The Complexity of Computing Fibonacci Numbers

John Karro

August 25, 1993

Abstract

This paper examines the time and space required to recognize and generate Fibonacci numbers. We show that the Fibonacci numbers are a deterministic context-sensitive language and prove that they are not context-free. We show that the generation of Fibonacci numbers cannot be accomplished by a finite state or push-down automaton, and that any algorithm generating the nth Fibonacci number f_n will require at least $\log_2 f_n$ bits of memory. We discuss the possibility of time-space tradeoff on such algorithms, and the possibility of an algorithm to generate them in linear time.

Introduction:

Most readers are familiar with the Fibonacci numbers, the sequence of numbers generated by the difference equation:

$$f_{n+1} = f_n + f_{n-1} \quad f_0 = 0 \quad f_1 = 1$$

In the paper of Cull and Holloway [2], several alternative algorithms for computing the nth Fibonacci number, f_n, are presented, along with the time each algorithm takes to compute f_n (as measured by the number of bit operations). Based on this work, Babb[1] was able to find a new sequence of algorithms to compute these numbers, the limit of which will compute it in the least bit operations of any known algorithm. However, the number of bit operations is still no better than $O(n \log n \log \log n)$. Is there an algorithm which will compute the nth Fibonacci number in $O(n)$ bit operations?

The purpose of this research was to prove any one of three objectives. Ideally we would have liked to find an algorithm that would compute f_n in linear time. Alternatively, we would have been quite happy to prove that no such algorithm exists, and to find the actual lower bound. Last of all, it would have been nice to show that there is a time-space tradeoff on such algorithms, and find a reasonable lower bound on that tradeoff. None of these objectives were actually accomplished; the following paper discusses the methods we used in trying to prove these results, and presents some small results that may help in eventually achieving any of our original objectives.

Notation and background results:

The notation used in this paper will be standard number theory and automata notation. f_n will be the nth Fibonacci number. For integers a, b, and c: $a \mid b$ indicates that a divides

b, (a,b) is the greatest common divisor of a and b, and $a \equiv b \ (mod\ c)$ means $c \mid (a-b)$. If v is any string of characters, then $|v|$ is the length of the string and v^i is the string formed by the concatenation of i copies of v.

Several identities and theorems about Fibonacci numbers are used in our results, so we will present them now. For the proofs consult [7].
Identities:

$$f_n = \frac{\lambda_0^n - \lambda_1^n}{\sqrt{5}} \tag{1}$$

where

$$\lambda_0 = \frac{1+\sqrt{5}}{2} \text{ and } \lambda_1 = \frac{1-\sqrt{5}}{2}$$

$$f_{n+k} = f_{n-1}f_k + f_n f_{k+1} \tag{2}$$

$$f_{2n} = f_n(f_{n-1} + f_{n+1}) \tag{3}$$

Theorem 1 *$a \mid b$ implies $f_a \mid f_b$, and for $a \neq 2$ $f_a \mid f_b$ implies $a \mid b$.*

Theorem 2 $(f_a, f_b) = f_{(a,b)}$

Theorem 3 *Given any integer m, there exists an integer n such that $0 < n \leq m^2$ and $m \mid f_n$.*

Space requirements for the recognition of Fibonacci numbers:

The first resulted we wanted was to find a bound on the space needed to recognize Fibonacci numbers. Obviously, given a number m, we can determine if m is a Fibonacci number simply by calculating all Fibonacci numbers up to m, and comparing m to the largest of these. If n is the greatest integer such that $f_n \leq m$, and the calculations are done in base b, it takes no more than $\log_b f_n$ space to hold any needed Fibonacci number in memory, and we need never hold more than two Fibonacci numbers in memory at once. So we can certainly classify m in $2\log_b(f_n) \approx 2n \log_b \lambda_0$ space, and thus the space needed is no more than $O(n)$. Our next theorem employs the Pumping Lemma and Ogden's Lemma from [4] to give a lower bound on the space required.

Theorem 4 *There are no finite state or push-down automata that can recognize Fibonacci Numbers.*

Proof: Assume there is such an automaton. Let b be the input base, n_1 denote the constant from the Pumping Lemma for CFLs, n_2 denote the constant from Ogden's Lemma, and $n = n_1 + n_2$.

