ECS 20 — Lecture 14 — Fall 2013 —12 Nov 2013

Phil Rogaway

Today:

o Asymptotic notation
o Solving recurrence relations .

Announcements:

o It's dog day – and there came to class **one** (1) dog.

 (not the actual dog who visited, but a reasonable approximation)

Asymptotic notation and view

Note: below, I am showing O and Θ together. In class, might do one and then the other.

Last time I defined

$O(g) = \{ f\colon \mathbf{N} \to \mathbf{R}\colon \ \exists\, C, N \text{ s.t. } \ f(n) \le C\, g(n) \text{ for all } n \ge N\}$.
$\Theta(g) = \{f\colon \mathbf{N} \to \mathbf{R}\colon \ \exists c, C\, N \text{ s.t. } \ c\, g(n) \le f(n) \le C\, g(n) \text{ for all } n \ge N \}$

Note: some people throw absolute value signs, | | , signs around the f's.
I am omitting them, as, almost always, $f(n)$ is a nonnegative function.
I find it "weird" to consider negative fs in this context.

Here's an almost-equivalent form

$O(g) = \{ f\colon \mathbf{N} \to \mathbf{R}\colon \ \exists\, C, C' \text{ s.t. } \ f(n) \le C\, g(n) + C' \ \text{ for all } n \ge N\}$.
$\Theta(g) = \{f\colon \mathbf{N} \to \mathbf{R}\colon \ \exists c, C\, N \text{ s.t. } \ c\, g(\mathrm{n}) - C \le f(n) \le C\, g(n) \text{ for all } n \ge N \}$

Or, how about

$O(g) = \{ f\colon \mathbf{N} \to \mathbf{R}\colon \ \exists\, C, N \text{ s.t. } \ f(n)\,/g(n) \le C \ \text{ as long as } \ n \ge N \}$.
$\Theta\,(g) = \{ f\colon \mathbf{N} \to \mathbf{R}\colon \ \exists\, C, c, N \text{ s.t. } \ c \le f(n)\,/g(n) \le C \ \text{ as long as } \ n \ge N \}$

<u>People often use "**is**" for "is a member of" or "is an anonymous element of"</u>

They even define things that way, ever regarding O(g) or Θ(g) or as a defined "thing", but only defining what it means to to say that "$f(n)$ is O($g(n)$)" or "$f(n)$ = O($g(n)$)".

"$f(n)$ is O($g(n)$)"

The "qualitative behavior" of practical computation – where, very roughly, things go from "practical" to "impractical" – is often determined more by asymptotic growth rates than constants.

See http://www.csupomona.edu/~ftang/courses/CS240/lectures/analysis.htm for some nice stuff on big-O.

```
n          n lg n     n^2          n^3         2^n
-----------------------------------------------------------
10          30 ns     100 ns        1 us        1 usec
100        700 ns      10 us        1 ms       10^13 yrs
1000        10 us       1 ms        1 sec      10^284 yrs
10000      100 us     0.1 s        17 mins      ---
10^5         2 ms      10 s         1 day       ---
10^6        20 ms      17 mins     32 years     ---
```

Suppose 1 step = 1 nsec (10^-9 sec)

The simplicity afforded by dealing with asymptotics
O(n^2) + O(n^2) = O(n^2)
O(n^2) + O(n^3) = O(n^3)
O($n \log n$) + O(n) = O($n \log n$)
etc.

True/False:

$5n^3 + 100n^2 + 100 = O(n^3)$
If $f \in \Theta(n^2)$ then $f \in O(n^2)$ TRUE
$n! = O(2^n)$ NO
$n! = O(n^n)$ YES

(Truth: $n! = \Theta((n/e)^n$ sqrt$(n))$ --- indeed $n! \sim \left(\frac{n}{e}\right)^n \sqrt{2\pi n}.$ (Stirling's formula)

Claim: $H_n = 1/1 + 1/2 + \ldots + 1/n = O(\lg n)$

Upperbound by 1 + integral_1^n $(1/x)dx = 1 + \ln(n) = O(\lg n)$

Draw picture showing common growth rates

```
Theta(n!)
Theta(2^n)
Theta(n^3)
```

```
            Theta(n^2)
            Theta(n log n log log n)
            Theta(n lg n)
            Theta(n)
                 Theta(sqrt(n)
            Theta(log n)
            Theta(1)

exercise: where is \sqrt(n)
```

The highest degree term in a polynomial is the term that determines the asymptotic growth rate of that polynomial.

General rules: Characterizing Functions in Simplest Terms -- material from URL above

In general we should use the big-Oh notation to characterize a function as closely as possible. For example, while it is true that $f(n) = 4n^3 + 3n^2$ is $O(n^5)$ or even $O(n^4)$, it is more accurate to say that $f(n)$ is $O(n^3)$.

It is also considered a poor taste to include constant factors and lower order terms in the big-Oh notation. For example, it is unfashionable to say that the function $2n^3$ is $O(4n^3 + 8n\log n)$, although it is completely correct. We should strive to describe the function in the big-Oh in **simplest terms**.

