Asymptotic Growth (3)

Michael L. Littman

September 9th, 1997

1 ANNOUNCEMENTS

1.1 Updates to Web Pages

Here are some recent changes:

- I've put revised versions of the slides on the web, along with links to the postscript of the handouts.
- I've put Hai's information onto the course home page.
- I've put up my current plan for office hours: 1:00pm - 2:00pm Wednesday, 2:00pm - 3:00pm Thursday, 30 minutes after each class, any time for quick questions, other times by request (send email).
- I added a set of links on the course home page to animations, so they are easier to find and play with. We're looking into making them more interactive.
- How's HW1 going? I put the wrong date up (Sep. 12th is a Friday, not a Thursday), but I'll stick with it. The due time is still 10:55am, though.

2 EUCLID'S ALGORITHM

2.1 Euclid's Algorithm

Recall that we are interested in computing the GCD of two non-negative integers: the largest number that divides both numbers evenly.

Euclid's algorithm computes the GCD of a and b by returning a if $b = 0$ and otherwise returning the GCD of b and a mod b.

Here is an animation to illustrate this process and also to help make it clear why it computes the correct answer.

2.2 Efficiency Analysis

Can express run time as a recurrence relation: $T(a,0) = 1$, $T(a,b) = 1 + T(b, a \bmod b)$ if $b > 0$.
Recall that we are interested in computing the *worst-case* running time for this process. (Why?)
Some of the obvious candidates for this don't work out. For example, how big is $a \bmod b$ compared to b? Neither of these hold for all a and b:

$$\begin{aligned} a \bmod b &\not\leq b/2 \\ a \bmod b &\not\geq b/2 \end{aligned}$$

2.3 Classic Sequence

k	0	1	2	3	4	5	6	7	8	9	10	11	12
$F(k)$	0	1	1	2	3	5	8	13	21	34	55	89	144

$$\begin{aligned} F(0) &= 0 \\ F(1) &= 1 \\ F(k) &= F(k-1) + F(k-2), \text{ for } k > 1 \end{aligned}$$

Do you recognize it?

2.4 Rate of Growth of F

Note: $F(k) > F(k-1)$ for all $k \geq 3$, and $F(k) \geq 0$, for all $k \geq 0$.
Theorem: For $k \geq 4$, $2F(k-2) \leq F(k) < 2F(k-1)$.

- $2F(k-2) \leq 2F(k-2) + F(k-3) = F(k-2) + F(k-1) = F(k)$.
- $F(k) = F(k-1) + F(k-2) < F(k-1) + F(k-1) = 2F(k-1)$.

2.5 A Corollary

Corollary: For $k \geq 4$, $2^{\lfloor (k-1)/2 \rfloor} \leq F(k) < 2^k$.
Proof comes from the fact that:

- $T(0) = 1$, $T(k) = 2T(k-1)$ is $T(k) = 2^k$.
- $T(0) = 1$, $T(1) = \sqrt{2}$, $T(k) = 2T(k-2)$ is $T(k) = 2^{k/2}$.

k	0	1	2	3	4	5	6	7	8	9	10	11	12
$2^{\lfloor (k-1)/2 \rfloor}$	0.5	1	1	2	2	4	4	8	8	16	16	32	32
$F(k)$	0	1	1	2	3	5	8	13	21	34	55	89	144
2^k	1	2	4	8	16	32	64	128	256	512	1024	2048	4096

2.6 Inverse Fibonacci

Define $F^{-1}(x)$ to be the value of k such that

$$F(k-1) < x \leq F(k).$$

We'll call F^{-1} the "inverse Fibonacci" function.
Now, how fast does $F^{-1}(x)$ grow?
For $x \geq 3$, $\log(x) \leq F^{-1}(x) \leq 2\log(x) + 1$.

2.7 Fibonacci Meets Euclid

What happens when we apply Euclid's algorithm to a consecutive pair of Fibonacci numbers, $F(k+2)$ and $F(k+1)$?