For $b = 1$: Let $a = n_1 + 7$, and it follows that $a < f_{a-1} < f_a$. Let $z = 1^{f_a}$. Then $n_1 < a < f_a = |z|$, so we can apply the Pumping Lemma and get strings u, v, w, x, and y such that $0 < |v| + |x| \leq n_1$, $|uvwxy| = f_a$, and $|uv^2wx^2y|$ is also a Fibonacci Number. But:

$$f_a = |uvwxy| < |uv^2wx^2y| \leq f_a + n_1 < f_a + a < f_a + f_{a-1} = f_{a+1}$$

Thus $f_a < |uv^2wx^2y| < f_{a+1}$, so $|uv^2wx^2y|$ is not a Fibonacci Number, and we have a contradiction.

For $b > 1$: By Theorem 3 we can choose some k_1 such that $b^n \mid f_{k_1}$, so the n right-most bits of the string representing f_{k_1} in will all be zeroes. Then we can apply the Pumping Lemma to f_{n_1} and find binary numbers u, v, w, x and y such that for any $i \geq 0$, the binary number uv^iwx^iy is a Fibonacci Number (and equal to f_{k_1} if $i = 1$). From this we create an increasing sequence of integers k_i such that for all $i \geq 0$:

$$f_{k_i} = uv^iwx^iy = ub^{i|v|+|w|+i|x|+|y|} + vb^{|w|+i|x|+|y|}(1 + b^{|v|} + b^{2|v|} + \cdots + b^{(i-1)|v|}) +$$

$$wb^{i|x|+|y|} + xb^{|y|}(1 + b^{|x|} + \cdots + b^{(i-1)|x|}) + y \tag{4}$$

Apply Ogden's Lemma by "marking" the n_2 right-most digits of f_{k_1}. That Lemma then assures us that at least one bit of the string vwx is marked, which in turn implies the right-most bit of x is marked. Then the Pumping Lemma for push-down automata assures us that the string vwx is no longer than n_1, which implies the the left-most bit of vwx extends no farther left than the $(n-1)$th bit from the right end. Since $b^n \mid f_{n_1}$, these bits must all be zero, so $v = w = x = y = 0$, and we can rewrite (4) as

$$f_{k_i} = ub^{i|v|+|w|+i|x|+|y|} + 0 + 0 + 0 + 0 = b^{|v|+|x|}(ub^{(i-1)|v|+|w|+(i-1)|x|+|y|}) = b^{(|v|+|x|)}f_{k_{i-1}} \tag{5}$$

If we now substitute (1) into (5) we get:

$$\lambda_0^{k_i} - \lambda_1^{k_i} = b^{(|v|+|x|)}(\lambda_0^{k_{i-1}} - \lambda_1^{k_{i-1}}),$$

which reduces to:

$$\lambda_0^{k_{i-1}}(\lambda_0^{k_i - k_{i-1}} - b^{(|v|+|x|)}) = \lambda_1^{k_{i-1}}(\lambda_1^{k_i - k_{i-1}} - b^{(|v|+|x|)}). \tag{6}$$

k_i is an increasing sequence of integers, $|\lambda_0| > 1$, and $|\lambda_1| < 1$. Thus as i goes to infinity, the right side of (6) converges to zero and $\lambda_0^{k_{i-1}}$ diverges, hence we eventually have $\lambda_0^{k_i - k_{i-1}} = b^{|v|+|x|}$. This means there exists a constant k such that, for large enough i, $k_i - k_{i-1} = k$, and we get.

$$b^{(|v|+|x|)} = \lambda_0^k \Rightarrow$$

$$b^{(|v|+|x|)}\lambda_1^k = (\lambda_0\lambda_1)^k = (-1)^k \tag{7}$$

However:

$$\lambda_0^k = b^{(|v|+|x|)} \Rightarrow$$

$$\frac{\lambda_0^k - \lambda_1^k}{\sqrt{5}} = \frac{b^{(|v|+|x|)} - \lambda_1^k}{\sqrt{5}} \Rightarrow$$

$$f_k = \frac{b^{(|v|+|x|)} - \lambda_1^k}{\sqrt{5}} \tag{8}$$

Since $b, |x|$, and $|y|$ are all rational and $\sqrt{5}$ is irrational, (7) implies that λ_1^k is rational, and (8) implies it is irrational, thus giving us our contradiction. □

Space requirements for the generation of Fibonacci numbers:

In the same way that we wanted bounds on space needed to recognize Fibonacci numbers, it would also be nice to have bounds on space requirements for generating them. We can use the same method as before to show that the required space is no more than $O(n)$, and we will now present a theorem similar to the last to give a lower bound on the space required. However, in order to do this, we need to first prove four lemmas.

