Fibonacci Sequence Comparison

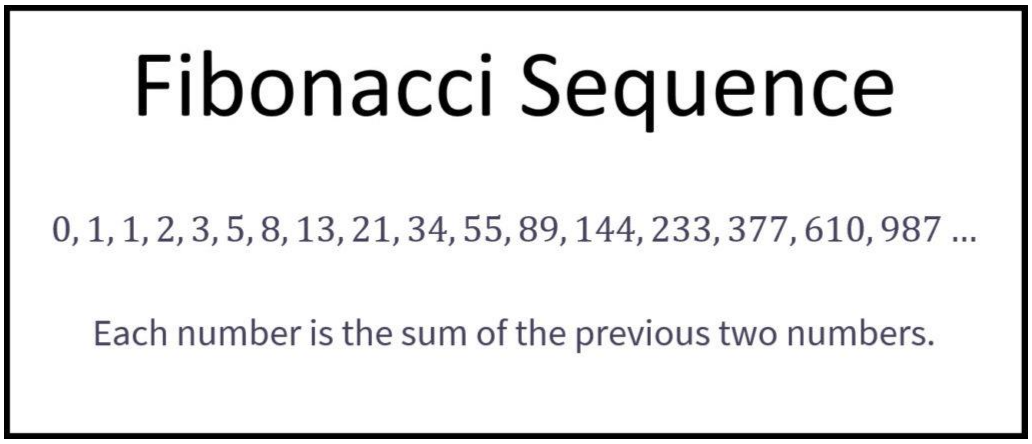
Jesus Sanchez-Rivera

1.0 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.

1.5 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.



Fibonacci Sequence

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987 ...

Each number is the sum of the previous two numbers.

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.

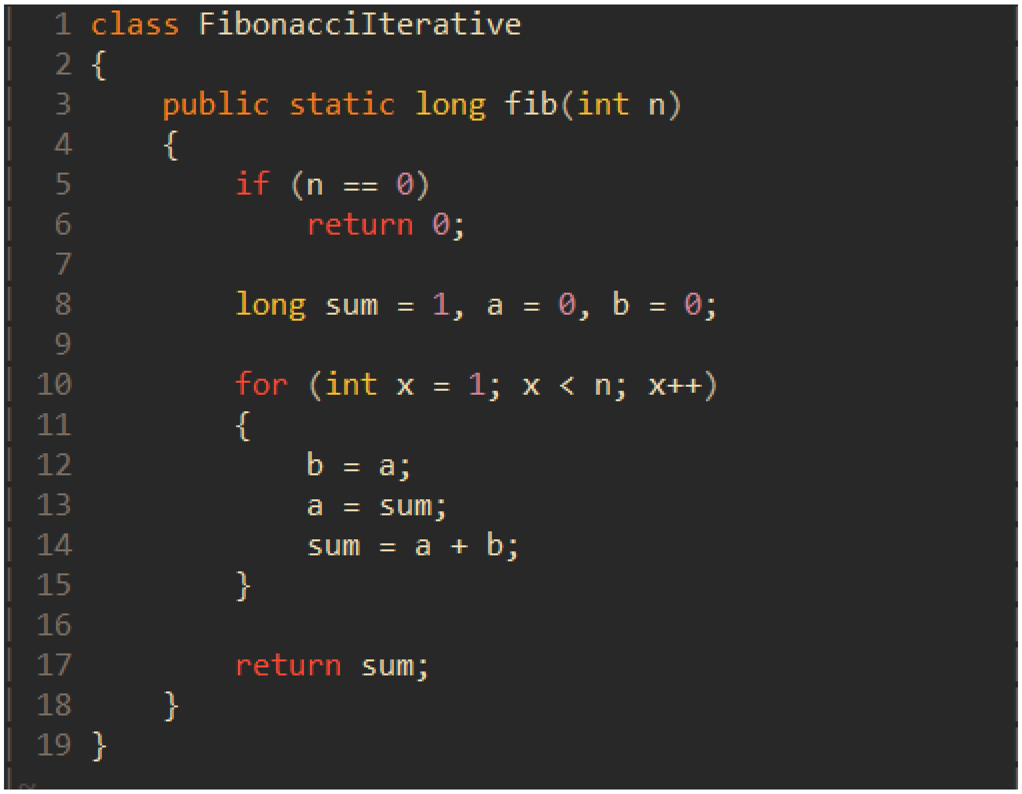
2.0 Steps to solve the problem

The formula used to calculate any value in the sequence is: $F_n = F_{n-1} + F_{n-2}$

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.

3.0 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.

