Exponential Functions
Logarithmic Functions

Fix $a > 0$, $a \neq 1$.

The exponential function with base a is

$f(x) = a^x \quad (\forall x \in \mathbb{R})$

($\forall$ = for all
$\exists$ = there exists ... such that)

$a^{x+y} = a^x a^y$

$(ab)^x = a^x b^x$

$a^0 = 1$

$a^{-x} = \frac{1}{a^x}$

$(a^x)^y = a^{xy} = (a^y)^x$

For $a > 1$,

$a^x = \Omega(x^n)$ for any constant n

$a = 2$, usually

o, ω notation

Given asymptotically positive f, g,

$f(n) = o(g(n)) \quad [f \in o(g)]$

means

$\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$

$\forall c > 0, \exists n_0, \forall n \geq n_0,$

$\frac{f(n)}{g(n)} \leq c \quad [f(n) \leq c\, g(n)]$

$f(n) = o(g(n)) \Rightarrow f(n) = O(g(n))$

o means "$<$" asymptotically

$f(n) = \omega(g(n)) \quad [f \in \omega(g)]$

if $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$

equivalently, $\lim_{n\to\infty} \frac{g(n)}{f(n)} = 0$,

equivalently $g(n) = o(f(n))$

ω means "$>$"

$f(n) = \omega(g(n)) \Rightarrow f(n) = \Omega(g(n))$

Summary: O, Ω, Θ, o, ω

$f(n) = \text{Poly}(g(n))$ means

$\exists k$ constant such that

$f(n) = O(g(n)^k)$

$n^6 = \text{Poly}(n)$ (example)

$2^n \neq \text{Poly}(n)$

$a^n = \omega(n^k)$ for any k, and any $a > 1$.

Logs. $a > 0$, $a \neq 1$.

$\log_a$ is the logarithmic function with base a.

$\log_a a^x = x \quad (\forall x \in \mathbb{R})$

$a^{\log_a x} = x \quad (\forall x > 0)$

$\log_a x$ is defined for all $x > 0$.

Use $\log_2$ a lot.

So $\lg = \log_2$

e^x is natural exponential

($e \doteq 2.71828\ldots$)

$\log_e x$ is natural log

$\ln x$

Properties: $\forall x,y > 0$, $\forall r$

$\log_a(xy) = \log_a x + \log_a y$

(product rule)

$\log_a 1 = 0$

$\log_a \frac{x}{y} = \log_a x - \log_a y$

(quotient rule)

$\log_a(x^r) = r \log_a x$

(power rule)

$b^{\log_a c} = c^{\log_a b}$

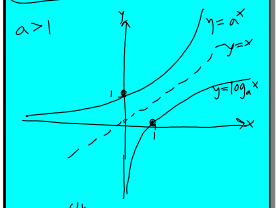
Why?

$(\log_a b)(\log_a c) = (\log_a c)(\log_a b)$

$\log_a\left(c^{\log_a b}\right) = \log_a\left(b^{\log_a c}\right)$

take a to the power of both sides:

$c^{\log_a b} = b^{\log_a c}$ //



$\log_2 2^{64} = 64$

$\log_a n = o(n^\varepsilon)$

for any real $\varepsilon > 0$.

So algo that runs in time $\Theta(n \lg n)$ is better than an algo for the same problem that runs in time $\Theta(n^{1.00001})$

Dealt with arithmetic sums.

Geometric sums:

Fix r such that $r \neq 1$.

The geometric sum with ratio r is either

$\sum_{k=0}^{n} r^k$ or $\sum_{k=0}^{\infty} r^k$

finite, depends on n — infinite, defined when $|r| < 1$.

Closed forms:

$$\sum_{k=0}^{n} r^k = \frac{r^{n+1} - 1}{r - 1}$$

$$\sum_{k=0}^{\infty} r^k = \lim_{n\to\infty} \sum_{k=0}^{n} r^k$$

$$= \lim_{n\to\infty} \left(\frac{r^{n+1} - 1}{r - 1}\right) = \frac{1}{1 - r}$$

Deriving the closed form

Set $S := \sum_{k=0}^{n} r^k$

$rS = \sum_{k=0}^{n} r r^k = \sum_{k=0}^{n} r^{k+1}$

$[k' := k+1] \quad = \sum_{k'=1}^{n+1} r^{k'} = \sum_{k=1}^{n+1} r^k$

subtract 2nd eqn from 1st:

$S - rS = \sum_{k=0}^{n} r^k - \sum_{k=1}^{n+1} r^k$

$S(1-r) = r^0 - r^{n+1} = 1 - r^{n+1}$

$\therefore \quad S = \frac{1-r^{n+1}}{1-r} = \frac{r^{n+1}-1}{r-1}$

Change of base rule for logs

$\log_a b \log_b c = \log_a c$

$\therefore \boxed{\log_b c = \frac{\log_a c}{\log_a b}}$

Ex: $\lg n = \frac{\log_a n}{\log_a 2}$

$\lg n = \Theta(\log_a n)$ any constant $a > 1$.

Integer multiplication algorithms

multiplying two n-bit numbers:

$\Theta(n^2)$ — naive grade school algo

Divide & conquer algo

Assume can recursively multiply 2 n-bit numbers, want to multiply 2 2n-bit numbers

$a = a_h 2^n + a_l$

$b = b_h 2^n + b_l$

a_h, b_h are n high-order bits of a, b, respectively

a_l, b_l are n low-order bits of a, b.

Then

$ab = (a_h 2^n + a_l)(b_h 2^n + b_l)$

$= a_h b_h 2^{2n} + (a_h b_l + a_l b_h) 2^n + a_l b_l$

Let $T(n)$ be the max time for two n-bit numbers.

$T(2n) = 4T(n) + \Theta(n)$

Solving this recurrence gives

$T(n) = \Theta(n^2)$

(same as the grade school method)

Another method:

$a = a_h 2^n + a_l$

$b = b_h 2^n + b_l$

$ab = a_h b_h 2^{2n} + (\overbrace{a_h b_l + a_l b_h}^{P - a_l b_l - a_h b_h}) 2^n + a_l b_l$

Trick:

compute $(a_h + a_l)(b_h + b_l) = P$ recursively

also compute $a_h b_h$, $a_l b_l$ recursively

$P = a_h b_h + a_l b_l + (\overbrace{a_h b_l + a_l b_h}^{P - a_h b_h - a_l b_l})$

$T(n) = 3T(n) + \Theta(n)$ [$T(n) = n^{\log_2 3}$]