Lemma 1 *There is no deterministic finite state or push-down automaton that, given an input n in base $b > 1$, can give f_n as an output.*

Proof: As there are only a finite amount of states, there are only a finite amount of possible inputs to the stack as the automaton changes state. Let s be the largest number of elements that can be added to the stack at any given movement. If the initial height of the stack is c, then the height of the stack after the last input can be no greater than $c + s \log_b n$.

At the point the input halts, the automaton will be in a specific state and have a specific element at the top of the stack, and this will determine exactly what output the automaton will give from that point until that element of the stack is removed. As there are a finite number of possible elements on the stack and a finite number of states, one state in combination with one element must produce a longest such output of length l_1. Thus, when the input terminates, the automaton will produce no more than $c + s \log_b n$ strings of input, none of which will be longer than l_1. Let l_2 designate the length of the longest output produced when the automaton moves from one state to another. Then we know the upper bound for the length of the input is $l_2 \log_b n + l_1(c + s \log_b n) = c_1 + c_2(\log_b n)$ for some constants c_1, c_2. However, the length of the output needed is either f_n, if the output is in base 1, or $\log f_n = \log(\frac{\lambda_0^n - \lambda_1^n}{\sqrt{5}}) = \log(\frac{\lambda_0^n}{\sqrt{5}}(1 - \frac{\lambda_1^n}{\lambda_0^n})) \geq c_1' + c_2' n$. Thus, for sufficiently large n, the length of the output that it should produce will be larger than the length of output the automaton can produce, and thus the machine cannot work.

Lemma 2 *There is no deterministic finite state or push-down automaton that, given an input n in base 1, can give an output f_n in base 1.*

Proof: The proof follows exactly as the proof of Lemma 1. The maximum stack height will be $c + sn$, so the maximum length of output will be $l_2 n + l_1(c + sn) = c_1 + c_2 n$, while the needed length of output will be f_n, which is clearly greater for large enough n.

Lemma 3 *Any deterministic finite state or push-down automaton that takes a base 1 input of n and produces an output of f_n in base $b > 1$ can never give more than one significant digit of output before the input has terminated, and can only give one digit of output if that digit is the most significant digit of f_n and $b = 2$.*

In order to prove Lemma 3, we will first need to prove three claims.

Claim 1: For any given m, there exists an $n > m$ such that the representations of f_n and f_{n+1} in base $b > 1$ do not have identical least significant digits.

Assume the converse. Then for all $n > m$, $f_n \equiv f_{n+1} \equiv f_{n+2} \ (mod\ b) \Rightarrow b \mid (f_{n+1} - f_n) = f_{n-1}$ and $b \mid (f_{n+2} - f_{n+2}) = f_n \Rightarrow b \mid (f_{n-1}, f_n) = 1$, which contradicts our assumption that $b > 1$.

Claim 2: For any given integer m, there exists an $n > m$ such that the strings representing f_n and f_{n+1} in base $b > 2$ do not have the same most significant digit.

Assume the converse. Then for all $n > m$, f_n, f_{n+1}, f_{n+2} and f_{n+3} all have the same left most digit a, and note that $a > 0$.

Case 1: The lengths of the strings representing f_n and f_{n+1} are the same. Consider their sum. If we add the digits of the two numbers pairwise to get f_{n+2}, then the sum of the left most digits of f_n and f_{n+1} along with any carry will determine the left most digit of f_{n+2}. It is easy to show that a carry can never be greater than one, so we must consider the sums $a + a$ and $a + a + 1$. If these sums are less than b then we have $a + a = a$, indicating $a = 0$, or $a + a + 1 = a$, indicating $a = -1$, neither of which are possible. Thus $a + a \geq b$, so the carry from this is the most significant digit of f_{n+2}. But that carry must be one, which implies that $a = 1$. Thus $2 = a + a \geq b$, which contradicts our original assumption.

Case 2: The length of the string representing f_{n+1} is longer than f_n. If $a < b - 1$ or if $a = b - 1$ and there is no carry when adding the second most significant digits, then the sum of these two numbers, f_{n+2}, will be the same length of f_{n+1}, and we can apply case 1 to show that f_{n+3} does not have a as its most significant digit. If $a = b - 1$ and there is a carry, then the carry from $a + a + 1$ will be the most significant digit of f_{n+2}. The carry must be 1, so $a = 1 \Rightarrow b = 2$, which contradicts or assumption that $b > 2$.

Claim 3: For any given integer m, there exists an $n > m$ such that the binary strings representing f_n and f_{n+1} do not have the same second most significant digit.