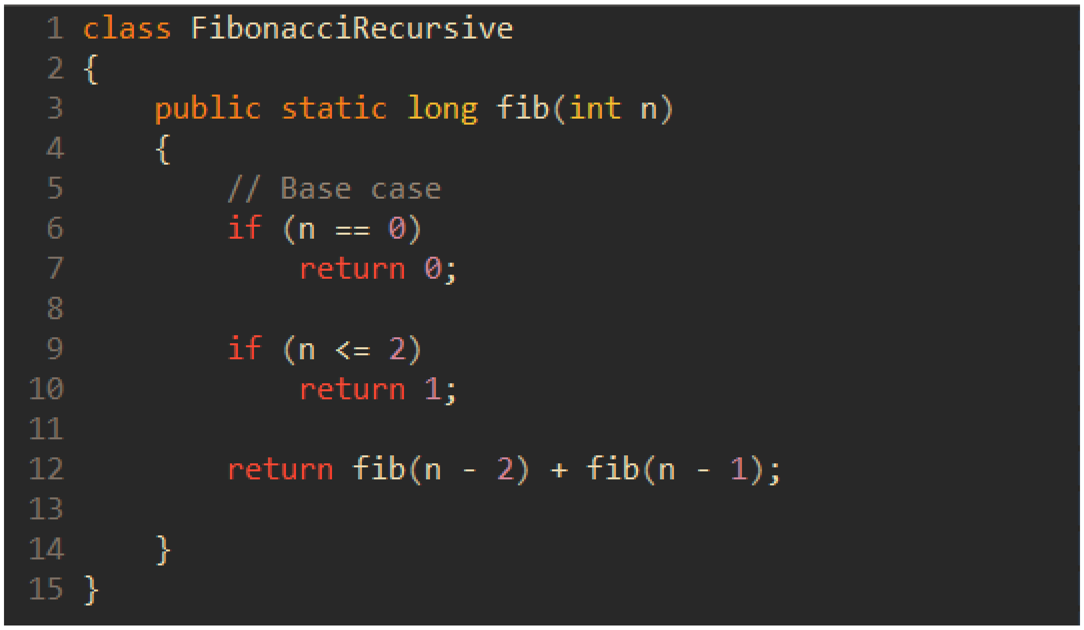


```java
class FibonacciIterative
{
    public static long fib(int n)
    {
        if (n == 0)
            return 0;

        long sum = 1, a = 0, b = 0;

        for (int x = 1; x < n; x++)
        {
            b = a;
            a = sum;
            sum = a + b;
        }

        return sum;
    }
}
```

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

4.0 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.

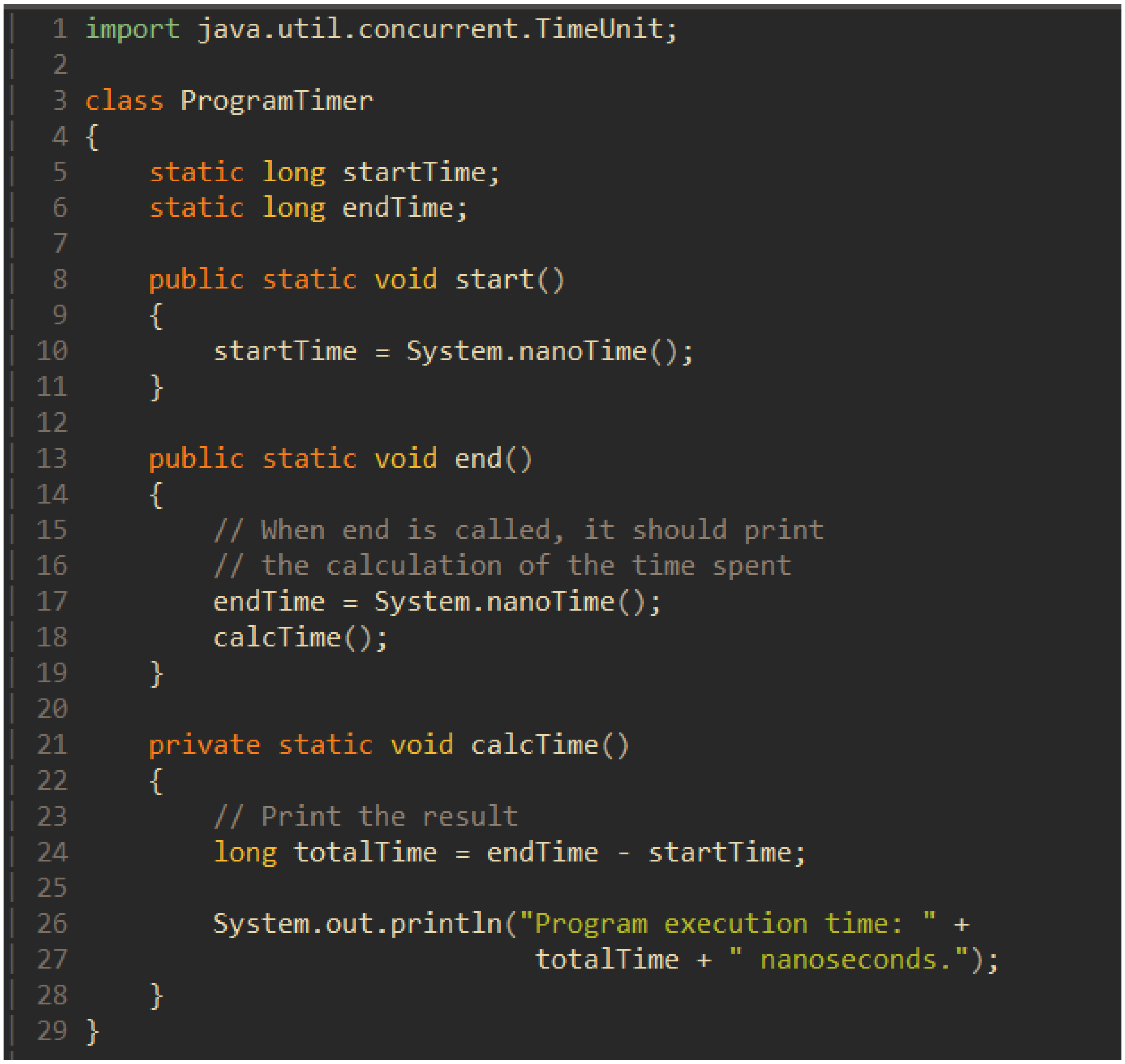


```
class FibonacciRecursive
{
    public static long fib(int n)
    {
        // Base case
        if (n == 0)
            return 0;

        if (n <= 2)
            return 1;

        return fib(n - 2) + fib(n - 1);

    }
}
```

