The well-known Fibonacci numbers are defined recursively in the box to the right. The sequence of Fibonacci starts off like this:

**Fibonacci numbers**

F(0) = 0. F(1) = 1.

For n > 1, F(n) = F(n-1) + F(n-2)

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, …

This discussion of the Fibonacci numbers provides an interesting look at the development of algorithms and data structures, with interesting historical tidbits. Moreover, Fibonacci numbers are connected with numbers called the golden ratio and golden angle, they have connections with architecture, and they appear in various ways in nature. Another pdf file in this JavaHyperText treats discusses these topics.

Some of this material is taken from en.wikipedia.org/wiki/Fibonacci\_number.

Many people think that Fibonacci numbers first appeared in Fibonacci’s historic book on arithmetic, *Liber Abaci* (*The Book of Calculation*), in 1202. Actually, Fibonacci was not his real name! That name was given to him by a writer well after his death. When he was living, he was known as Leonardo of Pisa. Moreover, the Fibonacci numbers were used and discussed in ancient Sanskrit texts in India as early as 450BC–200BC. A paper by Parmanand Singh, written in 1985, on the early introduction of Fibonacci numbers in Sanskrit texts can be found in the JavaHyperText entry for Fibonacci.

**A naïve implementation of Fibonacci numbers**

**/\*\*** = F(n).

\* Precondition: n ≥ 0. \*/

public static int fib(int n) {

if (n <= 1) return n;

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

}

Function fib to the right is the obvious translation of the definition of function F into Java. It is also the worst way to compute it. Consider computing fib(15), as shown in the tree to the right. That call requires calling fib(13) and fib(14). These two calls then require calling fib(13), fib(12) twice, and fib(11). You can imagine the next level. In total,

fib(13) is called 2 times,  
fib(12) is called 3 times,  
fib(11) is called 5 times,  
fib(10) is called 8 times

fib(15)

fib(14)

fib(13)

fib(13)

fib(12)

fib(12)

fib(11)

Hey, the number of times each is called forms the Fibonacci sequence!

We prove that the time complexity of fib(n) is in O(2n). We first write a (recursive) function that gives an upper bound on the number of basic steps taken in computing fib(n). Constant c is an upper bound on the number taken in the case n ≤ 1 and the basic steps needed besides the recursive calls in the case n ≥ 2.

Theorem. fib(n) ∈ O(2n).

Proof. We prove that f(n) ≤ c 2n for n ≥ 0, where c is given in the definition of f(n) to the left. The proof is by induction on n. We have:

f(0) = c ≤ c 20 (since 20 = 1).

f(1) = c ≤ c 21 (since 21 = 2).

Assume that n ≥ 2. Assume f(k) ≤ c 2k for k < n holds. We prove it f(n) ≤ c 2n. We start with the definition of f(n):

c + f(n-1) + f(n-2)

= <inductive hypotheses>

c + a2n-1 + a2 n-2

= <arithmetic>

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

≤ <for n ≥ 2, 1+2 n-2 ≤ 2 n-1>

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

= <arithmetic>

c 2n

f(0) = c

f(1) = c

For n ≥ 2, f(n) = c + f(n–1) + f(n–2)

Then, to the left is the proof by mathematical induction that this function is in O(2n).