Comparative Analysis of Recursion and Iteration

Ethan J. Nephew

Documentation Analysis

The purpose of this document is to compare the system time to complete an iterative Fibonacci sequence compared to a recursive Fibonacci sequence. Below, in Figure 1.1, is a chart that shows the time in nanoseconds to produce the Fibonacci results. What can be observed is that the raw time required to complete the recursive method appears to boarder on an exponential growth pattern.

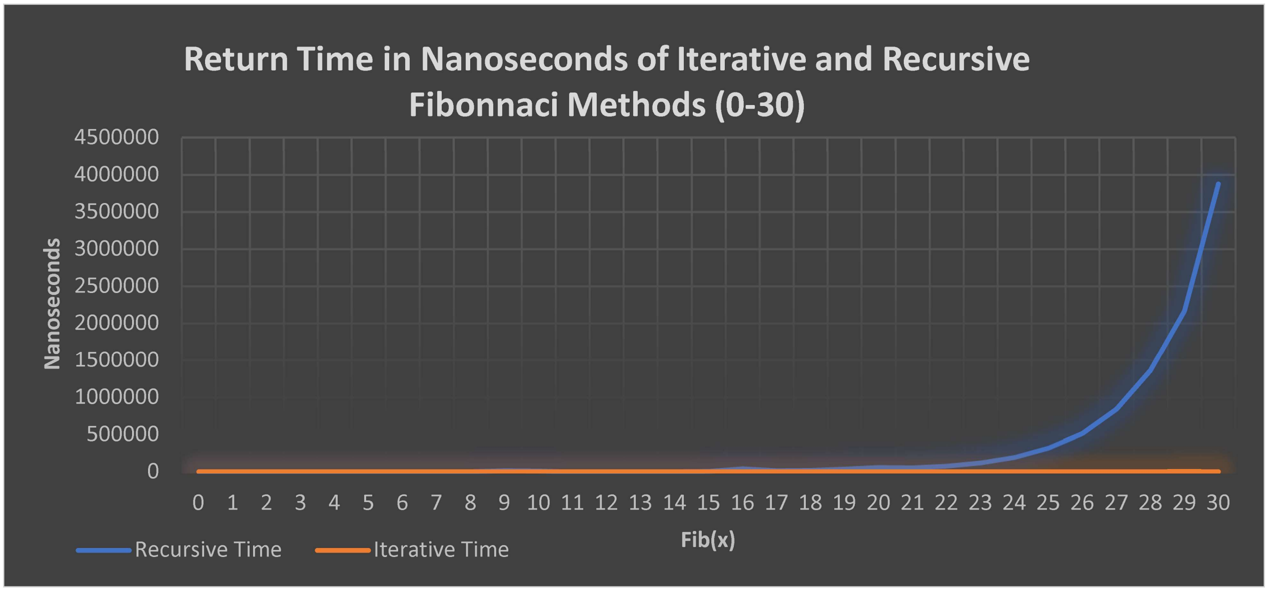

Figure 1.1 Comparative chart of iterative and recursive return times in nanoseconds.

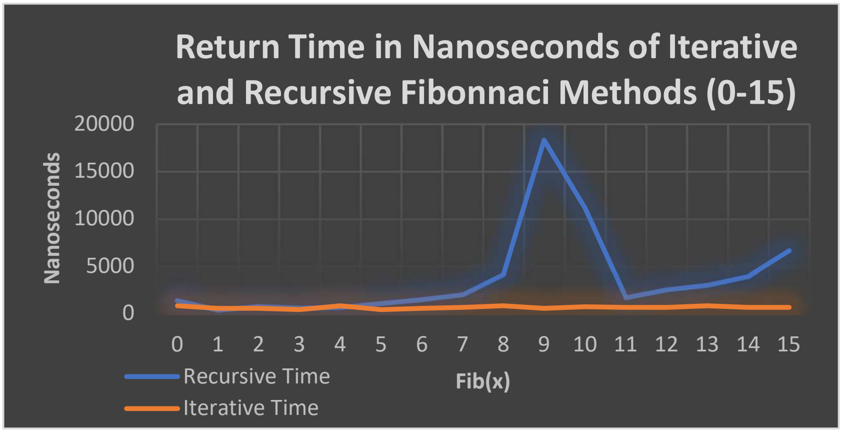

Figure 1.2 The first 0-15 return times, this is a broken-down segment from Figure 1.1. At Fib(9), there appears to be an example of time turbulence.

At this scale (Figure 1.1), it can appear as though there is no difference in time efficiency at smaller recursive calls. While it is true that the ratio to resolve the recursive method call is less significant, as shown in Figure 1.3, there is still a significant difference between time to return the recursive call, in comparison to the iterative method call. The difference in return time is less stark when calculating less significant numbers in the Fibonacci sequence. The recursive method call seems to be suspectible to what I would describe as "time turbulence" as shown in Figure 1.2. At times my system appears to take a non-uniformly greater amount of time for recursive Fibonacci method calls.