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.

5.0 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.



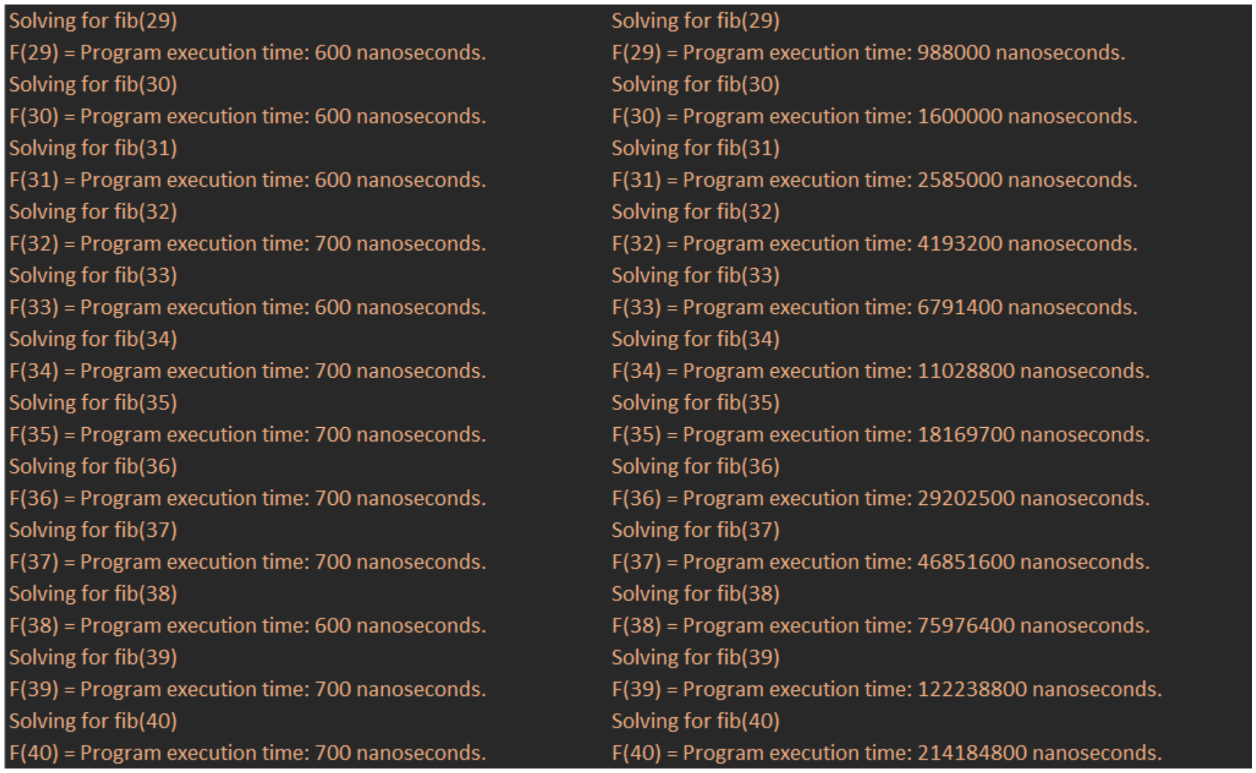
```java
import java.util.concurrent.TimeUnit;

class ProgramTimer
{
    static long startTime;
    static long endTime;

    public static void start()
    {
        startTime = System.nanoTime();
    }

    public static void end()
    {
        // When end is called, it should print
        // the calculation of the time spent
        endTime = System.nanoTime();
        calcTime();
    }

    private static void calcTime()
    {
        // Print the result
        long totalTime = endTime - startTime;

        System.out.println("Program execution time: " +
                           totalTime + " nanoseconds.");
    }
}
```

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

6.0 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.

