**Fibonacci Sequence Comparison**

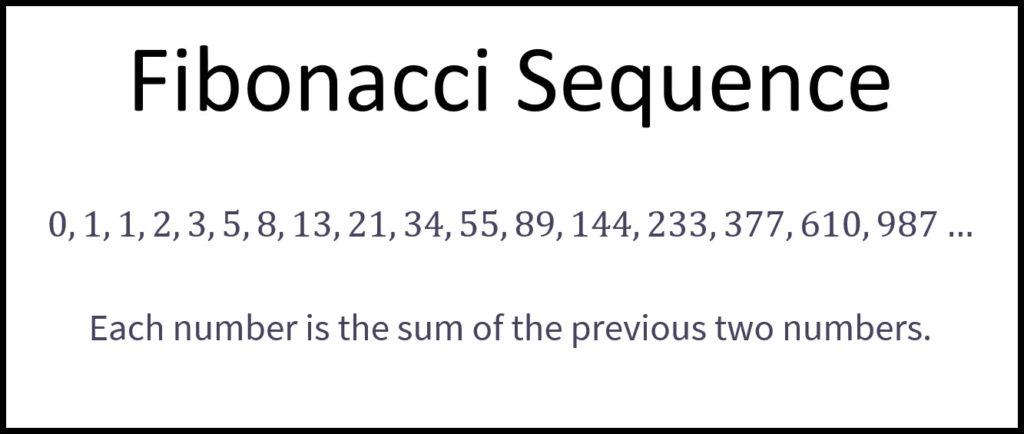
Jesus Sanchez-Rivera

1. **Objective**

The program’s design is to implement the calculation of Fibonacci sequences using iterative and recursive methods. Both implementations have different advantages and disadvantages and the program will through empirical data show the efficiency of each method.

* 1. **Fibonacci Sequence**

The Fibonacci sequence is described as the sum of two previous terms. Figure 1 represents the result of previous sums to the corresponding location in the sum.



**Figure 1**: The first term, Fibonacci[0] is equal to 0, fib[1] is equal to 1, fib[3] is equal to 3 and so on.

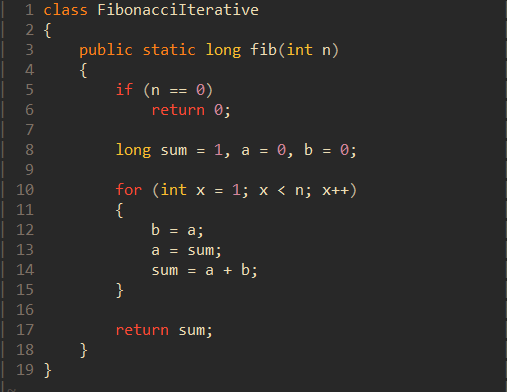
1. **Steps to solve the problem**

The formula used to calculate any value in the sequence is: 

This formula forms the basis of the iterative solution, where Fn represents the sum of the previous two values while becoming one of the two terms to calculate the next value in the sequence. The recursive solution afterwards can be calculated from the creation of the iterative method.

1. **Iterative Method**

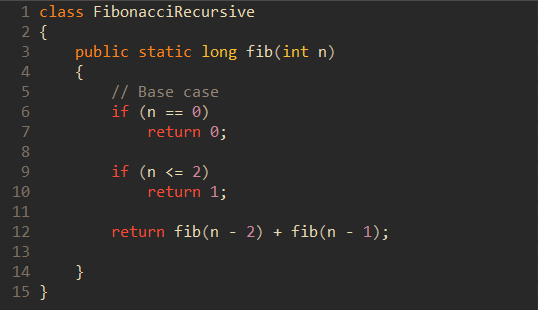
The variable n represents the sequence location or how many steps are required to calculate the value. Solving the Fibonacci sequence iterative requires a single for loop since the amount of steps it will take are pre-determined by the sequence number we are solving for. Figure 2 shows the java solution for the iterative method.