- Well, if $k = 1$, then the algorithm returns the answer 1 after one recursive call.
- Otherwise, we recurse on $F(k+1)$ and $F(k+2) \bmod F(k+1)$. We know that $F(k+1) > 1/2F(k+2)$, so $F(k+2) \bmod F(k+1) = F(k+2) - F(k+1) = F(k)$. So the next pair Euclid examines is the Fibonacci numbers $F(k+1)$ and $F(k)$.

Example: EUCLID(144,89) = EUCLID(89,55).
How many recursive calls? Exactly k!
Thus, the worst-case time to compute EUCLID(a,b) can be at least $F^{-1}(a) - 2$ (lower bound on worst case).

2.8 Fibonacci Upper Bound

But the Fibonacci sequence also gives us an upper bound on the worst case.
Theorem: If EUCLID(a, b) takes $k \geq 1$ recursive calls, then $a \geq F(k+2)$ and $b \geq F(k+1)$.

- $k = 1$: To have at least one recursive call, we must have $b \geq 1 = F(k+1)$ and $a \geq 2 = F(k+2)$.
- $k \geq 2$: Inductive hypothesis implies EUCLID$(b, a \bmod b)$ takes $k-1$ recursive calls and $b \geq F(k+1)$ and $a \bmod b \geq F(k)$.

$$\begin{aligned} a &\geq b + (a \bmod b) \\ &\geq F(k+1) + F(k) \\ &= F(k+2) \end{aligned}$$

The first step follows from the fact that $a \bmod b$ is what's left over after removing one or more copies of b from a. So one copy of b plus $a \bmod b$ can't be any bigger than a.

Therefore, we can't take any more than k recursive calls to compute the GCD unless a is bigger than $F(k+2)$. That means that $F^{-1}(a) - 2$ is also an upper bound on the number of iterations needed.

2.9 Summary

Consecutive Fibonacci numbers are the worst-case input for Euclid's algorithm.
Thus, the worst-case run time for Euclid is the inverse Fibonacci function for a (minus 2), which itself is bounded between $\log(a) - 2$ and $2\log(a) - 1$.
We will next develop a (less precise, but more convenient) short-hand notation for this: The running time of Euclid is $\Theta(\log(a))$.

3 ASYMPTOTIC GROWTH

3.1 Comparing Algorithms

We want to compare algorithms to decide which we prefer.

1. But there are many dimensions along which we might compare. So we decide that what really matters is *run time*.

2. But run time depends on input. So we decide that what really matters is run time as a function of *input size*.

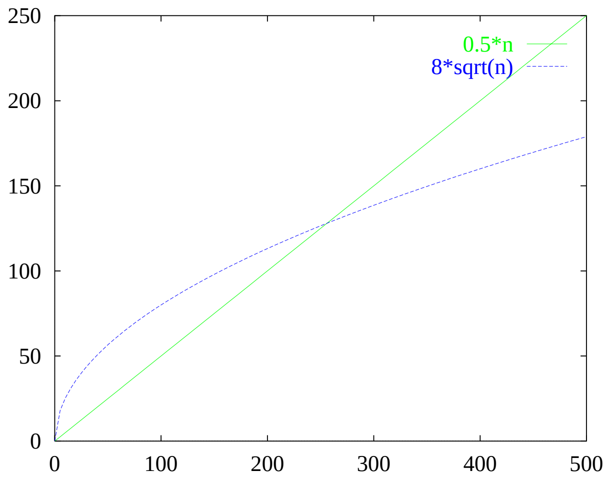
3. But this can be hard to characterize. So we decide that what really matters is *worst-case* run time as a function of input size.

4. But one algorithm might be better on small inputs and the other on large inputs. So we decide that what really matters is worst-case run time as a function of input size for *large inputs*.

5. But this can still be quite hard to determine precisely. So we decide that what really matters is worst-case run time as a function of input size for large inputs, *ignoring constant factors*.

3.2 Asymptotic Growth

This leads us directly (!) to a notion of asymptotic growth.
Function $g(n)$ is asymptotically bigger than function $f(n)$ if, for any scaling factor $c > 0$, there's some threshold n_0 such that $g(n) > cf(n)$ for all $n \geq n_0$.



