CS 161 **Pre-Lecture Exercise for Lecture 12**
Autumn 2017 **Date:** November 6, 2017

Pre-lecture exercises will not be collected for credit. However, you will get more out of each lecture if you do them, and they will be referenced during lecture. We recommend **writing out** your answers to pre-lecture exercises before class. Pre-lecture exercises usually should not take you more than 20 minutes.

Consider the Fibonacci numbers, defined by

$$F(0) = F(1) = 1$$

and

$$F(n) = F(n-1) + F(n-2).$$

For example, the first several Fibonacci numbers are:

$$1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, ...$$

Consider the following divide-and-conquer algorithm to compute Fibonacci numbers.

```
def Fibonacci(n):
    if n == 0 or n == 1:
        return 1
    return Fibonacci(n-1) + Fibonacci(n-2)
```

1. Is this algorithm correct?
2. What is the running time of this algorithm? You don't need to find it exactly, but is it $O(n)$? $O(n^2)$? $O(n^3)$? $O(n^c)$ for any constant c?
3. How could you make this algorithm better?

Solution.

1. Yes, the algorithm is correct.
2. The running time is exponential in n. This satisfies the recurrence relation

$$T(n) = T(n-1) + T(n-2) + O(1). \tag{1}$$

   In particular, for $n \geq 2$, we have

$$T(n) \geq T(n-1) + T(n-2),$$

   and so

$$T(n) = \Omega(F(n)),$$

   where $F(n)$ is again the n'th Fibonacci number.

   In fact, this scales like ϕ^n, where ϕ is the golden ratio. That's a bit tricky to see (see CLRS for an overview) but here's an easy way to see that (1) grows exponentially: for $n \geq 2$, we have

$$T(n) \geq 2T(n-2),$$

using the fact that $T(n-1) \geq T(n-2)$. Thus,

$$\begin{aligned}
T(n) &\geq 2T(n-2) \\
&\geq 4T(n-4) \\
&\geq 8T(n-6) \\
&\cdots \\
&\geq 2^j T(n-2j) \\
&\cdots \\
&\geq 2^{\lfloor n/2 \rfloor} T(n - 2\lfloor n/2 \rfloor) \\
&= 2^{\Omega(n)}
\end{aligned}$$

So this grows exponentially.

3. We can make this algorithm better with memoization or dynamic programming! See Lecture 12 for details.