

CSE 101 Class Notes

January 27, 2005

Today: Euclid's GCD algorithm's time analysis

- What is a step?
- What is time analysis?

“Constant” Time and Steps

An operation or algorithm is *constant time* if its time does not depend on the input size. Typically, we assume the following are constant time:

- reading/writing memory
- boolean logic
- parsing/interpreting the program
- recursion
- arithmetic and comparison (not always true)
- random number generation (but only in these notes)

Note that these all represent assumptions, which may be invalid in some situations, e.g. memory hierarchy boundary crossings, enormous numbers.

On this model, the input “size” is the number of integers, and each GCD iteration runs in constant time. However, if the time for an arithmetic operation depends on the operands' sizes, then we need to be more precise. In particular, how do we compute the mod?

Grade school operations on n - and m -bit numbers, $n \geq m$:

- Addition, subtraction: $\Theta(n)$
- Multiplication: $O(nm)$ (note that we can do better and will show how in later classes)
- Division: How long is the quotient? At most $n - m + 1$ bits. Since we do one one-bit-multiply-and-subtract for each quotient digit, the algorithm runs in $O(m(n - m - 1))$. Note this is fast when $m \ll n$ or $m \approx n$, (when it is around $O(n)$) and slow when $m \approx n/2$ (when it is $O(n^2)$).

Let (x_t, y_t) = values of (x, y) after t loops, and (n_t, m_t) = binary length of (x_t, y_t) .

GCD: Worst-Case Time

Lemma: $y_{t+2} \leq x_t/2$, so $n_{t+2} \leq n_t - 1$.

(proved last lecture) It follows that the number of iterations is at most $2n$, since every 2 iterations, the binary length of y drops by 1, and when this length reaches 0, the algorithm terminates.

Given the above, each iteration takes $O(n_t^2) = O(n^2)$ to perform its mod operation, and there are $O(n)$ iterations, so the entire algorithm is $O(n^3)$. Note these are all O s, not Θ s: is GCD $\Theta(n^3)$?

One way to show this would be to come up with a case where GCD actually requires $O(n^3)$ time. Let's look for when the algorithm is slow.

Loops required	x	y
1	1	2
2	2	3
3	3	5
4	5	8

What's this? A Fibonacci sequence. (*note: if you are ever taking a fill-in-the-blank test and have no idea of the answer, Fibonacci is a good guess.*)

For an iteration with $x = F_n, y = F_{n-1}$, $x \bmod y = F_{n-2}$, so there will be n iterations for F_n . Also, since $F_n \leq 2^n$, F_n is an n -bit number, and we know GCD takes no more than n iterations, i.e. the number of iterations is $\Theta(n)$.

Since F_n and F_{n-1} differ in length by at most 1 bit, grade-school division will take $\Theta(n)$ time, so GCD requires $O(n^2)$ time for adjacent Fibonacci numbers.

This isn't as bad as our $O(n^3)$ upper-bound above, so we suspect that we may be able to come up with a better upper bound.

Example: Amortization

```

1   i = 0;
2   while i < n {
3       x = rand(1, n - i);
4       i += x;
5       O(x);
6   }
```

Worst-case time for line 5 is $O(n)$, and the loop repeats $O(n)$ time in the worst case, so the algorithm could take $O(n^2)$. But this is too pessimistic – we know the total time for all iterations' line 5 is $\sum_j \Theta(x_j) = \Theta(n)$, and the loop repeats $\leq n$ times, so the algorithm is actually $\Theta(n)$.

Amortization for GCD

Here we find a tighter bound for GCD. The t -th mod by long division takes

$$\Theta(n_t(m_t - n_t + 1)) \leq n(m_t - n_t + 1) = n(n_{t-1} - n_t + 1)$$

So the total time is no greater than

$$\begin{aligned}\sum_t n(n_{t-1} - n_t + 1) &= n(n - n_1 + 1) + n(n_1 - n_2 + 1) + \dots \\ &= n^2 - nn_1 + n + nn_1 - nn_2 + n + \dots \\ &= n^2 + n_{iter}n \leq n^2 + 2n^2 = O(n^2)\end{aligned}$$

where n_{iter} is the number of iterations, which we already saw was at most $2n$.

Now since we have an example that takes $O(n^2)$, and since we have just shown that GCD is $O(n^2)$ for any input, we know that our bound is tight, and GCD is $\Theta(n^2)$.