**Figure 2**: The sum is equivalent to Fn, a represents Fn-1 and b is Fn-2.

1. **Recursive Method**

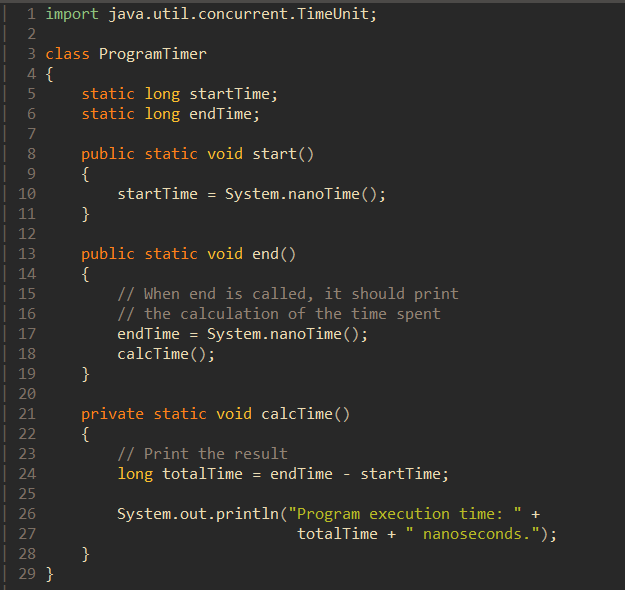
The formula to calculate the Fibonacci value in a sequence can be seen easier on the recursive method shown in Figure 3. The solution for the recursive method is a top, down then back up as it begins to calculate from the nth sequence all the way down to sequence 0, then sum all the results back to find the final result for Fn.



**Figure 3**: The base cases are used to prevent the recursion method from infinitely calling itself, the step where it calls the same function is referred to as the recursive step.