3.3 Example

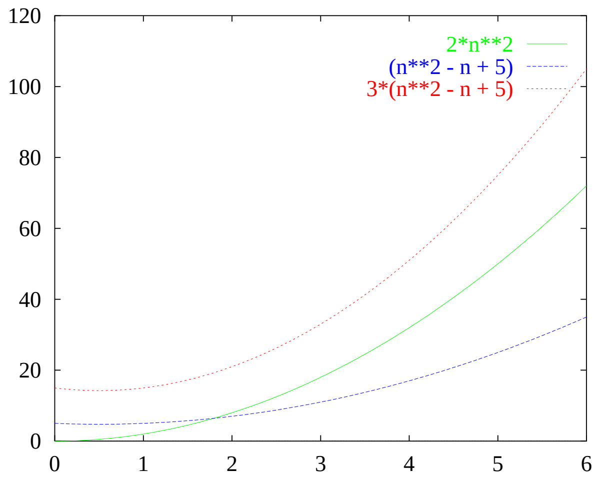
By this definition, $g(n) = 1/2n$ is asymptotically bigger than $f(n) = 8\sqrt{n}$. Why? Well, give me any scaling factor c.

$$
\begin{aligned}
g(n_0) &> cf(n_0) \\
1/2n_0 &> c(8\sqrt{n_0}) \\
n_0 &> 16c\sqrt{n_0} \\
n_0/\sqrt{n_0} &> 16c \\
\sqrt{n_0} &> 16c \\
n_0 &> 256c^2
\end{aligned}
$$

So any n_0 at least that big will satisfy the conditions of the definition.

3.4 Counterexample

On the other hand, $g(n) = 2n^2$ is not asymptotically bigger than $f(n) = n^2 - n + 5$.
Take the constant $c = 3$. We have $g(n) < cf(n)$ for all n.
On the other hand, $f(n)$ is not asymptotically bigger than $g(n)$ either.



3.5 Big Theta

The notion of asymptotic growth can be used to partition the space of functions.
Two functions $f(n)$ and $g(n)$ are "equal" if neither is asymptotically bigger than the other.
We write this as $f(n) = \Theta(g(n))$.
Formally, $f(n) = \Theta(g(n))$ if and only if there exists scaling factors c_1 and c_2 and threshold n_0 such that $0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)$ for all $n \geq n_0$.
It's a an $f(n)$ sandwich on $g(n)$ bread!

3.6 Examples

- $1/2n = \Theta(n)$
- $8\sqrt{n} = \Theta(\sqrt{n})$
- $n^2 - n + 5 = \Theta(n^2) = 2n^2$
- $F^{-1}(n) = \Theta(\log(n))$
- $7n^4 + 2n^3 - 4n \log n + \sqrt{n} = \Theta(n^4)$
- $\pi^2 = \Theta(1)$

3.7 Base of Logs

$\log_{10}(n) = \Theta(\log_2(n))$

$$\begin{aligned} f(n) &= \log_{10}(n) \\ 10^{f(n)} &= n \\ \log_2(10^{f(n)}) &= \log_2(n) \\ f(n)\log_2(10) &= \log_2(n) \\ f(n) &= \log_2(n)/\log_2(10) \end{aligned}$$

So for $c_1 = (1/2)/\log_2(10)$, $c_2 = 2/\log_2(10)$, $n_0 > 0$,
$0 \leq c_1 \log_2(n) \leq f(n) \leq c_2 \log_2(n)$ for all $n \geq n_0$.

3.8 Complete Set

Big Theta gives us a notion of asymptotic equality. We also have symbols for all of the other comparative notions as well.

$$\begin{array}{rcl} f(n) = o(g(n)) & \approx & f(n) < g(n) \\ f(n) = O(g(n)) & \approx & f(n) \leq g(n) \\ f(n) = \Theta(g(n)) & \approx & f(n) = g(n) \\ f(n) = \Omega(g(n)) & \approx & f(n) \geq g(n) \\ f(n) = \omega(g(n)) & \approx & f(n) > g(n) \end{array}$$

3.9 Big O

Everyone has probably heard of big-O.
When we say the running time of a sorting algorithm is $O(n^2)$, we are giving an asymptotic upper bound on the running time for all inputs of size n.
This is often what we want. However, saying that the running time is $\Theta(n^2)$ is often a more precise way of saying the same thing.
Note that $f(n) = \Theta(g(n))$ implies $f(n) = O(g(n))$.

