Asymptotic Growth of Functions: Big-Oh Notation

Def^n Let $f: [0,\infty) \to \mathbb{R}$
$g: [0,\infty) \to \mathbb{R}$ eventually positive (i.e. positive for $x >$ some bound)

We say $f(x) = O(g(x))$ if there are constants $C > 0$ and $M > 0$ such that

$$|f(x)| \leq C\, g(x) \quad \text{for } x > M.$$

Intuitively: $f = O(g)$ if g dominates f if allowed ① a constant scaling (C)
② we look at the tail/far enough out (M)

Write $f = O(g)$
a kind of "upper bd"

Examples. $7x^2 + 3 = O(x^2)$
$300x^2 = O(x^2)$
$x^2 = O(x^3)$ silly

Examples.

$f = 500 \log x$

$g = x$

$f = O(g)$

$g \neq O(f)$

$f = |\sin x|$

$g = |\cos x|$

$f \neq O(g)$

$g \neq O(f)$

$f = |\sin x|$

$g = \frac{1}{2}$

Runtimes

time complexity = runtime is the # of operations performed in an algorithm as a function of input size

space complexity is # of bits of memory needed to run an algorithm as a function of input size.

Ex. Counting. Input: an integer n in decimal notation

> input size = # digits
> $M := \lceil \log_{10}(n) \rceil$

Algorithm: print "Hello!" n times.

n steps, each consist of a constant (near-) # of operations

runtime $= \Theta(n) = O(10^M)$

exponential runtime

First Assessment, Friday, September 16th in class. Covers everything so far (details on website).

Trial Division for factoring.

Input = n, (integer to factor) input size = $M = \log_{10}(n)$

Algorithm: trial divide n by m for all $2 \leq m \leq \sqrt{n}$

Goal: find a factor.

Runtime = $O(\sqrt{n})$ trial divisions

Runtime of one trial division is polynomial in M

Runtime = $O((\sqrt{n}) \cdot \text{poly}(M)) = O(10^{M/2}\,\text{poly}(M))$

exponential! (bad)

Runtimes. **polynomial**: $O(p(M))$ for some polynomial p

(M = input size)

exponential: $O(e^{p(M)})$ for some polynomial p

Subexponential: runtime $< e^{p(M)}$ for **all** poly p

"in between"

Examples:

$e^{c\sqrt{M \log M}}$ comes up a lot

$M^{\log M} = e^{(\log M)^2}$

both are **not** polynomial, but are faster than exponential

Modular Exponentiation $g^e \pmod n$

Input: base g, exponent e, modulus n.

Size: $O(\log(n))$ bits

Steps: $\log(n)$ squarings $g, g^2, g^4, g^8, \ldots$

+ $\log(n)$ multiplications to build $g^e = g^1 g^4 \cdot g^{32} \ldots$

$O(\log n)$ multiplications

↳ are each polynomial time ie. $\text{poly}(\log(n))$ ← school algorithm

So overall: $O(\log n) \cdot \text{poly}(\log(n)) = \text{poly}(\log(n))$

polynomial time.

Discrete Logarithm Problem

Notation: $h = g^a \pmod p$ assume g is invertible.

then $a = L_g(h) \pmod{p-1}$ "discrete log base g of h mod p"

often we take $0 \le a < \text{ord}(g) \le p-1$.

Example. mod 5.

$2^0 = 1,\ 2^1 = 2,\ 2^2 = 4,\ 2^3 = 3,\ 2^4 = 1$

primitive root

$4^0 = 1,\ 4^1 = 4,\ 4^2 = 1$

order 2

So $L_2(3) = 3$

$L_2(4) = 2$

$L_2(1) = 0$

$L_4(3) =$ not defined.

Logs are exponents so:

$$L_g(h_1 h_2) = L_g(h_1) + L_g(h_2)$$