Fast Fibonacci algorithms

Definition: The Fibonacci sequence is defined as $F(0) = 0$, $F(1) = 1$, and $F(n) = F(n-1) + F(n-2)$ for $n \geq 2$. So the sequence (starting with $F(0)$) is $0, 1, 1, 2, 3, 5, 8, \ldots$.

If we wanted to compute a single term in the sequence (e.g. $F(n)$), there are a couple of algorithms to do so, but some algorithms are *much* faster than others.

Algorithms

Textbook recursive algorithm (extremely slow)

Naively, we can directly execute the recurrence as given in the mathematical definition of the Fibonacci sequence. Unfortunately, it's hopelessly slow: It uses $O(n)$ stack space and $O(\phi^n)$ arithmetic operations, where $\phi = \frac{\sqrt{5}+1}{2}$ (the golden ratio). In other words, the number of operations to compute $F(n)$ is proportional to the resulting value itself, which grows exponentially.

Dynamic programming (slow)

It should be clear that if we've already computed $F(k-2)$ and $F(k-1)$, then we can add them to get $F(k)$. Next, we add $F(k-1)$ and $F(k)$ to get $F(k+1)$. We repeat until we reach $k = n$. Most people notice this algorithm automatically, especially when computing Fibonacci by hand. This algorithm takes $O(1)$ space and $O(n)$ operations.

Matrix exponentiation (fast)

The algorithm is based on this innocent-looking identity (which can be proven by mathematical induction):

$$\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^n = \begin{bmatrix} F(n+1) & F(n) \\ F(n) & F(n-1) \end{bmatrix}.$$

It is important to use exponentiation by squaring with this algorithm, because otherwise it degenerates into the dynamic programming algorithm. This algorithm takes $O(1)$ space and $O(\log n)$ operations. (Note: This is counted in terms of the number of bigint arithmetic operations, not primitive fixed-width operations.)

Fast doubling (faster)

Given $F(k)$ and $F(k+1)$, we can calculate these:

$$\begin{aligned} F(2k) &= F(k)[2F(k+1) - F(k)]. \\ F(2k+1) &= F(k+1)^2 + F(k)^2. \end{aligned}$$

These identities can be extracted from the matrix exponentiation algorithm. In a sense, this algorithm is the matrix exponentiation algorithm with the redundant calculations removed. It should be a constant factor faster than matrix exponentiation but the asymptotic time complexity is still the same.

Summary: The two fast Fibonacci algorithms are matrix exponentiation and fast doubling, each having an asymptotic complexity of $O(\log n)$ bigint arithmetic operations. Both algorithms use multiplication, so they become even faster when Karatsuba multiplication is used. The other two algorithms are slow; they only use addition and no multiplication.