3.10 Some Technical Details

The "big" and "little" functions behave almost exactly like their standard real-number counterparts: transitivity, reflexivity, symmetry, and transpose symmetry.
However, one standard property does not hold: trichotomy. There are pairs of functions $f(n)$ and $g(n)$ such that none of $f(n) = o(g(n))$, $f(n) = \Theta(g(n))$, nor $f(n) = \omega(g(n))$ holds. The book gives an example of $n^{1+\sin n}$ which can't be compared directly to n because it often acts like n^2.

3.11 Applying to Algorithms

Also, in using these functions to describe algorithms, there are a few things to note.

- If we say the algorithm's running time is $O(n^2)$, it means its worst-case running time is $O(n^2)$, but its best case might be better.
- If we say the algorithm's running time is $\Omega(n)$, it means its best-case running time is $\Omega(n)$, but its worst-case might be worse.
- Therefore, an algorithm with running time $\Theta(n \log n)$ takes $\Theta(n \log n)$ on all inputs of size n.

3.12 Math in Big Theta World

Here are some examples of reductions we can do when we're inside a big theta.

- $1 + 1 = 1$
- $10n + 6 = n$
- $2^{n+1} = 2^n$
- $5\sqrt{n}\log n + 21\log^5 n = \sqrt{n}\log n$
- $2^{2n} = 4^n$

In general, we'd like to report functions in their simplest possible form.

3.13 An Example with Proof

$6n\log n + \sqrt{n}\log^2 n = \Theta(n\log n)$.
Need to find n_0, c_1, and c_2, such that
$c_1 n\log n \le 6n\log n + \sqrt{n}\log^2 n \le c_2 n\log n$
$c_1 = 5$, $c_2 = 7$, $n_0 = ...$

$$\begin{aligned}
5n_0\log n_0 &\le 6n_0\log n_0 + \sqrt{n_0}\log^2 n_0 \\
5\log n_0 &\le 6\log n_0 + \sqrt{n_0}/n_0\log^2 n_0, n_0 > 0 \\
5\log n_0 &\le 6\log n_0 + \log^2 n_0 \\
5 &\le 6 + \log n_0, n_0 > 1 \\
\log n_0 &\ge 1 \\
n_0 &\ge 2
\end{aligned}$$

$$\begin{aligned}
6n_0\log n_0 + \sqrt{n_0}\log^2 n_0 &\le 7n_0\log n_0 \\
6n_0 + \sqrt{n_0}\log n_0 &\le 7n_0 \\
\sqrt{n_0}\log n_0 &\le n_0 \\
\log n_0 &\le \sqrt{n_0} \\
n_0 &> 0
\end{aligned}$$

So, $n_0 = 3$ suffices.

4 HOMEWORK

4.1 Homework 1

Due September 12th:

- 1.1 Send me (mlittman@cs.duke.edu) your email address (right away).
- 1.2 Multiply 948×364 using the Russian Peasant's Algorithm.
- 1.3 For Naive multiplication, carry out algorithm development Step 2.
- 1.4 Write a recursive algorithm [Step 1] for computing a^b (where both a and b are positive integers). Prove that it is correct [Step 2]. Analyze the number of recursive calls it makes [Step 3] (it should be logarithmic). You can assume you have built-in (unit-time) multiplication and bit shifting, but not exponentiation or general division.
- CLR Problem 2-6 (pg 40).

4.2 Homework 2

Background: Ch. 33.2, 2.1.
Due September 18th:

- 2.1 CLR Exercise 33.2-4 (pg 813).
- 2.2 CLR Exercise 2.1-2 (pg 31). Use the definition of big Theta, not the normal-form rules.