Problem 3

1. testThrowsIllegalArgument failed because instead of checking for only negative numbers ($n < 0$), getFibTerm was checking for numbers less than 1 ($n \leq 0$). I fixed this by changing $n \leq 0$ to $n < 0$.

2. testBaseCase failed because when $n = 0$, 0 should be returned. But since an exception is thrown when $n = 0$, the test fails. After fixing the error from testThrowsIllegalArgument, testBaseCase worked fine, so I didn't have to change anything else.

3. testInductiveCase failed because n was returned when $n \leq 2$, but 2 shouldn't be returned when $n = 2$. To fix this, I changed the check for the base cases from $n \leq 2$ to $n < 2$. Then, the terms after that wasn't being calculated correctly because instead of getting the previous two terms and adding them, the $(n + 1)$th term and the $(n - 2)$th term was being subtracted. To fix this, I changed it to add the $(n - 1)$th and $(n - 2)$th terms.

4. testLargeN failed because the execution was taking too long. The given implementation of Fibonacci was an exponential time algorithm, so I had to come up with a different recursive algorithm that would be more efficient. I remembered an $O(n)$ algorithm that we went over in Intro to Algos that calculated Fibonacci starting from $n = 1$ and only calculating every term exactly once, so I tried to implement that recursively. I made a helper function that getFibTerm called and this worked fine for testInductiveCase, but returned the wrong answer for testLargeN. I checked what the max int size was and saw that the desired answer was larger than that, so I changed the datatypes of some parameters and the return type to a long.

5. It took too long for testLargeN to run because the algorithm had exponential running time (every term needs to calculate the previous 2 terms which are unknown in its context, leading to a pretty bad running time). To fix this, I had to come up with a way to store the previous 2 terms while still keeping the implementation recursive. I did that by passing in the previous 2 terms as arguments to the helper function, getting rid of the need to constantly recalculate the same term.