1. **Calculate the time for each method**

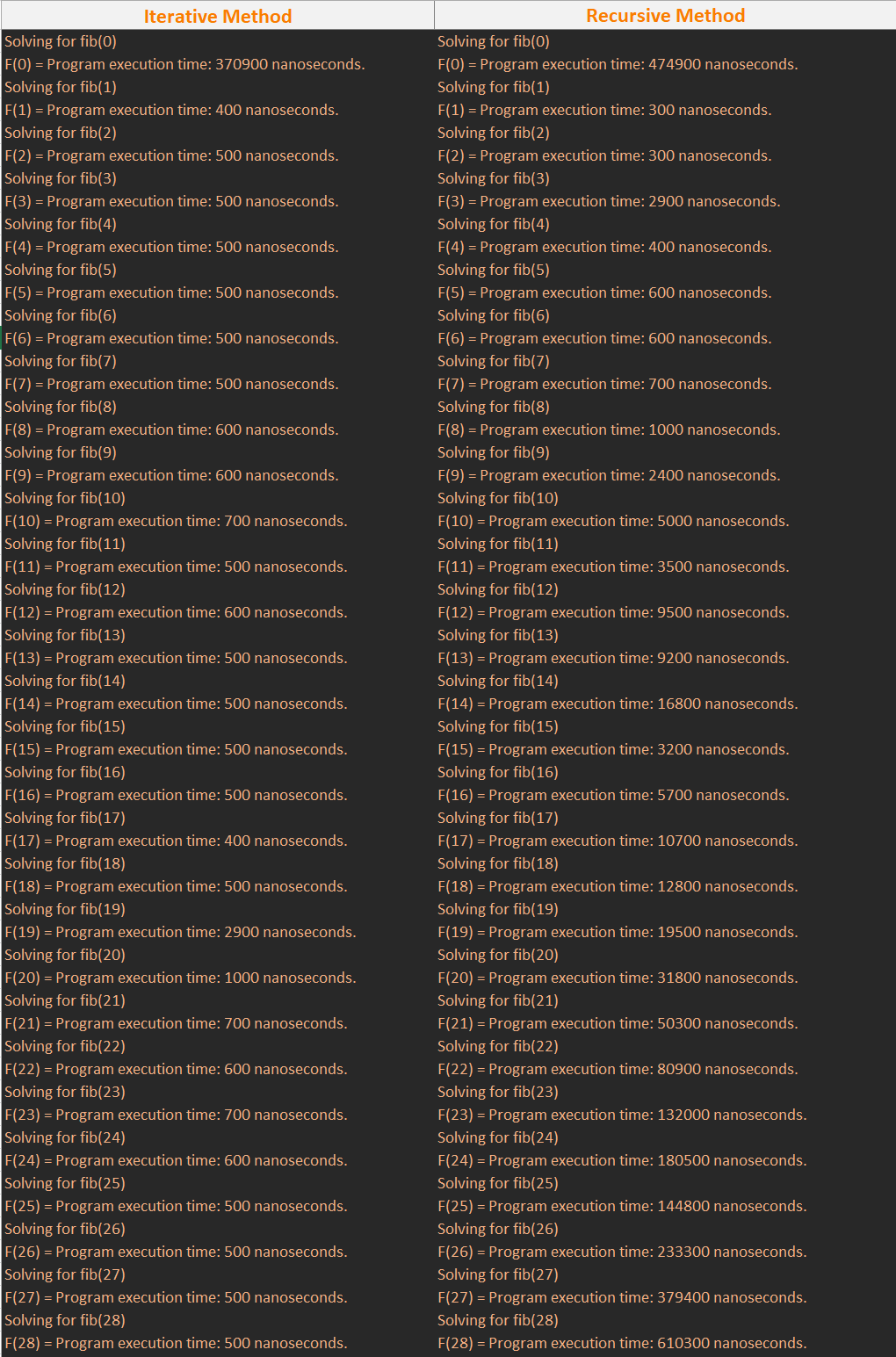
To display the difference between the methods, Java’s System.nanoTime() function can be used to calculate how long the program spends finding the Fibonacci sequence. Using several cases up to the same number of sequences for each method will empirically show whether one method is more efficient over another. Instead of creating the step to calculate how much time is spent within each method, taking advantage of Java’s multi class design allows for the creation of a separate class that is can be called whenever the calculations are taking place for each Fibonacci sequence, as shown in Figure 4.