First note that the difference of the lengths of the binary representations of two consecutive Fibonacci numbers can never be greater than 1. For if there was such a case, and the length of the lesser of the two, f_n, was $l + 1$, then:

$$\frac{f_{n+1}}{f_n} > \frac{2^{l+1}}{2^l} \geq 2 \Rightarrow$$

$$2f_n < f_{n+1} = f_n + f_{n-1} < f_n + f_n = 2f_n$$

Now assume the converse of the claim. Then there exists an m such that all Fibonacci number greater than f_m have the same second most significant digit.

Case 1: The second most significant digit is 1. Then there must be a greatest Fibonacci number f_n which has 0 as the second most significant digit. Let $l+1$ be the binary length of f_n. Then $f_n = 2^l + x_1$, where $x_1 < 2^{l-1}$. If the length of f_{n+1} is greater than l, then we know that $f_{n+1} = 2^{l+1} + 2^l + x_2$, where $x_2 < 2^l$. Then $f_{n+2} = 2^{l+2} + x_3$ where $x_3 < 2^l + 2^{l-1} < 2^{l+1}$, which contradicts our assumption that f_{n+2} has a 1 as its second most significant digit. On the other hand, if f_n and f_{n+1} are the same length, then we have that $f_{n+1} = 2^l + 2^{l-1} + x_2$ where $x_2 < 2^{l-1}$. We then calculate the next two Fibonacci numbers, and get $f_{n+3} = 2^{l+2} + x_4$, where $x_4 < 2^l + 2^{l-1} < 2^{l+1}$, so we have our contradiction again.
Case 2: The second most significant digit is a 0. Then there must be a greatest Fibonacci number f_n such that the second most significant digit is 1. Let $l+1$ be the length of f_n, so that $f_n = 2^l + 2^{l-1} + x_1$, where $x_1 < 2^{l-1}$. If f_{n+1} is the same length than $f_{n+1} = 2^l + x_2$ where $x_2 < 2^{l-1}$. From this we must calculate that $f_{n+4} = 2^{l+2} + 2^{l+1} + x_5$, where $x_5 < 2^l + 2^{l-1}$, which means its second most significant digit is a 1. If f_{n+1} has a longer length than f_n, then $f_{n+1} = 2^{l+1} + x_2$ where $x_2 < 2^l$. Then we can calculate that $f_{n+2} = 2^{l+1} + 2^l + 2^{l-1} + x_3$ where $x_3 < 2^l + 2^{l-1}$. However, in order to insure that f_{n+2} does not have a 1 as its second most significant digit, we need that $x_3 \geq 2^{l-1}$. Then we calculate $f_{n+3} = 2^{l+2} + 2^l + 2^{l-1} + x_4$ where $2^{l-1} \leq x_4 < 2^{m+1} + 2^{m-1}$. If we write $x_4 = 2^{l-1} + y$ where $y \leq 0$ then we have $f_{n+3} = 2^{l+2} + 2^{l+1} + y$, which implies $y \geq 2^{l+1}$. But then $2^{l+1} + 2^{l-1} \leq x_4 < 2^{l+1} + 2^{l-1}$.

Proof of Lemma 3: Assume that the first digit the automaton prints is the least significant digit, and that it prints that digit before the input, n, has terminated. Since all inputs larger than n will follow the same path, the least significant digit for all $f_{n'}$, $n' > m$, will be the same, which contradicts claim 1.

Now assume that the first digit it outputs is the most significant. By the same argument, this would imply that all large Fibonacci Numbers have the same most significant digit, which contradicts claim 2 if $b > 2$. If $b = 2$, then the most significant digit will always be 1, but claim 3 shows that the second digit output cannot be fixed, and thus cannot appear before the input terminates.

Lemma 4 *There is no deterministic finite state or push-down automaton that, given any input n in base 1, can give an output f_n in base $b > 1$ without giving more than one digit of output before the input has terminated.*

Consider how such an automaton would work. Since all input is identical in all aspects except length, if the automaton travels down a given path for an input of n, it will travel down the same path and create an identical stack for any input $m > n$ up through the length of n. As there are a finite number of states, it must at some point enter a loop, and will continue around that loop until it reaches the end of the input. Thus, for sufficiently large n, the top of the stack will have become a repetitive cycle of elements, which can be made arbitrarily long.

For any state on the loop, there are an infinite number of n that will cause the automaton to be at that state when the input terminates. From that point on, only the stack will determine what states the automaton reaches. As the stack will always be an arbitrarily

large set of cycles down to a certain point, all of these inputs will cause the automaton to follow identical paths for an arbitrarily large amount of time, and it must therefore go into another loop for large enough n. Upon reaching the end of the cycles in the stack that were caused by the initial loop, the automaton will the follow a specific path of a set length, dependent at this point on the state it is in.

