1 Problem: Fibonacci Numbers

Definition 1.1: Fibonacci Numbers

The **Fibonacci numbers** are the sequence defined by:

$$F_0 = F_1 = 1$$

$$F_n = F_{n-1} + F_{n-2} \text{ for } n \geq 2$$

1.1 Naive Algorithm

There is an easy recursive algorithm:

```
Fib(n)
    if n <= 1
        Return 1
    Else
        Return Fib(n - 1) + Fib(n - 2)
```

1.2 Runtime

Let $T(n)$ be the number of lines of code `Fib`(n) needs to execute.

- The `if` and `else` statements make up 2 lines of code.
- Otherwise, we need to run $1 + T(n-1) + T(n-2)$ lines.

So:

$$T(n) = \begin{cases} 2 & n \leq 1 \\ T(n-1) + T(n-2) + 3 & \text{Otherwise} \end{cases}$$

Here, if we want to compute `Fib`(100), the runtime would be:

$$T(100) \approx 2.87 \cdot 10^{21}$$

At a billion lines of code per second, this would take over 90,000 years to run.

1.3 Why So Slow?

This algorithm, in particular, has too many (redundant) recursive calls. For example:

$$f(5) = f(4) + f(3)$$

$$f(4) = f(3) + f(2)$$

$$f(3) = f(2) + f(1)$$

Already, there are some repeated calls. How can we make this more efficient?

1.4 Improving this Algorithm

First, we avoid recomputing things. When we do this by hand, we often already have the last two numbers. So, we can use an array to store the nth Fibonacci number.

1.5 Improved Algorithm

If we were to simulate finding the Fibonacci numbers by hand, we would have an algorithm similar to:

```
Fib2(n)
    Initialize A[0..n]
    A[0] = A[1] = 1
    For k = 2 to n
        A[k] = A[k - 1] + A[k - 2]
    Return A[n]
```

In the first two lines (initializing the array and setting initial values), there are 2 lines. In the `for` loop, we run $2(n-1)$ lines. Finally, we run 1 line of code. So, we have the final result of:

$$T(n) = 2n + 1$$

With this new algorithm, we have $T(100) = 201$. So, this is easily runnable on almost any computer.

Essentially, the power of algorithms is:

> Sometimes, the right algorithm is the difference between something working and not finishing in your lifetime.