More on the recurrences from class 2

Fall 2013

In class we saw the recurrence for the Fibonacci numbers, $F_n = F_{n-1} + F_{n-2}$, and the related recurrence for the *number of function calls in a recursive calculation of the Fibonacci numbers*, $T_n = 1 + T_{n-1} + T_{n-2}$. The base cases for these recurrences are: $F_0 = 0, F_1 = 1$, and $T_0 = 1, T_1 = 1$.

These notes will analyze these recurrences and give hints about general tools for similar recurrences. CS157 will not emphasize recurrences because you should have already seen this material in the prerequisite classes. However, homeworks and exams may depend on your familiarity with recurrences.

General Principles

When studying algorithms, the recurrences we solve will generally be related to the run-time of an algorithm. And we generally only care about the "rate of growth" of the run time, namely, we will use "big-O" notation to describe the performance of algorithms.

Because of these two factors, the base cases of most recursions will not matter, which is an important point to keep in mind. Making the base case of your recursive algorithm 10 times slower may slow down your algorithm up to 10 times (or it may not). However, such a change will never change the "rate of growth" of your algorithm's run-time. Making the base case 10 times faster will never change a quadratic-time algorithm to a linear-time algorithm. For this reason you should get in the habit of reasoning only about the inductive step, when analyzing algorithm performance.

Of course, in a formal proof you must take into account the base cases. But when you are solving a problem, we *suggest* that you pay way more attention to the inductive/recursive step.

These notes start out informally, and end with short rigorous proofs. Make sure to check out the last section of the notes.

Comparing F and T

Which grows faster, F or T? It seems like T should win because each recursive invocation adds an extra 1, and there are a lot of recursive invocations!

However, consider the sequence T^{+2}, which we will define here to be exactly 2 more than the corresponding value of T, namely, $T^{+2}_n = T_n + 2$. Clearly T and T^{+2} grow at the same rate since they are so close to each other. (Can you prove that $T_n = O(T_n + 2)$ and that $T_n + 2 = O(T_n)$, for any sequence of positive integers T? This is an opportunity to remind yourself of the definition of "big-O".) However, think of how one would define T^{+2} recursively. Spend a minute deriving its recursive definition, without referring to T.

Did you find that T^{+2} satisfies the recursion $T^{+2}_n = -1 + T^{+2}_{n-1} + T^{+2}_{n-1}$? This is a bit surprising: the two sequences T and $T + 2$ satisfy variants of the Fibonacci recurrence with a "+1" added in each case for T, and a "-1" added in each case for T^{+2}. This is the *opposite* of what one would expect. Also, it is surprising that changing the recurrence like this has so *little* change on the rate

of growth of the sequence. Explicitly, the "+1" and "−1" have *no* change on the rate of growth, since adding 2 converts a sequence that satisfies the first recurrence into a sequence that satisfies the second. (Try this out by hand if you are confused; again, this is the opposite of what one might expect.)

The rate of growth of F and T

In class we presented upper and lower bounds on F and T, though there was some confusion about how the base cases fit into the derivation.

One way to get intuition about the bounds for T, without math, is to analyze the structure of the recursive call tree for the Fibonacci recurrence. To compute F_n, we recursively call the function twice, to compute F_{n-1} and F_{n-2}. This fact gives us rigorous bounds on the structure of the recursion tree: the longest path down the tree must have length n, since at each recursive call we subtract either 1 or 2 from n, and thus we can do this at most n times. Conversely, each path on the tree must have length *at least* $(n-1)/2$, since we must subtract 1 or 2 from n at least that many times to get down to the base cases of 0 or 1. Since the tree is binary, the number of leaves must be at most 2^n, two to the power of the maximum depth, and at least $2^{(n-1)/2}$, two to the power of the minimum depth. These are, roughly, the two bounds derived in class. Both of these bounds are rather loose, because relatively few leaves are at depth n or depth $(n-1)/2$, and most are in between.

At this stage, we know enough to try a "guess and check" solution of the recurrence. Our guess is going to be that $F_n = x^n$, for some positive value x. Stated differently, *we guess that Fibonacci numbers grow exponentially.* As we have seen, x should be between $2^{1/2} = \sqrt{2}$ and 2. How should we solve for x? Think for a minute here.