By Theorem 3 there are an infinite amount of Fibonacci Numbers that have an arbitrarily large number of zeroes as their least significant digits in their base b representation. The only way to produce an arbitrarily large number of zeroes in the output is to have at least one loop that prints only zeroes, and that must be one of the loops entered after the input has terminated. We pick an f_n as a multiple of b^m, m large enough such that the input n will force the automaton to circle the first loop several times, stop at some state when input terminates, enter a second path containing the loop that outputs only zeroes, circle that loop several times, and leave it at some state to follow the remaining part of the path.

Now consider all inputs larger than n that lead down the identical path, varying only in the number of times they circle around each loop. The original n, say n_1, will give an output of f_{n_1}. The next n will differ only in that it has gone around the first loop a few extra times, which does not produce any output, and then has gone around the second loop a constant number of times, thus adding a constant number c of zeroes and giving an output of $b^c f_{n_1}$. In general:

$$f_{n_{i+1}} = b^c f_{n_i},$$

and from there we can a drive a contradiction identical to the last part of the proof of Theorem 4.

Now a theorem giving us a lower bound on the space required to generate Fibonacci numbers follows directly from the four lemmas just presented.

Theorem 5 *There is no deterministic finite state or push-down automaton that can take any number n as input and produce the nth Fibonacci Number, f_n.*

The proof of Theorem 5 actually tells us a little more than what the theorem states. The problem the machine runs into in any case other then a base 1 input and base > 1 output is that for large enough n, the machine cannot count how many bits are in n, and thus does not know when to terminate. From this we can conclude that not only does the machine need random access memory, but it needs at least $\log n$ memory to count the bits in f_n.

Conjectured generalities of Theorems 4 and 5:

While theorems 4 and 5 give us lower bounds on the space need for the recognition and generation of Fibonacci numbers, it would be nice to generalize these theorems to a broader range of difference equations. The final contradiction reached in both the recognition and generation problems was that the Fibonacci relation has characteristic roots which are not rational powers of any integer. This leads us to our conjecture:

Conjecture 1 *Let g_n be any difference equation such that $g_n = \sum_{i=1}^{k} a_i g_{n-i}$. A finite state or push-down automaton that identifies a base b input as a member of that sequence, or a*

deterministic finite state or push-down automaton that generates g_n in base b, cannot exist if b is not a rational power of any of the characteristic roots of the relation.

In the case of Fibonacci numbers, there are two properties we used which we cannot assume in other difference equations. First, Theorem 3 allows us to "zero out" the right most bits of the Fibonacci numbers, thus allowing us to use the Pumping Lemma to show that there is a sequence of Fibonacci numbers that are constant multiples of each other. The second property is that in the closed-form expression for generating Fibonacci numbers, the coefficients of the characteristic roots have the same absolute value, and thus drop out in the algebraic manipulation.

Another case of the conjecture which we were able to prove for $b = 2$ was the relation: $g_n = 3g_{n-1}$, $g_0 = 1$ (whose solution is the sequence of powers of three). This recurrence relation has the same two properties that make the Fibonacci numbers manageable. Since there is only one root, we know that all coefficients of the roots have the same absolute value. As for zeroing out the low order bits, the following lemma helps.

Lemma 5 *For $n \geq 1$, $3^{2^n} \equiv 1 \, (mod \; 2^{n+2})$.*

Proof: Induct on n. This is trivial to verify for $n = 1$, so assume that $3^{2^n} \equiv 1 \, (mod \; 2^{n+2})$, and assume that $3^{2^{n+1}} \not\equiv 1 \, (mod \; 2^{n+3})$, hence $(3^{2^n})^2 \not\equiv 1 \, (mod \; 2^{n+3})$. Obviously $3^{2^n} \not\equiv 1 \, (mod \; 2^{n+3})$, so it must be congruent to $2^{n+2} + 1$. By substituting and expanding this congruence, we get $1 \not\equiv (3^{2^n})^2 \equiv 2^{2n+4} + 2^{n+3} + 1 \, (mod \; 2^{n+3})$, and thus $2^{2n+4} \not\equiv 0 \, (mod \; 2^{n+3})$. That is clearly a contradiction. □

Now, in the same way as before, we can prove that the base two powers of three cannot be recognized in finite space or with a stack.