Iterative Method	Recursive Method
Solving for fib(0)	Solving for fib(0)
F(0) = Program execution time: 370900 nanoseconds.	F(0) = Program execution time: 474900 nanoseconds.
Solving for fib(1)	Solving for fib(1)
F(1) = Program execution time: 400 nanoseconds.	F(1) = Program execution time: 300 nanoseconds.
Solving for fib(2)	Solving for fib(2)
F(2) = Program execution time: 500 nanoseconds.	F(2) = Program execution time: 300 nanoseconds.
Solving for fib(3)	Solving for fib(3)
F(3) = Program execution time: 500 nanoseconds.	F(3) = Program execution time: 2900 nanoseconds.
Solving for fib(4)	Solving for fib(4)
F(4) = Program execution time: 500 nanoseconds.	F(4) = Program execution time: 400 nanoseconds.
Solving for fib(5)	Solving for fib(5)
F(5) = Program execution time: 500 nanoseconds.	F(5) = Program execution time: 600 nanoseconds.
Solving for fib(6)	Solving for fib(6)
F(6) = Program execution time: 500 nanoseconds.	F(6) = Program execution time: 600 nanoseconds.
Solving for fib(7)	Solving for fib(7)
F(7) = Program execution time: 500 nanoseconds.	F(7) = Program execution time: 700 nanoseconds.
Solving for fib(8)	Solving for fib(8)
F(8) = Program execution time: 600 nanoseconds.	F(8) = Program execution time: 1000 nanoseconds.
Solving for fib(9)	Solving for fib(9)
F(9) = Program execution time: 600 nanoseconds.	F(9) = Program execution time: 2400 nanoseconds.
Solving for fib(10)	Solving for fib(10)
F(10) = Program execution time: 700 nanoseconds.	F(10) = Program execution time: 5000 nanoseconds.
Solving for fib(11)	Solving for fib(11)
F(11) = Program execution time: 500 nanoseconds.	F(11) = Program execution time: 3500 nanoseconds.
Solving for fib(12)	Solving for fib(12)
F(12) = Program execution time: 600 nanoseconds.	F(12) = Program execution time: 9500 nanoseconds.
Solving for fib(13)	Solving for fib(13)
F(13) = Program execution time: 500 nanoseconds.	F(13) = Program execution time: 9200 nanoseconds.
Solving for fib(14)	Solving for fib(14)
F(14) = Program execution time: 500 nanoseconds.	F(14) = Program execution time: 16800 nanoseconds.
Solving for fib(15)	Solving for fib(15)
F(15) = Program execution time: 500 nanoseconds.	F(15) = Program execution time: 3200 nanoseconds.
Solving for fib(16)	Solving for fib(16)
F(16) = Program execution time: 500 nanoseconds.	F(16) = Program execution time: 5700 nanoseconds.
Solving for fib(17)	Solving for fib(17)
F(17) = Program execution time: 400 nanoseconds.	F(17) = Program execution time: 10700 nanoseconds.
Solving for fib(18)	Solving for fib(18)
F(18) = Program execution time: 500 nanoseconds.	F(18) = Program execution time: 12800 nanoseconds.
Solving for fib(19)	Solving for fib(19)
F(19) = Program execution time: 2900 nanoseconds.	F(19) = Program execution time: 19500 nanoseconds.
Solving for fib(20)	Solving for fib(20)
F(20) = Program execution time: 1000 nanoseconds.	F(20) = Program execution time: 31800 nanoseconds.
Solving for fib(21)	Solving for fib(21)
F(21) = Program execution time: 700 nanoseconds.	F(21) = Program execution time: 50300 nanoseconds.
Solving for fib(22)	Solving for fib(22)
F(22) = Program execution time: 600 nanoseconds.	F(22) = Program execution time: 80900 nanoseconds.
Solving for fib(23)	Solving for fib(23)
F(23) = Program execution time: 700 nanoseconds.	F(23) = Program execution time: 132000 nanoseconds.
Solving for fib(24)	Solving for fib(24)
F(24) = Program execution time: 600 nanoseconds.	F(24) = Program execution time: 180500 nanoseconds.
Solving for fib(25)	Solving for fib(25)
F(25) = Program execution time: 500 nanoseconds.	F(25) = Program execution time: 144800 nanoseconds.
Solving for fib(26)	Solving for fib(26)
F(26) = Program execution time: 500 nanoseconds.	F(26) = Program execution time: 233300 nanoseconds.
Solving for fib(27)	Solving for fib(27)
F(27) = Program execution time: 500 nanoseconds.	F(27) = Program execution time: 379400 nanoseconds.
Solving for fib(28)	Solving for fib(28)
F(28) = Program execution time: 500 nanoseconds.	F(28) = Program execution time: 610300 nanoseconds.



```
Solving for fib(29)
F(29) = Program execution time: 600 nanoseconds.
Solving for fib(30)
F(30) = Program execution time: 600 nanoseconds.
Solving for fib(31)
F(31) = Program execution time: 600 nanoseconds.
Solving for fib(32)
F(32) = Program execution time: 700 nanoseconds.
Solving for fib(33)
F(33) = Program execution time: 600 nanoseconds.
Solving for fib(34)
F(34) = Program execution time: 700 nanoseconds.
Solving for fib(35)
F(35) = Program execution time: 700 nanoseconds.
Solving for fib(36)
F(36) = Program execution time: 700 nanoseconds.
Solving for fib(37)
F(37) = Program execution time: 700 nanoseconds.
Solving for fib(38)
F(38) = Program execution time: 600 nanoseconds.
Solving for fib(39)
F(39) = Program execution time: 700 nanoseconds.
Solving for fib(40)
F(40) = Program execution time: 700 nanoseconds.
```

```
Solving for fib(29)
F(29) = Program execution time: 988000 nanoseconds.
Solving for fib(30)
F(30) = Program execution time: 1600000 nanoseconds.
Solving for fib(31)
F(31) = Program execution time: 2585000 nanoseconds.
Solving for fib(32)
F(32) = Program execution time: 4193200 nanoseconds.
Solving for fib(33)
F(33) = Program execution time: 6791400 nanoseconds.
Solving for fib(34)
F(34) = Program execution time: 11028800 nanoseconds.
Solving for fib(35)
F(35) = Program execution time: 18169700 nanoseconds.
Solving for fib(36)
F(36) = Program execution time: 29202500 nanoseconds.
Solving for fib(37)
F(37) = Program execution time: 46851600 nanoseconds.
Solving for fib(38)
F(38) = Program execution time: 75976400 nanoseconds.
Solving for fib(39)
F(39) = Program execution time: 122238800 nanoseconds.
Solving for fib(40)
F(40) = Program execution time: 214184800 nanoseconds.
```

