The Fibonacci Sequence

Noah Tillier

March 17, 2026

This is the augmented transcript of a lecture given by Luc Devroye on the 6th of January 2026 for the Honors Data Structures and Algorithms class (COMP 252, McGill University). The subject was computing the Fibonacci Sequence. These notes are heavily inspired by David Eppstein's notes for his 1996 course Design and Analysis of Algorithms (ICS 161, UCI).

There are many algorithms to compute the Fibonacci sequence. We must evaluate the efficiency of each algorithm to implement the best option.

Defining the Problem

The Fibonacci sequence is defined by a recurrence. A *recurrence* expresses a sequence in terms of its previous elements. Let x_n denote the n^{th} term of the Fibonacci sequence. Then:

$$x_n = x_{n-1} + x_{n-2}, \qquad x_0 = 0, \qquad x_1 = 1.$$

The Fibonacci sequence is a *linear recurrence* of the general form

$$R_n = c_1 R_{n-1} + c_2 R_{n-2} + \cdots + c_k R_{n-k},$$

where each c_i is constant and k is fixed. Because each term is a linear combination of earlier terms, we look for a solution whose growth is multiplicative from one term to the next. If some growth ratio r exists, then substituting r^n for R_n gives us the characteristic equation for r:[1]

[1] Wikipedia contributors [2026]

$$r^n = c_1 r^{n-1} + c_2 r^{n-2} + ... + c_k r^{n-k}.$$

In the Fibonacci example, substituting $x_n = r^n$ into the recurrence gives

$$r^n = r^{n-1} + r^{n-2},$$

i.e.,

$$r^2 = r + 1.$$

Solving the quadratic equation gives two solutions,

$$r = \frac{1 \pm \sqrt{5}}{2}.$$

Thus, x_n can be expressed as a linear combination of n-th powers of the two roots:

$$x_n = A \left(\frac{1+\sqrt{5}}{2} \right)^n + B \left(\frac{1-\sqrt{5}}{2} \right)^n .$$

Using the initial conditions, $x_0 = 0$ and $x_1 = 1$, when $n = 0$,

$$A + B = 0,$$

and when $n = 1$,

$$A\left(\frac{1+\sqrt{5}}{2}\right) + B\left(\frac{1-\sqrt{5}}{2}\right) = 1.$$

Solving the system of linear equations yields $A = \frac{1}{\sqrt{5}}$, and $B = -\frac{1}{\sqrt{5}}$. Therefore,

$$x_n = \frac{\left(\frac{1+\sqrt{5}}{2}\right)^n - \left(\frac{1-\sqrt{5}}{2}\right)^n}{\sqrt{5}} = \Theta\left(\left(\frac{1+\sqrt{5}}{2}\right)^n\right).$$

We must evaluate $\frac{1+\sqrt{5}}{2}$ and $\frac{1-\sqrt{5}}{2}$ before computing x_n on a computer. However, if we round $\frac{1+\sqrt{5}}{2}$ and $\frac{1-\sqrt{5}}{2}$ to three decimal places, our mathematical function diverges from the Fibonacci sequence as soon as $n = 19$. This divergence occurs regardless of the number of decimal places retained, and raising a floating-point number to the n-th power is computationally expensive. Therefore, it is advantageous to work with integers.

We expect $x_{19} = 4181$. However,

$$\frac{1.618^{19} - 0.618^{19}}{2.236} = 4180,$$

assuming we round the result up.

Proposal 1: Recursion Model

Our first instinct is to translate the recurrence relation into a recursive function:

```
RECURSIVE_COMPUTE($n$)
1  if $n = 0 || n = 1$
2      return $n$
3  else
4      return RECURSIVE_COMPUTE($n - 1$) + RECURSIVE_COMPUTE($n - 2$)
```

The time complexity of Recursive_Compute is:

$$T_n = 1 + T_{n-1} + T_{n-2},$$

with $T_0 = T_1 = 1$. Adding 1 to both sides yields

$$T_n + 1 = (T_{n-1} + 1) + (T_{n-2} + 1).$$

We can express $T_n + 1$ and x_n in table form:

T_0+1	T_1+1	T_2+1	T_3+1	T_4+1	$\dots$
2	2	4	6	10	$\dots$

x_0	x_1	x_2	x_3	x_4	x_5	$\dots$
0	1	1	2	3	5	$\dots$

$T_n + 1$ and x_n visualized

Notice that $T_n + 1 = 2x_{n+1}$. Therefore,

$$T_n \leq 2x_{n+1} = \Theta\left(\left(\frac{1+\sqrt{5}}{2}\right)^n\right).$$

In other words, the complexity grows exponentially in n.

Proposal 2: Dynamic Programming

WE CAN DO BETTER, by computing and storing all values $x_0, x_1, \ldots, x_n$:

```
LINEAR_COMPUTE($n$)
1  $x_0 = 0, x_1 = 1$
2  for $i = 2$ to $n$
3      $x_i = x_{i-1} + x_{i-2}$
4  return $x_n$
```

The time complexity of LINEAR_COMPUTE is $T_n = \Theta(n)$. Methods that compute all sub-solutions are commonly referred to as dynamic programming methods.

Proposal 3: Fast Exponentiation

WE CAN DO EVEN BETTER by pairing consecutive Fibonacci numbers into a vector. Defining the matrix $M = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}$, we have

$$\begin{bmatrix} x_n \\ x_{n-1} \end{bmatrix} = M \begin{bmatrix} x_{n-1} \\ x_{n-2} \end{bmatrix} = M^2 \begin{bmatrix} x_{n-2} \\ x_{n-3} \end{bmatrix} = \cdots = M^{n-1} \begin{bmatrix} x_1 \\ x_0 \end{bmatrix}.$$

Let $m = n - 1$. When m is even $M^m = (M^{\frac{m}{2}})^2$, and when m is odd $M^m = (M^{\frac{m-1}{2}})^2 M$. By halving the exponent size at each iteration, this technique efficiently computes M in approximately $\log_2 m$ steps.

```
FAST_EXPONENTIATE($M, n$)
1  if $n = 1$
2      return $M$
3  else if $n = 2$
4      return $M^2$
5  else
6      if $n \mod 2 == 0$
7          return FAST_EXPONENTIATE($M$, $n/2$)$^2$
8      else
9          return FAST_EXPONENTIATE($M$, $\frac{n-1}{2}$)$^2 \times M$
```

```
FAST_COMPUTE($n$)
1  $X \leftarrow$ FAST_EXPONENTIATE($M$, $n$-1)
2  return $X[0, 0]$
```

The time complexity of FAST_COMPUTE is

$$T_n = \Theta\left(\log_2 n\right).$$

Our fast exponentiation solution is orders of magnitude faster than our dynamic programming solution, where $T_n = \Theta(n)$, and our recursion model solution, where $T_n = \Theta\left(\left(\frac{1+\sqrt{5}}{2}\right)^n\right)$.

References

Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. MIT Press, Cambridge, MA, 4 edition, 2022. ISBN 9780262046305.

David Eppstein. Fibonacci numbers. `https://web.archive.org/web/20250724132900/https://ics.uci.edu/~eppstein/161/960109.html`, 1996. Lecture notes for ICS 161: Design and Analysis of Algorithms, UC Irvine.

Wikipedia contributors. Linear recurrence with constant coefficients — Wikipedia, the free encyclopedia. `https://en.wikipedia.org/wiki/Linear_recurrence_with_constant_coefficients`, 2026. Accessed: 2026-03-02.