Theorem 6 *There is no finite state or push-down automaton that can recognize powers of three in base 2.*

Proof: Let n_1 be the constant from the Pumping Lemma, n_2 be the constant from Ogden's Lemma, and $n = n_1 + n_2$. Then choose the number 3^{2^n}, which has a string of zeroes of length $n + 2$. As before, we mark the right most zero, and know that our pumped strings are now all zeroes. From this we have the relation:

$$g_{k_i} = 2^c(g_{k_{i-1}} - 1) + 1$$

for some integer c. By substitution and manipulation we get:

$$3^{k_i - k_{i-1}} - 2^c = \frac{1 - 2^c}{3^{k_i}}.$$

Since the right side converges to zero as i goes to infinity, we know that for all large enough i there exists a k such that $k_i - k_{i-1} = k$, and $3^k = 2^c$. Thus 2 is an rational power of 3, which is clearly not true. □

The proof for the generator would then follow roughly as before.

Time-Space Tradeoffs on Algorithms Computing Fibonacci Numbers:

It was our hope that in researching the various methods of proving time-space tradeoffs for different algorithms, we would come across something which we could apply to the Fibonacci algorithms. The most likely candidate seemed to be methods proving tradeoffs for integer multiplication algorithms.

One of the major papers in proving the tradeoff in multiplication algorithms [5] uses the Pebble Game to prove the tradeoff. The pebble game has been used in many papers, with many variations, but the basic idea seems to be the same. Take an algorithm and represent it with a directed graph. Then consider a set of pebbles, which will represent the available space. In any given turn, one may put exactly one pebble on either an input node (a node with in-degree 0), or any node for which all the parent nodes have been pebbled. One may remove any amount of pebbles at any point without a time cost. The object is to place a limited number of pebbles on each output node (nodes with out-degree 0) in a minimum amount of time. (Note that these outputs need only be pebbled once each, and not simultaneously.) Valiant [6] proved that every graph representing a bilinear multiplication algorithm contains a sub-graph in the form of an *n-superconcentrator*, and Tompa used a pebbling argument to show that the pebbling of any graph containing an n-superconcentrator sub-graph requires that Time·Space= $\Omega(n^2)$.

Could this technique be used to prove a similar time-space tradeoff for the Fibonacci number algorithms? No, we do not believe so. The pebbling technique used by Valiant and Tompa is only effective in proving lower bounds when the space to be used is smaller than the size of the input. But we have already shown that for the Fibonacci numbers, any generating algorithm must have at least as much space as the size of the input, so in terms of the pebbling argument we need at least $\log_2 n$ pebbles. Then, if we consider each output bit as a function of the input bits, we will need $\log_2 n$ moves to pebble the inputs and at most two moves to pebble each output (one move to pebble it and one move to re-pebble the inputs). As there are $\log_2 f_n \approx n \log_2 \lambda_0$ outputs, this will take us a total of $O(n)$ moves. This shows us that Time·Space= $\Omega(n \log n)$, which we already knew.

The Bit Patterns of Fibonacci Numbers:

After considering the material on space-time tradeoffs, we came to the conclusion that the only direction left to us was to look for patterns in the bit representations of the Fibonacci numbers. Using results proved by J. Holloway [3], it is possible to calculate the low order bits in linear time. If we were able to find a relation of the high order bits to the low order bits, it would than become possible to calculate the entire number in linear time.

When we examine the bits of of Fibonacci numbers, one pattern is immediately obvious: the least significant bit of each number, starting with f_0, repeatedly runs through the cycle 011. Holloway was able to extend this to a general theorem concerning the sequence of the bits. If B_i is the sequence of ith bits from f_n for n from 0 to ∞, then:

Theorem 7 *(Holloway) The cycle length of the sequence B_n is $3 \cdot 2^n$.*

In the proofs of space bounds, we relied heavily on the fact that for any m, we could find

an n such that $2^m \mid f_n$. Is it possible to predict where these Fibonacci numbers will occur? As it turns out, it is quite easy to find them, and to show that for any $m \geq 3$, there is an n such that $2^m \mid f_n$ but $2^{m+1} \not| f_n$.

Lemma 6 *For any $m \geq 3$, $n = 3 \cdot 2^{m-2}$ is the smallest n such that $2^m \mid f_n$. Furthermore, $2^{m+1} \not| f_n$.*

Proof: The Lemma follows from induction on m. If $m = 3$ then $n = 6$, and it is trivial to verify this as the base case. So pick an arbitrary $m \geq 3$, and choose the least k such that $2^m \mid f_k$. (Theorem 3 assures us that such a k exists.) The induction assumption will be that $k = 3 \cdot 2^{m-2}$ and that $2^{m+1} \not| f_k$. Then we pick the least n such that $2^{m+1} \mid f_n$, and prove the assertion holds for n.