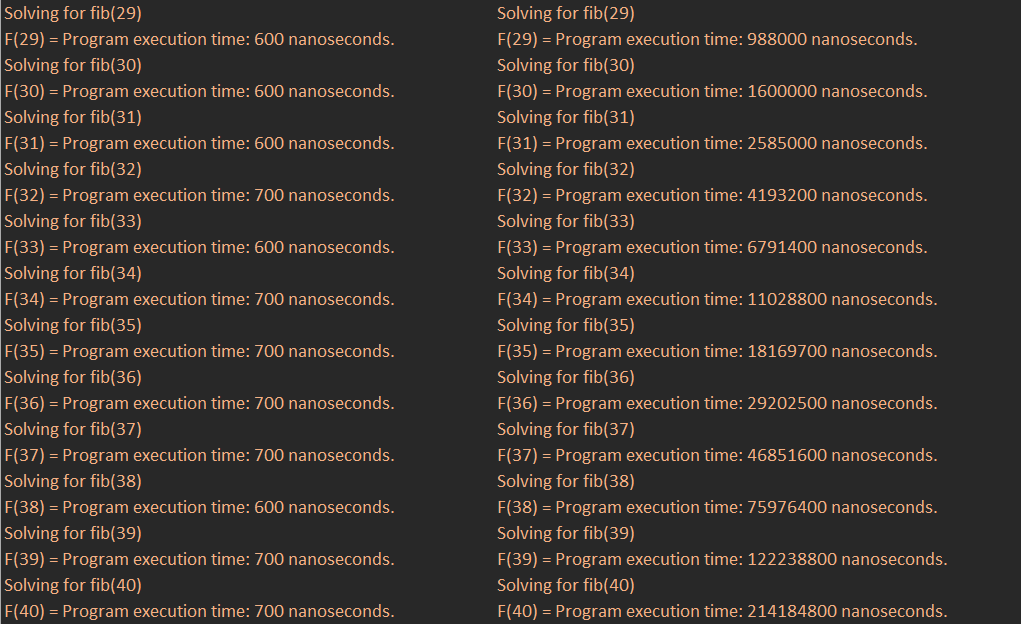


**Figure 4**: When the function end() is called by another class, it will print the resulting time difference between start and end time.

1. **Findings**

When calculating the time to solve each Fibonacci sequence up to sequence 40, there is a clear distinction between the times of the iterative method compared to the recursive approach. The time to calculate for the iterative method shows approximately a linear increase compared to the recursive approach, which at first appears to expand exponentially. Upon further research, when calculating for the Big O, or the time complexity, of the recursive approach, the result is O(1.6180)2. The results of the time difference using the time calculator class is shown in Figure 5.

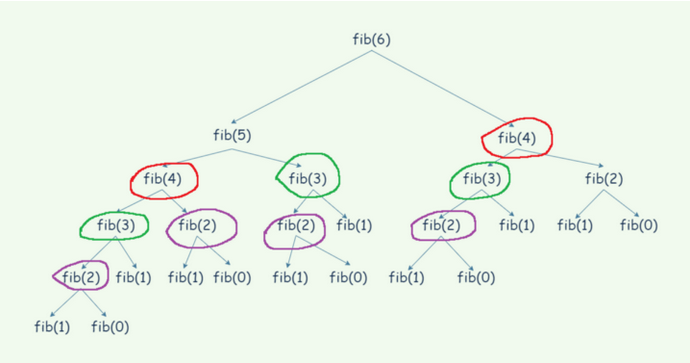




**Figure 5**: Taking into account compilation inconsistency, the iterative approach takes less time than the recursive method.

1. **Result of Program**

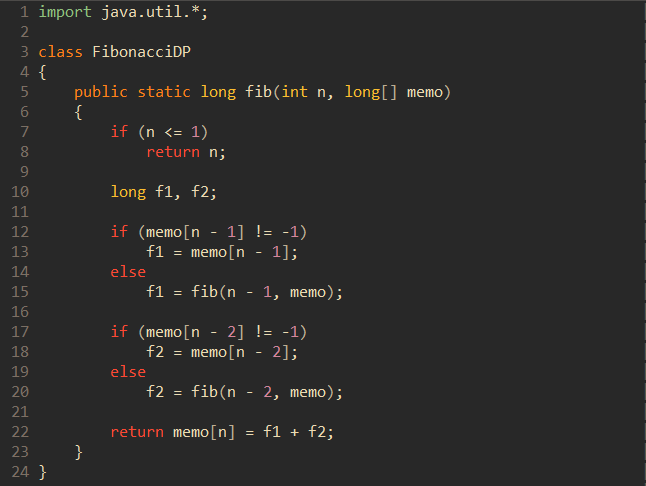
The reason for the time large time difference between both methods is due to the nature of the recursive approach. Figure 6 shows clearly what is happening underneath the program’s execution. There are many steps being calculated over again while reaching the nth term of the Fibonacci sequence, on top of the fact that every recursive call is requiring the program to allocate more resources. Every recursive call has to in order return a value after calculating the result of a Fibonacci sequence.