While it is nanoseconds, which is a relatively trivial amount of time, if we can be more efficient, we probably should be. At smaller Fib(x) calls there is not a huge difference, but the iterative method is many times more time efficient than the recursive function. In a consecutive Fib(0) to Fib(30) sequence, the iterative method returns the result 157 times faster, on average, compared to the recursive method. From a statistical perspective the trendline could be described as exponential. As the program progresses higher and higher through the Fibonacci sequence, the recursive method becomes exponentially less efficient. Why is this? The Fibonacci sequence is by its nature a recursive sequence, so calculating it recursively would seem to be the best solution. It sounds intuitive. It sounds logical, but this is clearly not the case.

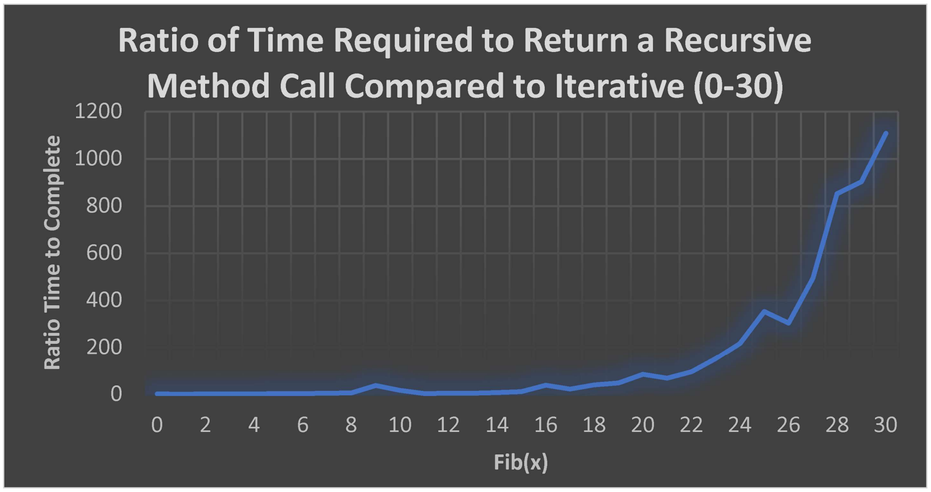

Figure 1.3 Ratio of iterative return times compared to recursive return times. The recursive method is on average 157 times slower to return the Fibonacci number.

I was curious about how many methods calls it takes to complete a recursive method call. So, I implemented a static variable to serve as a counter for the recursive method calls. While the iterative method needs to only run one more time for every parameter increase, the recursive method will need to run many more times. One could say, exponentially more times. I created a method that will output the number of times the recursive Fibonacci method was used. At Fib(30), the number of recursive method calls is 2.7 million. So, to return the answer it took 2.7 million method calls. That is a considerable amount of work and vastly more work than the iterative method requires.

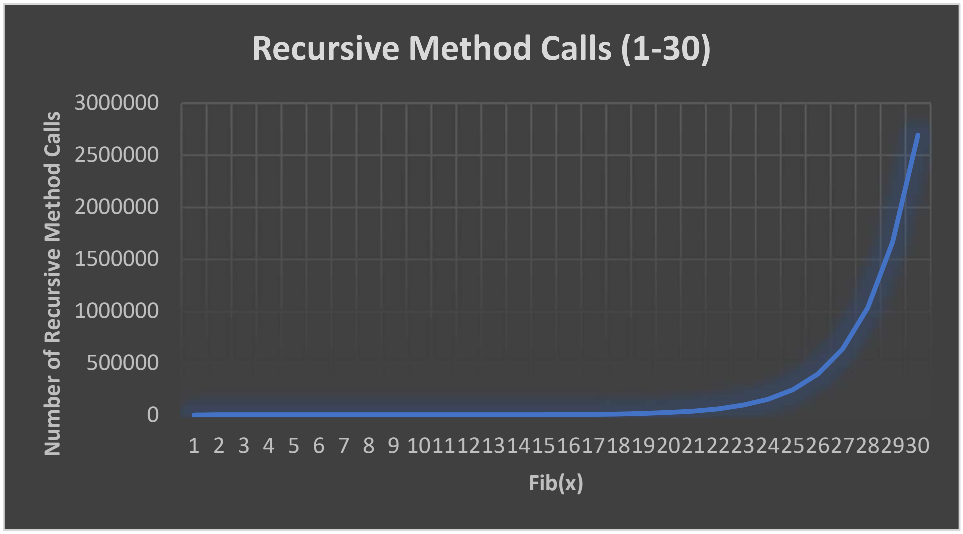

Figure 2.1 This is the raw number of times the recursive method must call itself to return the Fibonacci number. At Fib(5) it requires 25 method calls. At Fib(20) it requires 13529 method calls. At Fib(30) it requires 2.7 million method calls.

