Finding the Nth Fibonacci Number

COT6405 Project by Nick Petty

Table of Contents

[Abstract 1](#_Toc446542101)

[Problem Description 1](#_Toc446542102)

[Practical Applications 1](#_Toc446542103)

[Competing Algorithms 3](#_Toc446542104)

[Experiments 4](#_Toc446542105)

[Programming Implementation 10](#_Toc446542106)

[Conclusions 10](#_Toc446542107)

[Sources 10](#_Toc446542108)

# Abstract

The Fibonacci Numbers are a sequence of positive integers defined by a recursive relationship. That is, Fn = Fn-1 + Fn-2, where F0 = 0 and F1 = 1. Likely owning to its simplicity, the Fibonacci Numbers appear frequently in the natural world and a wide range sciences. However, the value of Fn grows very quickly, and the recursive method described becomes impossibly slow as n increases. This paper will discuss some uses of Fibonacci Numbers, then examine two methods of finding the nth Fibonacci Number, given n. The two methods examined will be the previously described recursive approach and the dynamic programming approach. By comparing the theoretical and experimental runtimes of both approaches, dynamic programming will be shown to be the better method.

# Problem Description

Given an integer, n > 1, the value of the nth Fibonacci Number, Fn, is sought. This is found by applying the recurrence relation Fn = Fn-1 + Fn-2, where F0 = 0 and F1 = 1. There is discrepancy in this definition, as some problem descriptions may choose F1 = 0 and F2 = 1, or F1 = and F2 =1, or other values and labels for the first two terms. As the values of n increase, the values of Fn grow rapidly and calculation becomes very time and space intensive. Because the Fibonacci Numbers have a wide array of uses and appear frequently in many scientific fields, an efficient method of calculating larger members of this set is necessary.

# Practical Applications

Leonardo of Pisa, commonly known as Fibonacci, introduced a series of recursive numbers to Western mathematics in his book Liber Abaci, published in 1202. It wasn’t until the 19th century, though, that number theorist Édouard Lucas gave the Fibonacci numbers their name (Knott). The concept of Fibonacci Numbers, and their generating function, had been known to mathematics in other parts of the world as well. Donald Knuth cites Indian scholars and their writings, saying:

“Before Fibonacci wrote his work, the sequence Fn had already been discussed by Indian scholars, who had long been interested in rhythmic patterns... both Gopala (before 1135 AD) and Hemachandra (c. 1150) mentioned the numbers 1,2,3,5,8,13,21 explicitly.” (p. 100)

Appearing frequently in mathematics and science for centuries, it is clear that these numbers are important and have intriguing uses. This section will discuss a few common applications of the Fibonacci Numbers and the general rule that it follows.

The Golden Ratio, ϕ, is a number that often results when establishing a “beautiful” or “natural” ratio between two measurements. Mathematically, two numbers exhibit this relationship if their ratio is the same as the ratio of their sum to the larger of the two (Wikipedia). That is,

This ratio was used frequently in classical architecture and it is observed in many instances of biological growth patterns. The value of ϕ can be calculated by taking any sufficiently large Fibonacci numbers, Fn = Fn-1 + Fn-2, where Fn-1 = a and Fn-2 = b, as shown:

Taking n = 25, for example, the calculation is:

If a highly precise value of ϕ is needed, large Fibonacci Numbers can be used to calculate it.

In computer science, the heap data structure provides an ordering of its elements such that maximum or minimum values can be quickly accessed. Chapter 19 of the CLRS book is dedicated to a special heap implementation, called the Fibonacci Heap, whose value is described in the opening paragraph:

“The Fibonacci heap data structure serves a dual purpose. First, it supports a set of

operations that constitutes what is known as a “mergeable heap.” Second, several

Fibonacci-heap operations run in constant amortized time, which makes this data

structure well suited for applications that invoke these operations frequently.” (p. 505)

The Fibonacci heap is actually a collection of trees that are min-heap ordered, and whose roots are connected in a list. It was named after Fibonacci because the numbers in the series appear in the runtime analysis of the data structure. Specifically, the Fibonacci heap uses a forest of trees, given an order, n, such that the number of nodes in the trees is ≥ Fn+2 (Growing the Web). Operational runtime depends on summing the numbers of nodes in the trees, which is a set of Fibonacci numbers.

In nature, the Fibonacci numbers are found in what is popularly called the “Bee Ancestry Code.” A point of bee biology is that female bees produce males when unfertilized and females when fertilized (Bee Ancestry). From this, tracing 1 male bee’s ancestry shows he has 1 parent, 2 grandparents, 3 great-grandparents, 5 great-great-grandparents, and so on, which are the Fibonacci numbers. The following table illustrates that n generations back, a male bee has Fn ancestors:

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| F |  | M | F | F |  | M |  | Generation 5: 3 + 2 = 5 bees | | F5 | |
| \ |  | / | | | \ |  | / |  |  |  | |  |
|  | F |  | M |  | F |  |  | Generation 2: 2 + 1 = 3 bees | | F4 | |
|  | \ |  | / |  | / |  |  |  |  | |  |
|  |  | F |  | M |  |  |  | Generation 3: 1 + 1 = 2 bees | | F3 | |
|  |  | \ |  | / |  |  |  |  |  | |  |
|  |  |  | F |  |  |  |  | Generation 2: 1 + 0 = 1 bee | | F2 | |
|  |  |  | | |  |  |  |  |  |  | |  |
|  |  |  | M |  |  |  |  | Generation 1: 1 bee |  | | F1 |