Plug our guess into the recurrence relation, to get $x^n = x^{n-1} + x^{n-2}$. Dividing through by x^{n-2} yields the quadratic equation $x^2 = x + 1$, which has as its solution the "golden ratio", denoted $\phi = \frac{1+\sqrt{5}}{2} \approx 1.618$.

We have been ignoring the base cases of the recursion so far, and indeed they cause problems with our guess of $F_n = \phi^n$. If this were a math course, the professor would emphasize how to solve for F_n exactly, which would yield the somewhat odd expression that $F_n = \frac{1}{\sqrt{5}}\phi^n - \frac{1}{\sqrt{5}}(-1/\phi)^n$. But computer science analysis does not need this level of precision, and in fact for most recurrences you will encounter, exact solutions like this are *impossible* to find. Instead, there are simple and powerful ways to tightly *bound* the solution to recurrences, without solving them, which is great.

Precise bounds on F and T

The following proofs are all simple variations on the obvious induction. Keep in mind which variations are needed, and why without these variations the proofs would fail.

Claim: $F_n \leq \phi^n$. **Proof:** The base cases of $F_0 = 0$ and $F_1 = 1$ clearly satisfy the claim, since $\phi^0 = 1$ and $\phi^1 > 1$. Further, since by definition $\phi^2 = \phi + 1$, the inductive step follows from $F_n = F_{n-1} + F_{n-2} \leq \phi^{n-1} + \phi^{n-2} = \phi^n$, as desired.

(Note that the above proof also works for any multiple $c \cdot \phi^n$ where c is your favorite number larger

than 1.)

Claim: $F_n \geq \frac{1}{3}\phi^n$, for $n > 0$. **Proof:** We use base cases $F_1 = 1$ and $F_2 = 1$, which clearly satisfy the claim, since ϕ^1 and ϕ^2 can both be checked to be less than 3. Further, since by definition $\phi^2 = \phi + 1$, the inductive step follows from $F_n = F_{n-1} + F_{n-2} \geq \frac{1}{3}\left(\phi^{n-1} + \phi^{n-2}\right) = \frac{1}{3}\phi^n$, as desired.

(Note that the fraction $\frac{1}{3}$ could be changed to anything less than $1/\phi^2$.)

Claim: $T_n \leq 2\phi^n - 1$. **Proof:** The base cases of $T_0 = 1$ and $T_1 = 1$ clearly satisfy the claim, since ϕ^0 and ϕ^1 are both at least 1. Further, since by definition $\phi^2 = \phi + 1$, the inductive step follows from $T_n = 1 + T_{n-1} + T_{n-2} \leq 2\phi^{n-1} + 2\phi^{n-2} + 1 - 1 - 1 = 2\phi^n - 1$, as desired.

(Note that the constants 2 and 1 in the induction hypothesis can both be made bigger, though not every pair works. In particular, though, it is crucial that the 1 be *subtracted* in the induction hypothesis; adding makes the induction fail.)

Claim: $T_n \geq \frac{1}{2}\phi^n$. **Proof:** We use base cases $T_0 = 1$ and $T_1 = 1$, which clearly satisfy the claim, since ϕ^0 and ϕ^1 are both less than 2. Further, since by definition $\phi^2 = \phi + 1$, the inductive step follows from $T_n = 1 + T_{n-1} + T_{n-2} \geq \frac{1}{2}\left(\phi^{n-1} + \phi^{n-2}\right) = \frac{1}{2}\phi^n$, as desired.

(Note that we threw out the 1 in the above proof because it only helped make the inequality stronger. That was easy!)

In total, these four proofs yield that both F and T grow at *exactly* the same rate as ϕ^n, thus formalizing the more intuitive arguments of previous sections.

The general strategy for coming up with proofs like the above is: first ignore everything you don't want to deal with to come up with a guess for a bound (like we ignored the $+1$ in the definition of T, to guess that it behaves like F). Then, ignoring the base cases, make sure the inductive step works for your guess (and modify your guess slightly as needed if it *almost* works). Next, modify your induction hypothesis as needed to make it work for the base cases and the inductive step.

Keep in mind this "guess and check" style of asymptotic analysis as you go through the algorithms in this course. The proofs here are so short that you can very easily mentally check whether they seem to work out, which is a habit you should get into as your perspective on algorithms becomes more sophisticated.