lot.

*Problem 8.5*

The testing code did not consider how robust the program could be, and therefore didn’t test for weird values like -1, 0, and 1.

*Problem 8.9*

Exhaustive tests are black-box tests as they don’t need to know how the method works to test what the expected outcome should be.

*Problem 8.11*

You can use the pairs of programmers to find the Lincoln index.

Alice and Bob: (5x4)/2 = 10

Alice and Carmen: (5x5)/2 = 12.5

Bob and Carmen: (4x5)/1 = 20

The average of all three Lincoln estimates is 14, but there are still 4 bugs unaccounted for.

*Problem 8.12*

If the testers don’t find any bugs in common, that means a division by 0, which means you have no idea how many errors. You can calculate a semi-accurate lower bound by assuming they all have 1 in common however.