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

7.0 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.

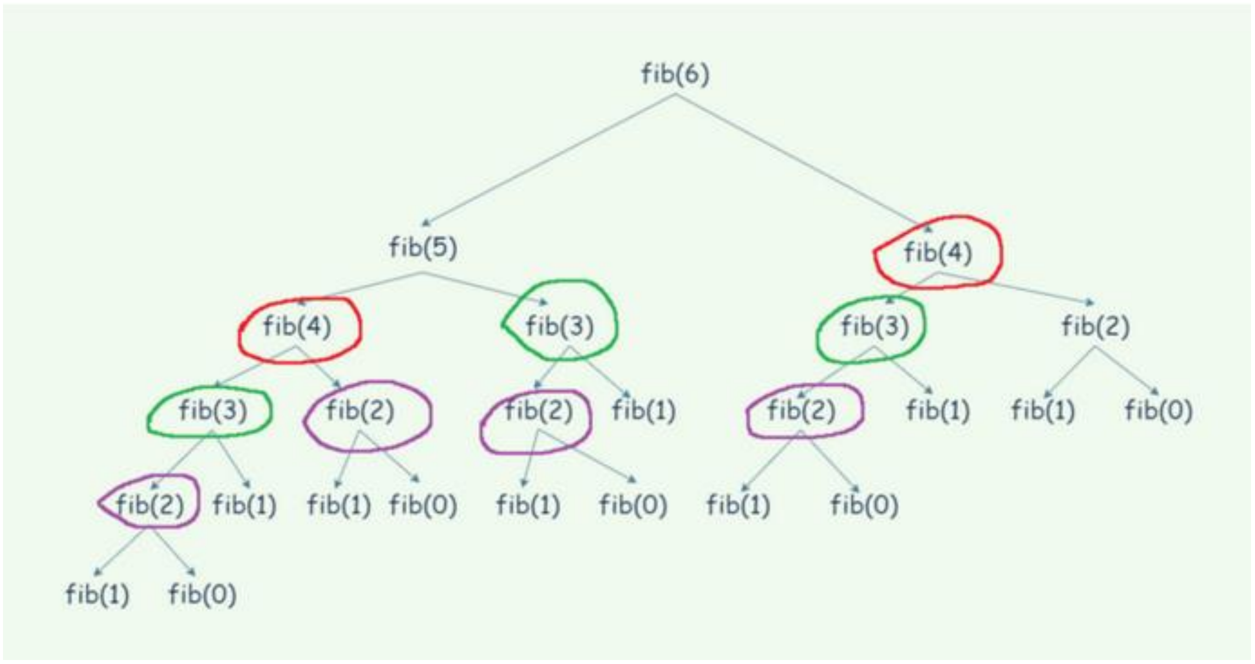
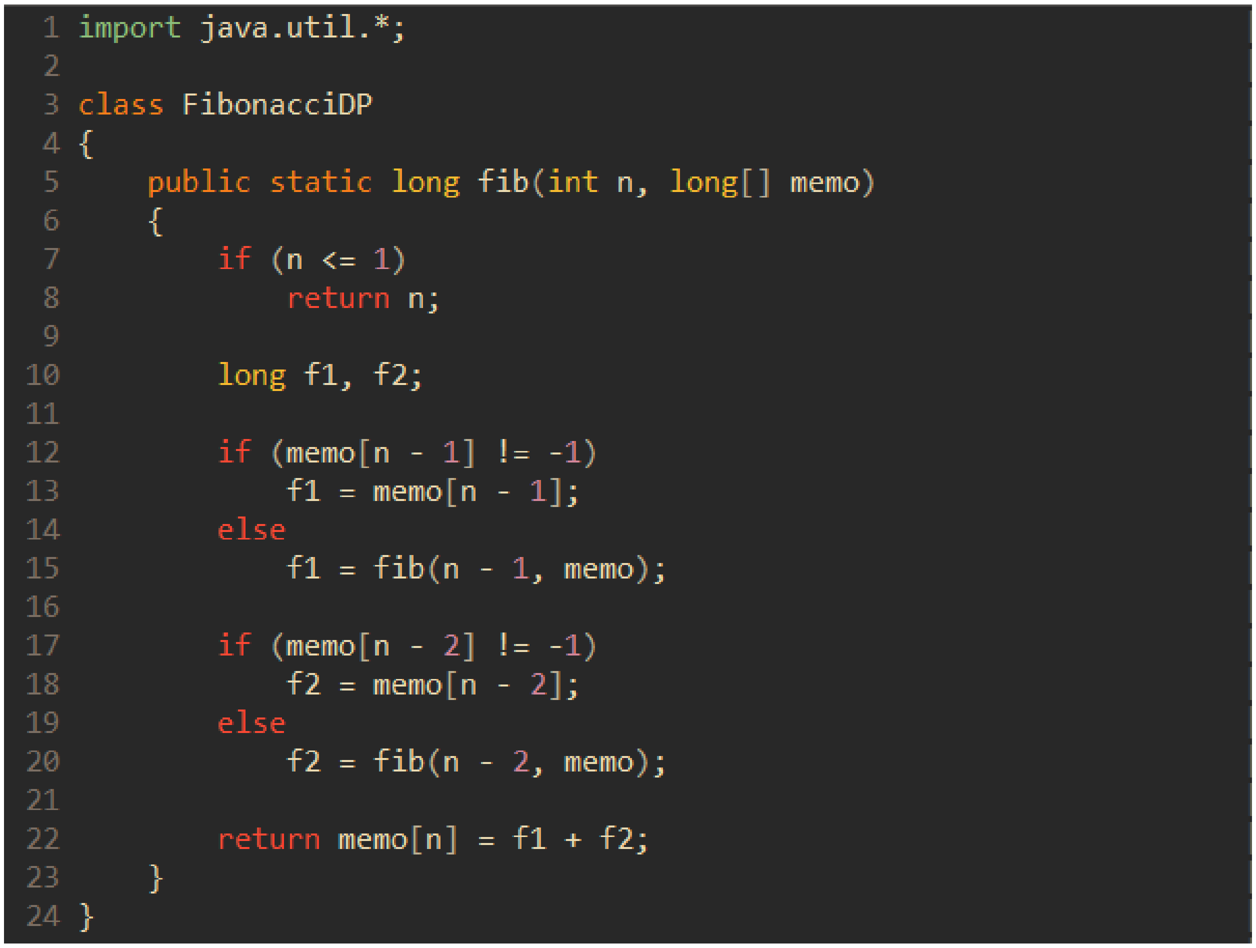


Figure 6: Each step circled is a sequence being calculated multiple times, using up program resources and time.