The Fibonacci Numbers are used frequently in the sciences, so much so that a dedicated research community has developed around them. This is the Fibonacci Association, and it publishes a journal called the Fibonacci Quarterly, which is at [www.fq.math.ca](http://www.fq.math.ca). The journal has been in regular publication since 1963, with new volumes released four times per year. The Fibonacci Association also regularly hosts conferences around the world focused on recent research related to the Fibonacci Numbers and sequence. Considering such long-running work dedicated to applications of Fibonacci Numbers, it’s clear that they have great importance in mathematics and beyond.

# Competing Algorithms

Two methods of calculating the nth Fibonacci Number are implemented, one done recursively and one done with dynamic programming. By running the two methods against the same values of n, their comparative execution speeds are measured. Even with small values of n, it is quickly apparent that the recursive method is unsuitable for regular use if Fibonacci numbers are needed.

The recursive algorithm is the simpler of the two, requiring only a base case and a recursive step. This is the pseudo code, from GeeksforGeeks:

recursive\_fib (int n)

if n ≤ 1

return n

return recursive\_fib(n-1) + recursive\_fib(n-2)

The single comparison and three arithmetic operations all take constant time, c, so the recurrence relation T(n) = T(n-1) + T(n-2) + c is used to determine the theoretical runtime (Mycodeschool). To find an upper bound, observe that T(n-2) ≤ T(n-1), but they are close enough that T(n-2) ≈ T(n-1) can be estimated. In this case, the recurrence relation becomes T(n) = 2T(n-1) + c, and backwards substitution can be applied:

T(n) = 2T(n-1) + c

= 2(2T(n-2) + c) + c = 4T(n-2) +3c

= 2(4T(n-3) + 3c) + c = 8T(n-3) + 7c

= 2(8T(n-4) + 7c) + c = 16T(n-4) + 15c

= 2kT(n-k) + (2k-1)c

Since T(0) = 0 🡪 n – k = 0 🡪 k = n, and then

T(n) = 2nT(0) + (2n – 1)c = 2nc – c

T(n) = O(2n)

This exponential runtime is the result of repeatedly calculating the same numbers because the recursion does not stop until it reaches F0 or F1. The recursion tree created by recursive\_fib(5) is shown below (GeeksforGeeks). In this example, there are 15 recursive calls, which find F0 three times, F1 five times, F2 three times, F3 two times, and F4 once before returning F5. Because of this large number of repeated calculations, the recursive implementation of the nth Fibonacci Number runs impossibly slow.

By using dynamic programming to calculate the nth Fibonacci Number, the excessive work done by recursion is avoided. This implementation creates an array and stores the initial sequence values of F0 and F1, then iterates up to n while filling the array with calculated Fibonacci Numbers. The index of each element in the array corresponds to its position in the sequence, with Fn being the last element added at the largest index. The pseudo code is somewhat longer than the recursive version, but still very short and straightforward (GeeksforGeeks):

dynamic\_fib (int n)

int fib\_array[n+1]

fib\_array[0] = 0

fib\_array[1] = 1

for i = 2 to n

fib\_array[i] = fib\_array[i-1] + fib\_array[i-2]

return fib\_array[n]

This method is initialized with three constant time operations, and has only a single for loop that traverses the n elements of the array. At each element, an operation is performed, but this runs in constant time and does not increase overall run time. The total theoretical runtime for this method is therefore O(n).

With one method of calculating the nth Fibonacci Number running at O(2n) time compared to another running at O(n) time, it is clear that the recursive implementation is vastly inferior to the dynamic programming version. The following experimental section will prove this with actual program execution times.

# Experiments

The first step in comparing the efficiency of the recursive algorithm to the dynamic programming algorithm is to make a direct measurement of their performance on the same data set. In the graph below, the first 50 Fibonacci Numbers are calculated with both versions of the algorithm and the runtimes for each value of n are recorded in milliseconds. This is 100 total runs. Clearly, the dynamic programming approach is superior, as the recursive implementation quickly tends towards impossibly long runtimes. However, the previously stated theoretical runtimes must be proven accurate as well, so further investigation is required.



Examining the recursive implementation on its own, verification of the theoretical runtime of O(2n) must be demonstrated. To show this, two constants, cmin and cmax are found such that cmin2n ≤ experimental runtime ≤ cmax2n. These values of c are calculated by taking the ratio of experimental runtime to theoretical runtime for each run, then finding the maximum and minimum of this set. To estimate theoretical runtime, the time to return F0 and F1 is assumed to be 0.001ms and 0.002ms, respectively. The following table shows the value of n for each run, the computed Fibonacci Number, the actual runtime, the theoretical runtime, and the constant c. The values of cmin and cmax are determined, then cmin2n and cmax2n are calculated, which provides the bounding for the experimental runtime. Graphing these points shows that the recursive implementation of the nth Fibonacci Number is indeed O(2n).