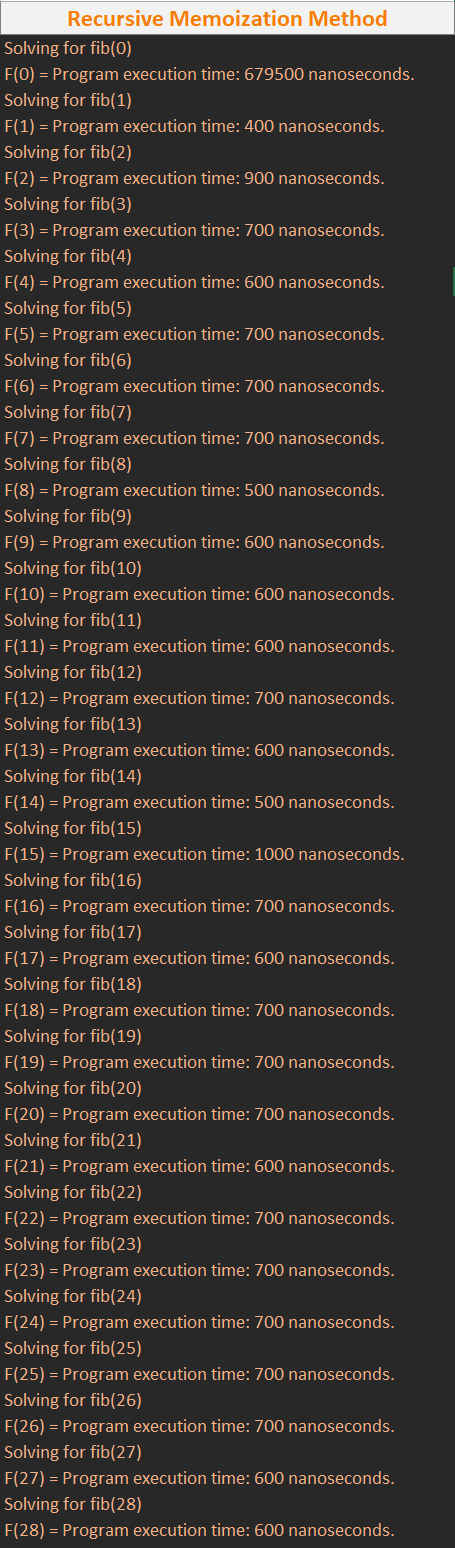
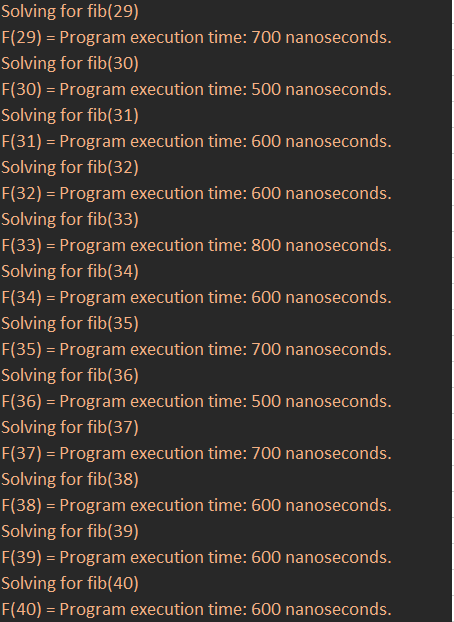
**Figure 6**: Each step circled is a sequence being calculated multiple times, using up program resources and time.

1. **Different Solutions to Recursion**

A method of improving the recursive method is to implement memoization, or storing each Fibonacci sequence result as you calculate for the different steps during the recursive calls. The program looks similar to the recursive as shown in Figure 7.This type of approach is referred to as Dynamic Programming, and increase the efficiency of the recursive method to match nearly the speed of the iterative step as shown in Figure 8.



**Figure 7**: The memoization part is storing the result of the Fibonacci sequence in an array to use later. When encountering the same step, the program will check if there is already a value not equal to the default set at -1.



**Figure 8**: The time of execution now matches the iterative solution.

**Sources**

* <https://www.geeksforgeeks.org/program-for-nth-fibonacci-number/>
* <https://andymath.com/fibonacci-sequence/>
* <https://www.geeksforgeeks.org/time-complexity-recursive-fibonacci-program/>
* <https://www.geeksforgeeks.org/memoization-1d-2d-and-3d/>