First notice that $2^m \mid f_k$ and $2^m \mid f_n \Rightarrow 2^m \mid (f_k, f_n) = f_{(k,n)} \geq f_k$, since k is the minimal such Fibonacci number. Thus $(n, k) = k$, so $k \mid n$, and we can write n as kx where $x > 1$ (since $2^{m+1} \not| f_k$). Now if we can show that if $2^{m+1} \mid f_{2k}$, then we have $n = 2k = 3 \cdot 2^{m-1}$.

Assume $2^{m+1} \not| f_{2k}$. We know $f_{2k} \equiv 0 \,(mod\ 2^m)$ and $f_{2k} \not\equiv 0 \,(mod\ 2^{m+1})$, hence $f_{2k} \equiv 2^m \,(mod\ 2^{m+1})$ and we have:

$$f_{2k} \equiv f_k \,(mod\ 2^{m+1}). \tag{9}$$

Using (3) we can rearrange (9) to get:

$$f_k(f_{k-1} + f_{k+1} - 1) \equiv 0 \,(mod\ 2^{m+1}) \Rightarrow$$
$$2^m(f_{k-1} + f_{k+1} - 1) \equiv 0 \,(mod\ 2^{m+1}) \Rightarrow$$

$$f_{k-1} + f_{k+1} \equiv 1 \,(mod\ 2)$$

This implies that f_{k-1} and f_{k+1} are of different parity. However, $3 \mid k$ implies f_k is even, and f_{k-1} and f_{k+1} are both relatively prime to f_k, hence both odd. Thus they are of the same parity, and we have a contradiction. So we know that $2^{m+1} \mid f_{2k}$.

Now it only remains to be shown that $2^{m+2} \not| f_{2k}$. Assume otherwise. Then by applying (3) we get:

$$f_k(f_{k-1} + f_{k+1}) \equiv 0 \,(mod\ 2^{m+2}) \Rightarrow$$
$$2^m(f_{k-1} + f_{k+1}) \equiv 0 \,(2^{m+2}) \Rightarrow$$

$$f_{k-1} \equiv -f_{k+1} \,(mod\ 4).$$

Since we have already shown that both terms are odd, it follows that $f_{k-1} \not\equiv f_{k+1} \,(mod\ 4)$. However, $f_{k+1} = f_k + f_{k-1} \equiv 0 + f_{k-1} \,(mod\ 4)$, and this gives us our contradiction. So we have $2^{m+2} \not| f_n$. $\square$

Lemma 6 gives us the first Fibonacci number divisible by 2^m but not by 2^{m+1}. Can we expand this into a theorem giving the positions of all such Fibonacci numbers?

Theorem 8 *Let m be any positive integer greater than or equal to 3. Then m is the maximum power of 2 dividing f_n if and only if $n = 3 \cdot 2^{m-2}i$ for some positive odd integer i.*

Proof:
($\Leftarrow$) Let $n_i = 3 \cdot 2^{m-2}i = n_1 i$. We prove the assertion by induction on i. Our base case, $i = 1$, follows directly from Lemma 6. So assume the statement is true for i, and consider n_{i+2}. Since $(n_1, n_{i+2}) = (n_1, in_1 + 2n_1) = n_1$, we have $2^m \mid n_1 \mid n_{i+2}$. Assume that $2^{m+1} \mid n_{i+2}$. By applying (2) and (3) we get:

$$f_{n_{i+2}} = f_{in_1+2n_1} = f_{in_1-1}f_{2n_1} + f_{in_1}f_{2n_1+1} =$$

$$f_{in_1-1}f_{n_1}(f_{n_1+1} + f_{n_1-1}) + f_{in_1}f_{2n_1+1} \equiv 0 \,(mod\ 2^{m+1})$$

From Lemma 6 we know that $f_{n_1} \equiv 2^m \,(mod\ 2^{m+1})$, and by the induction assumption we know the same for f_{n_i}. So now we have:

$$2^m f_{in_1-1}(f_{n_1+1} + f_{n_1-1}) + 2^m f_{2n_1} \equiv 0 \,(mod\ 2^{m+1}) \Rightarrow$$

$$f_{n_1 i-1}(f_{n_1+1} + f_{n_1-1}) + f_{2n_1+1} \equiv 0 \,(mod\ 2)$$

However, since $(f_n, f_{n+1}) = 1$, we know all the terms in the above congruence are odd, so it reduces to $1 \equiv 0 \,(mod\ 2)$, and we have a contradiction. Thus $2^{m+1} \not| f_{n_{i+2}}$.