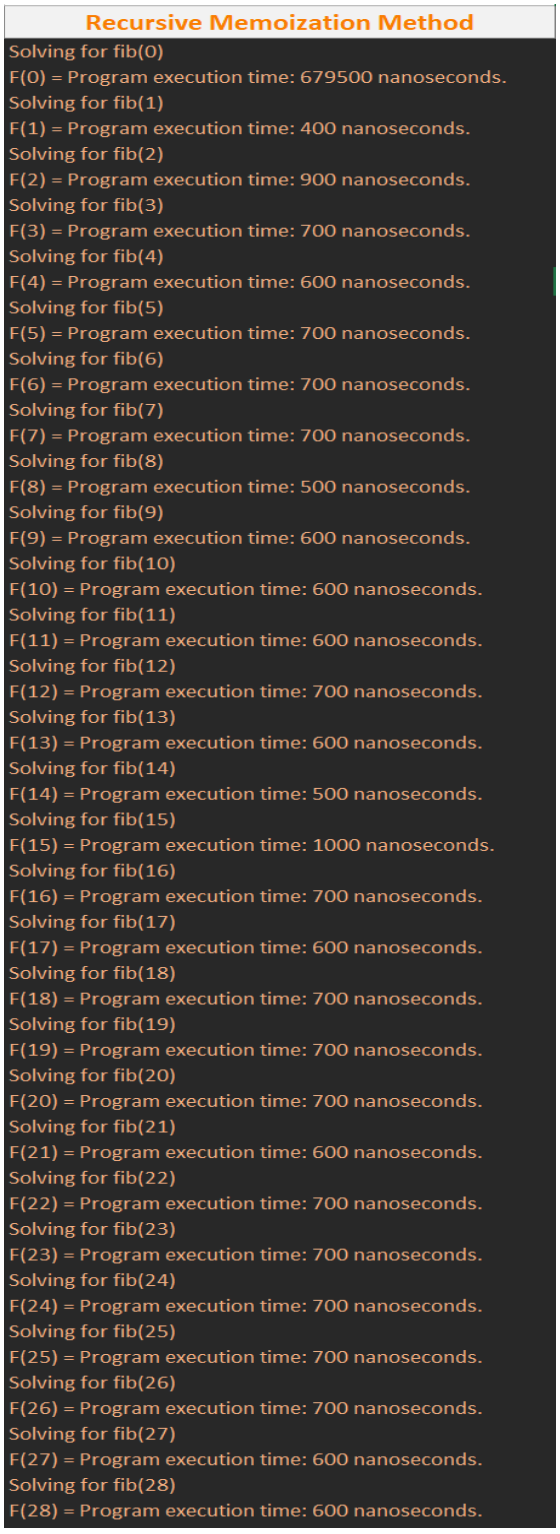
8.0 Different Solutions to Recursion

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.

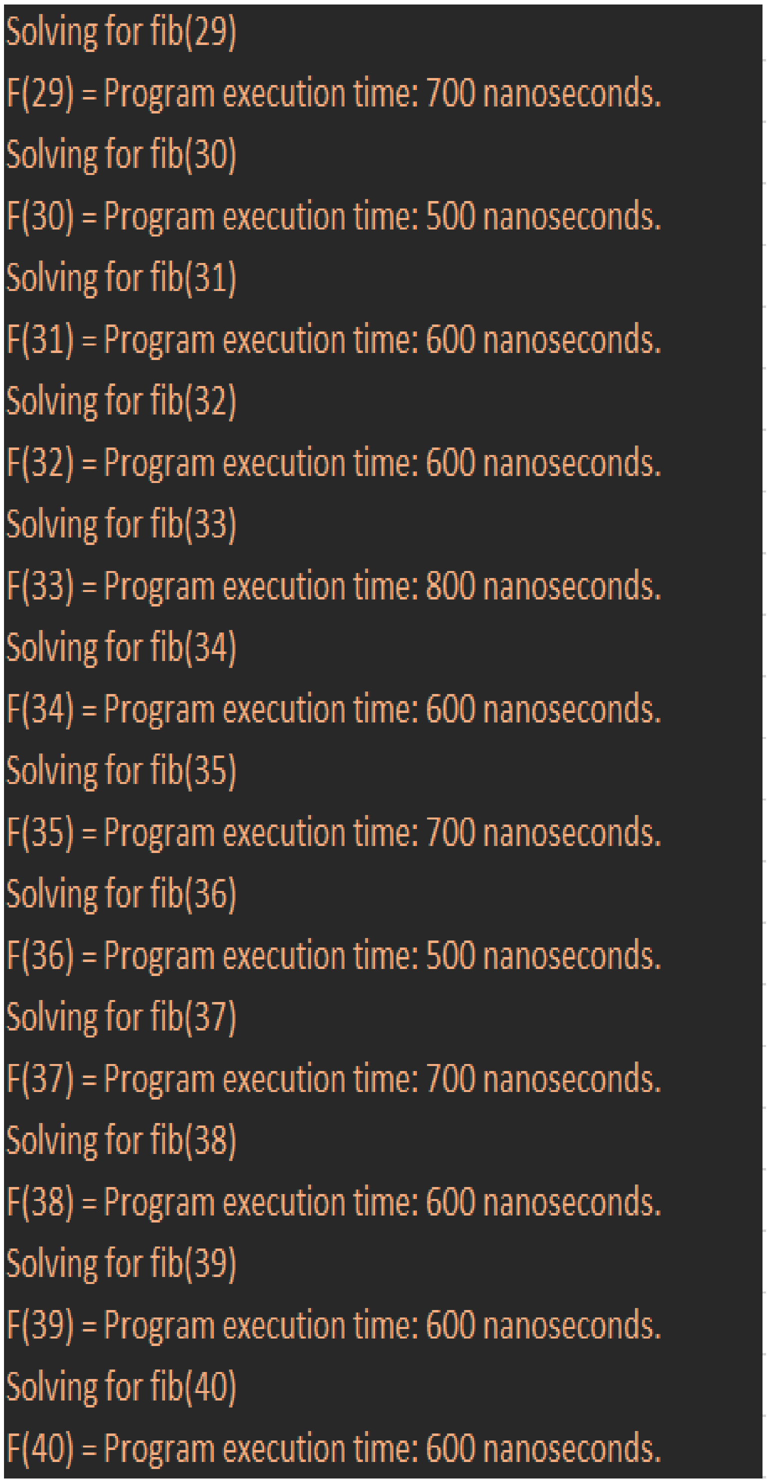


```java
import java.util.*;

class FibonacciDP
{
    public static long fib(int n, long[] memo)
    {
        if (n <= 1)
            return n;

        long f1, f2;

        if (memo[n - 1] != -1)
            f1 = memo[n - 1];
        else
            f1 = fib(n - 1, memo);

        if (memo[n - 2] != -1)
            f2 = memo[n - 2];
        else
            f2 = fib(n - 2, memo);

        return memo[n] = f1 + f2;
    }
}
```