Rules of using big-Oh:

- **If $f(n)$ is a polynomial of degree d, then $f(n)$ is $O(n^d)$**. We can drop the lower order terms and constant factors.
- **Use the smallest/closest possible class of functions**, for example, "$2n$ is $O(n)$" instead of "$2n$ is $O(n^2)$"
- **Use the simplest expression of the class, for example**, "$3n + 5$ is $O(n)$" instead of "$3n+5$ is $O(3n)$"

Example usages and recurrence relations

Intertwine examples with the analysis of the resulting recurrence relation

1. How long will the following **fragment of code** take [nested loops, second loop a nontrivial function of the first] -- something $O(n^2)$
2. How long will a computer program take, in the worst case, to run **binary search**, in the worst case? $T(n) = T(n/2) + 1$ -- reminder: have seen recurrence relations before, as with the **Towers of Hanoi** problem. – Then do another recurrence, say $T(n) = 3T(n/2) + 1$. Solution (repeated substitution) $n^{\log\_23} = n^{1.5849...}$ What about $T(n) = 3T(n/2) + n$? Or $T(n) = 3T(n/2) + n^2$? **[recursion tree]**
3. How many gates do you need to **multiply** two n-bit numbers using **grade-school** multiplication?
4. How many comparisons to "**selection sort**" a list of n elements? $T(n) = 1 + T(n-1)$
5. How many comparisons to "**merge sort**" a list of n elements? $T(n) = T(n/2) + n$
6. What's the running time of deciding SAT using the obvious algorithm?

Warning: don't think that asymptotic notation is only for talking about the running time or work of algorithms; it is a convenient way of dealing with functions in lots of domains

```
algorithm BS (X,A, low, high)          // Look for X in A[low..high]. low, high nonneg ints. Return -1 if absent
    if (low>high) return(-1)           // (range of A is empty – element not found)
    m ←⌊(low+high)/2⌋
    if (A[m] = X) return (m)
    if (A[m] < X) return BS (X, A, m + 1, high)      // X not in A[1..m]
    if (A[m] >X)  return BS (X, A, low,  m - 1 )     // X not in A [m..high]
```

From Wikipedia: Karatsuba algorithm (1960/1962) The basic step of **Karatsuba's algorithm** is a formula that allows us to compute the product of two large numbers x and y using three multiplications of smaller numbers, each with about half as many digits as x or y, plus some additions and digit shifts.
Let x and y be represented as n-digit strings in some base B – say B=10. For any positive integer m less than n, one can write the two given numbers as

$x = x_1 10^m + x_0$
$y = y_1 10^m + y_0$,

where x_0 and y_0 are less than 10^m. The product is then
$xy = (x_1 10^m + x_0)(y_1 10^m + y_0)$
$= z_2 10^{2m} + z_1 10^m + z_0$
where
$z_2 = x_1 y_1$
$z_1 = x_1 y_0 + x_0 y_1$
$z_0 = x_0 y_0$.

These formulae require **four multiplications**, and were known to Charles Babbage.[4] Karatsuba observed that xy can be computed in only **three multiplications**, at the cost of a few extra additions. With z_0 and z_2 as before we can calculate
$z_1 = (x_1 + x_0)(y_1 + y_0) - z_2 - z_0$
which holds since
$z_1 = x_1 y_0 + x_0 y_1$
$z_1 = (x_1 + x_0)(y_1 + y_0) - x_1 y_1 - x_0 y_0$.

Example To compute the product of 12345 and 6789, choose B = 10 and m = 3. Then we decompose the input operands using the resulting base ($B^m = 1000$), as:
$12345 = 12 \cdot 1000 + 345$
$6789 = 6 \cdot 1000 + 789$
Only three multiplications are required, and they are operating on smaller integers are used to compute three partial results:
$z_2 = 12 \times 6 = 72$
$z_0 = 345 \times 789 = 272205$
$z_1 = (12 + 345) \times (6 + 789) - z_2 - z_0 = 357 \times 795 - 72 - 272205 = 283815 - 72 - 272205 = 11538$
We get the result by just adding these three partial results, shifted accordingly (and then taking carries into account by decomposing these three inputs in base 1000 like for the input operands):
result = $z_2 \cdot B^{2m} + z_1 \cdot 10^m + z_0$, i.e.
result = $72 \cdot 1000^2 + 11538 \cdot 1000 + 272205 = 83810205$.

Then: solve the recurrence
T(n) = 1 if n=1,
T(n) = 3T(n/2) + n if n>1
(will do afresh next class)

There is more than O and Θ. (Table modified from Wikipedia)

Notation	Intuition	Informal definition: for sufficiently large n...	Formal Definition
$f(n) \in O(g(n))$	f is bounded above by g (up to constant factor)	$\|f(n)\| \leq g(n) \cdot k$ for some positive *k*	$\exists k > 0\ \exists n_0\ \forall n > n_0\ f(n) \leq g(n) \cdot k$
$f(n) \in \Omega(g(n))$	f is bounded below by g	$f(n) \geq g(n) \cdot k$ for some positive *k*	$\exists k > 0\ \exists n_0\ \forall n > n_0\ g(n) \cdot k \leq f(n)$
$f(n) \in \Theta(g(n))$	f is bounded above and below by g	$g(n) \cdot k_1 \leq f(n) \leq$ for some positive k_1, k_2	$\exists k_1 > 0\ \exists k_2 > 0\ \exists n_0\ \forall n > n_0$ $g(n) \cdot k_1 \leq f(n) \leq g(n) \cdot k_2$
$f(n) \in o(g(n))$	f is dominated by g	$\|f(n)\| \leq k \cdot \|g(n)\|$ for every fixed positive number k	$\forall k > 0\ \exists n_0\ \forall n > n_0\ \|f(n)\| \leq k \cdot \|g(n)\|$
$f(n) \in \omega(g(n))$	f dominates g	$\|f(n)\| \geq k \cdot \|g(n)\|$ for every fixed positive number k	$\forall k > 0\ \exists n_0\ \forall n > n_0\ \|f(n)\| \geq k \cdot \|g(n)\|$
$f(n) \sim g(n)$	f is equal to g asymptotically	$f(n)/g(n) \to 1$	$\forall \varepsilon > 0\ \exists n_0\ \forall n > n_0\ \left\| \frac{f(n)}{g(n)} - 1 \right\| < \varepsilon$