($\Rightarrow$) Let $k = 3 \cdot 2^{m-2}$, and choose some n such that $2^m \mid f_n$ and $2^{m+1} \not| f_n$. By Lemma 6, k is the minimum such Fibonacci number, so $n \geq k$. We know that $2^m \mid f_k$ and $2^m \mid f_n$, so we have $2^m \mid (f_k, f_n) = f_{(k,n)}$. Also, $2^{m+1} \not| f_{(k,n)}$, hence $f_{(k,n)} \geq f_k$ (since f_k is the minimal such Fibonacci number). Thus $k \mid n$, or $n = ki = 3 \cdot 2^{m-2}i$ for some positive integer i. Now we induct on i. For $i = 1$, Lemma 6 shows this works as our base case. Choose any odd i, and assume the statement holds for n_i. We know the statement holds for $i + 2$, so we need only show that it does not hold for $i + 1$. Assume it does. Then $f_{n_{i+1}} = f_{in_1+n_1} \equiv 2^m \,(mod\ 2^{m+1})$, so by (2): $f_{in_1-1}f_{n_1} + f_{in_1}f_{n_1+1} \equiv 2^m (mod\ 2^{m+1}) \Rightarrow f_{in_1-1} + f_{n_1+1} \equiv 1 \,(mod\ 2)$. This is clearly not possible, since f_{in_1-1} and f_{n_1+1} are both odd. □

While these results give some indication of what the low order bits look like at certain n, allowing us to quickly calculate the low order bits of other Fibonacci numbers, they do not help much with the high order bits. Another possibility considered was to show that certain blocks of bits could not occur within a Fibonacci number. For instance, if we look at only the first two bits of any of these numbers, all four possible values will occur at some point within the cycle. If we look at the first three bits, all eight possible values will occur except for four and six. In general, we have:

Conjecture 2 *Consider the first n bits of all Fibonacci numbers ranging from f_0 to $f_{3 \cdot 2^{n-1}}$. For any integer a such that $0 \leq a \leq 2^n - 1$, at least one of these blocks will be equal to a if and only if $a \not\equiv 4 \,(mod\ 8)$, $a \not\equiv 6 \,(mod\ 8)$, $a \not\equiv 10 \,(mod\ 16)$, and $a \not\equiv 18 \,(mod\ 32)$.*

The conjecture is obviously true in one direction, just by checking all the block patterns that occur from f_0 to $f_{3 \cdot 2^5}$. We have not proved the other direction, but have tested it up through $n = 14$.

Our last approach was to try the same method, but start the block at an arbitrary bit, not necessarily the first. This got us nowhere; it seems that for any block not including the least significant bit, every bit pattern that can occur will at some point during the cycle.

Conclusion:

The only solid results that have so far come out of this research are the bounds on space required to recognize and generate Fibonacci numbers: they are a context sensitive, but not context free, language that requires at lest $O(\log n)$ memory to generate. We also believe that the standard methods of proving lower bounds on time-space tradeoffs cannot be applied to the Fibonacci problem; if there is such a tradeoff, then the proof will most likely require a new method or model. The theorems we proved in looking for bit patterns may be helpful, but it is not yet obvious in what way. While it seems likely that there is an underlying pattern in the bits that will allow us to generate Fibonacci numbers in linear time, we still have no idea what that pattern is.

References

[1] B.J. Babb, Computing Fibonacci Numbers Rapidly (M.S. Paper, Oregon State University, Corvallis, Oregon, 1991)

[2] P. Cull and J.L. Holloway, Computing Fibonacci Numbers Quickly, *Inform. Process. Lett.* **32** (1989), 143-149.

[3] J. Holloway, Computing Fibonacci Numbers Quickly, (M.S. Thesis, Oregon State University, Corvallis, Oregon, 1988)

[4] J.E. Hopcraft and J.D. Ullman, *Introduction to Automata Theory, Languages, and Computation* (Addison-Wesley Publishing ., Menlo Park, California, 1979).

[5] Martin Tompa, Space-Time Tradeoffs for Computing Functions Using Connectivity Properties of Their Circuits, *J. Comput. System Sci.* **20** (1980), 118-132.

[6] L.G. Valiant, Graph-Theoretic Properties in Computational Complexity, *J. of Comput. System Sci.* **13** (1976), 278-285.

[7] N.N. Vorvob'ev *Fibonacci Numbers*, (Blaisdell Publishing Company, New York, 1961).