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.



```
Recursive Memoization Method
Solving for fib(0)
F(0) = Program execution time: 679500 nanoseconds.
Solving for fib(1)
F(1) = Program execution time: 400 nanoseconds.
Solving for fib(2)
F(2) = Program execution time: 900 nanoseconds.
Solving for fib(3)
F(3) = Program execution time: 700 nanoseconds.
Solving for fib(4)
F(4) = Program execution time: 600 nanoseconds.
Solving for fib(5)
F(5) = Program execution time: 700 nanoseconds.
Solving for fib(6)
F(6) = Program execution time: 700 nanoseconds.
Solving for fib(7)
F(7) = Program execution time: 700 nanoseconds.
Solving for fib(8)
F(8) = Program execution time: 500 nanoseconds.
Solving for fib(9)
F(9) = Program execution time: 600 nanoseconds.
Solving for fib(10)
F(10) = Program execution time: 600 nanoseconds.
Solving for fib(11)
F(11) = Program execution time: 600 nanoseconds.
Solving for fib(12)
F(12) = Program execution time: 700 nanoseconds.
Solving for fib(13)
F(13) = Program execution time: 600 nanoseconds.
Solving for fib(14)
F(14) = Program execution time: 500 nanoseconds.
Solving for fib(15)
F(15) = Program execution time: 1000 nanoseconds.
Solving for fib(16)
F(16) = Program execution time: 700 nanoseconds.
Solving for fib(17)
F(17) = Program execution time: 600 nanoseconds.
Solving for fib(18)
F(18) = Program execution time: 700 nanoseconds.
Solving for fib(19)
F(19) = Program execution time: 700 nanoseconds.
Solving for fib(20)
F(20) = Program execution time: 700 nanoseconds.
Solving for fib(21)
F(21) = Program execution time: 600 nanoseconds.
Solving for fib(22)
F(22) = Program execution time: 700 nanoseconds.
Solving for fib(23)
F(23) = Program execution time: 700 nanoseconds.
Solving for fib(24)
F(24) = Program execution time: 700 nanoseconds.
Solving for fib(25)
F(25) = Program execution time: 700 nanoseconds.
Solving for fib(26)
F(26) = Program execution time: 700 nanoseconds.
Solving for fib(27)
F(27) = Program execution time: 600 nanoseconds.
Solving for fib(28)
F(28) = Program execution time: 600 nanoseconds.
```



```
Solving for fib(29)
F(29) = Program execution time: 700 nanoseconds.
Solving for fib(30)
F(30) = Program execution time: 500 nanoseconds.
Solving for fib(31)
F(31) = Program execution time: 600 nanoseconds.
Solving for fib(32)
F(32) = Program execution time: 600 nanoseconds.
Solving for fib(33)
F(33) = Program execution time: 800 nanoseconds.
Solving for fib(34)
F(34) = Program execution time: 600 nanoseconds.
Solving for fib(35)
F(35) = Program execution time: 700 nanoseconds.
Solving for fib(36)
F(36) = Program execution time: 500 nanoseconds.
Solving for fib(37)
F(37) = Program execution time: 700 nanoseconds.
Solving for fib(38)
F(38) = Program execution time: 600 nanoseconds.
Solving for fib(39)
F(39) = Program execution time: 600 nanoseconds.
Solving for fib(40)
F(40) = Program execution time: 600 nanoseconds.
```

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/