A Point of Interest

When I was looking at the amount of recursive method calls, I noticed a pattern in the number of recursive method calls. If one were to divide a Fibonacci number by its previous number and continue along the sequence, they will forever more precisely approach the Golden Ratio. Interestingly, doing this begins with a degree of volatility. I made a chart to illustrate this in Figure 3.1. It begins with volatility, but as you continue calculating the ratio it quickly stabilizes to approximately 1.618. This is the approximation of the Golden Ratio. Now for the fascinating part, this is what the ratio of recursive method calls looks like in Figure 3.2. The ratio of recursive method calls approaches the Golden Ratio just as the ratio of the Fibonacci sequence does. While it does make perfect sense that this would be the case, I think it is interesting and cool.

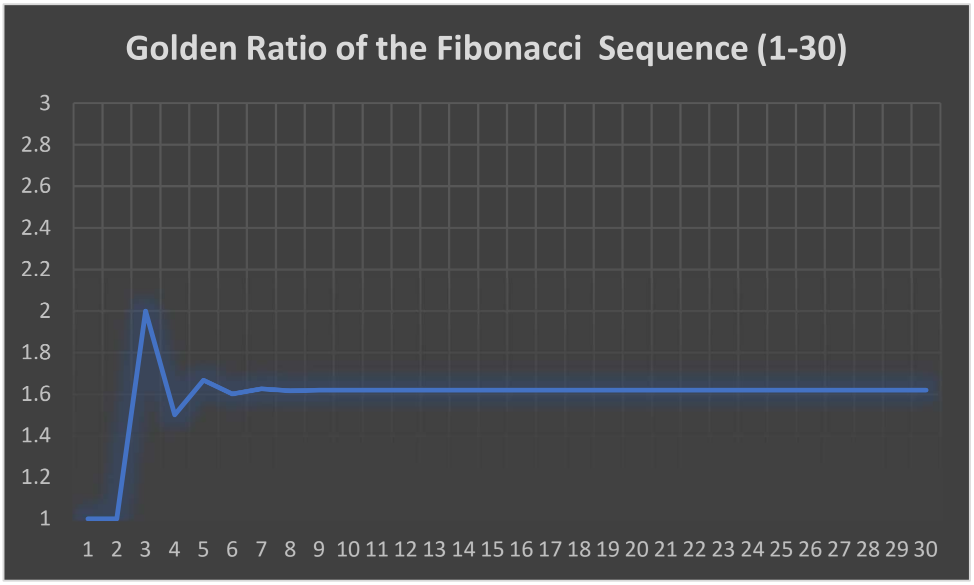

Figure 3.1 The Golden Ratio of the Fibonacci sequence.

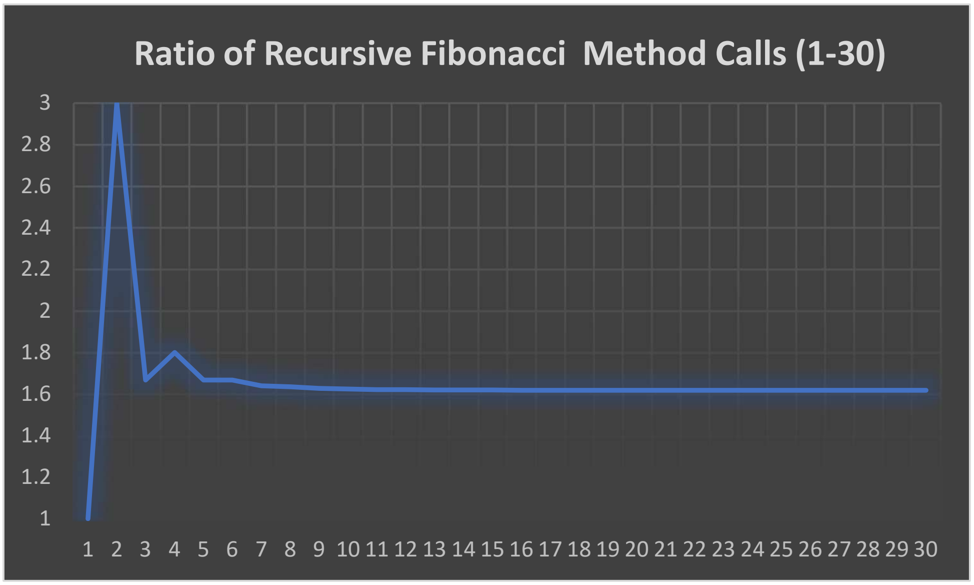

Figure 3.2 The ratio of recursive method calls that Java uses to calculate the next number in the Fibonacci Sequence.