HW0 Problem 3: Answering Questions About the Code

Jordan Ye

January 18, 2025

(1) **Why did Fibonacci fail the testThrowsIllegalArgumentException test? What did you have to do to fix it?**

Answer: The Fibonacci class originally threw an IllegalArgumentException for any input less than or equal to zero, which included the value 0. However, according to the tests, only negative numbers should cause an exception. In other words, getFibTerm(0) was expected to return 0. To correct this, I changed the condition so that the exception is only thrown when n is less than 0, thereby allowing 0 to return 0 as defined in the Fibonacci sequence.

(2) **Why did Fibonacci fail the testBaseCase test? What (if anything) did you have to do to fix it?**

Answer: The testBaseCase expected that getFibTerm(0) would return 0 and getFibTerm(1) would return 1. Initially, the Fibonacci class returned 1 for both n = 0 and n = 1 because it treated any n less than or equal to 2 as a base case. I fixed this by explicitly handling the case where n is 0, returning 0, and ensuring that for n = 1, the method returns 1, which aligns with the traditional Fibonacci sequence.

(3) **Why did Fibonacci fail the testInductiveCase test? What (if anything) did you have to do to fix it?**

Answer: The inductive case failed because the recursive logic in the Fibonacci class was implemented incorrectly. Instead of summing the two previous terms (getFibTerm(n-1) plus getFibTerm(n-2)), the code mistakenly subtracted one term from the other. I corrected this by updating the recursive call so that it properly adds the two preceding Fibonacci numbers, which results in the correct sequence.

(4) **Why did Fibonacci fail the testLargeN test? What (if anything) did you have to do to fix it?**

Answer: The testLargeN failed primarily because the naive recursive implementation had exponential time complexity. This made it extremely inefficient for larger values of n (like n = 60). In addition, the method

returned an int, which was too small for large Fibonacci numbers such as F(60) = 1548008755920. To resolve this, I changed the return type from int to long and added memoization using a HashMap to cache computed values. This optimization significantly improved the performance and allowed the method to handle large inputs.

(5) **What was causing Fibonacci to be so slow on testLargeN? What did you do to make Fibonacci faster while still preserving the recursive nature of your implementation?**

Answer: The slow performance in the testLargeN was due to the recursive approach, which involved recalculating the same Fibonacci numbers multiple times—resulting in exponential growth in the number of computations. To enhance the speed while keeping the recursive approach, I implemented memoization with a HashMap. This technique stores previously calculated Fibonacci numbers and reuses them in subsequent calls, thereby reducing the time complexity to linear and allowing the method to efficiently process large values of n.