Moving onto the dynamic programming implementation, the theoretical runtime is O(n), and proving this experimentally follows the same methodology that was done for the recursive implementation. This time, cminn ≤ experimental runtime ≤ cmaxn is the bounding function, however. From the graph to the right, it is clear that the first 50 Fibonacci numbers do not provide meaningful data regarding experimental runtimes. Only occasionally does the runtime exceed 0ms, so the values of c are worthless.

To get meaningful data, a larger investigation space is needed. For this, the values of n were calculated up to 150,000 – that is, F150,000 – at which point asymptotic behavior could be observed. The graph and table below show the results of this modified experiment. Here, the first Fibonacci Number calculated was F3,000, and its theoretical runtime was assumed to be 3ms, which was quite accurate. From there, the next tested value of n steps 3,000 each time, giving a total of 50 runs from n = 3,000 to n = 150,000. The values of cmin and cmax found on these runs were able to show that the experimental runtimes were bounded by a constant, c, multiplied by the theoretical runtimes. Therefore, the dynamic programming implementation has a runtime of O(n).





Through the use of a Java program, the recursive implementation of the nth Fibonacci Number calculation was shown to run in O(2n) time, while the dynamic programming implementation ran in O(n) time. This reinforces the stated argument that the recursive implementation is unsuitable for calculating large values of n and that the dynamic programming implementation is better for any purpose. The next section will discuss the details of how the program used for this experiment was created.

# Programming Implementation

Coding of the nth Fibonacci Number calculator was done in Java, with both versions of the algorithm contained as methods within a single class, Nth\_Fib. A helper method, run\_algo(), was created to select the version of the algorithm to run, track experimental and theoretical runtimes, calculate c, and manage input and output. No dedicated classes or frameworks were used to record runtimes; the system time was taken before and after each run and the difference was the runtime. Because output data could be very large, the console was only used for monitoring progress, while experimental results were saved in a .csv file. Furthermore, the calculated values of Fn grew extremely large, so the Long and BigInteger data types were used. For example, in the table above for F3,000 to F150,000, the values of Fn are thousands of digits long, but are not fully shown in the cells. Because of this, the dynamic programming implementation also reaches a limit based on hardware memory availability. In fact, the machine running the tests ran out of memory around F180,000. A more space-efficient implementation is possible, but for simplicity was not used here. This program also has no user interface; all experimental controls are handled through the main method in the Nth\_Fib class by calling the helper method with the parameters to be investigated. To run the program with the experimental configuration in this report, the nth\_fib.jar file can be executed from the command line or from a Windows/Mac/Linux GUI.

Additional output formatting, calculation of cmin, cmax, and related bounds, and graphing was done in Excel. With program output being stored in .csv files, these steps only required using Excel’s included tools and converting the document to .xlsx format. Since multiple files were created, raw output was cut and pasted into a single file.

# Conclusions

The primary observation of this experiment is that even simple algorithms can become impossibly slow, and that efficient algorithms need not be complicated. The recursive method of calculating the nth Fibonacci Number is easy to understand and program, but useless beyond n = 50 because of runtime growth. With dynamic programming, this problem is solved and without adding more than a few lines of code to the already brief program. Despite their simplicity, the Fibonacci Numbers and their generating function play an important role in mathematics and science. Research into and observations of these numbers has gone on for centuries and will continue as long as humans search for patterns in the world around them.

# Sources

"Bee Ancestry." University Child Development School (2007): n. pag. Web. <http://www.ucds.org/spark/magazine-curriculum/Fibonacci\_BeeAncestry.pdf>.

Cormen, Thomas H., Charles Eric. Leiserson, Ronald L. Rivest, and Clifford Stein. "Fibonacci Heaps." Introduction to Algorithms. Third ed. Cambridge (Mass.): MIT, 2009. 506-30. Print.

"Fibonacci Heap." Growing with the Web. N.p., 19 June 2015. Web. 20 Mar. 2016. <http://www.growingwiththeweb.com/2014/06/fibonacci-heap.html>.

"The Fibonacci Quarterly." The Fibonacci Quarterly. Ed. Curtis Cooper. N.p., n.d. Web. 07 Mar. 2016. <http://www.fq.math.ca/index.html>.

"Golden Ratio." Wikipedia. Wikimedia Foundation, n.d. Web. 20 Mar. 2016. <https://en.wikipedia.org/wiki/Golden\_ratio>.

Knott, R. "The Life and Numbers of Fibonacci." The Life and Numbers of Fibonacci. +Plus Magazine, n.d. Web. 20 Mar. 2016. <https://plus.maths.org/content/life-and-numbers-fibonacci>.

Knuth, Donald Ervin. The Art of Computer Programming. Fundamentals Algorithms. 3rd ed. N.p.: n.p., 1968. 100. Print.

Mycodeschool. "Time Complexity Analysis of Recursion - Fibonacci Sequence." YouTube. YouTube, 10 Oct. 2012. Web. 22 Mar. 2016. <https://www.youtube.com/watch?v=pqivnzmSbq4>.