Math Review

CSE 373
Data Structures & Algorithms
Ruth Anderson
Spring 2010

Today's Outline

- **Announcements**
  – Assignment #1 due Thurs, April 8 at 11:45pm
  – Email sent to cse373 mailing list – did you get it?
  – Have you installed Eclipse and Java yet?
- **Queues and Stacks**
- **Math Review**
  – **Proof by Induction**
  – **Powers of 2**
  – **Binary numbers**
  – **Exponents and Logs**

Mathematical Induction

Suppose we wish to prove that:

For all $n \geq n_0$, some predicate $P(n)$ is true.

We can do this by proving two things:

1. $P(n_0)$ --- this is called the "basis."
2. If $P(k)$ then $P(k+1)$ -- this is called the "induction step."

Example: Basis Step

Prove for all $n \geq 1$, sum of first n powers of 2 = $2^n - 1$

$2^0 + 2^1 + 2^2 + \ldots + 2^{n-1} = 2^n - 1.$

in other words: $1 + 2 + 4 + \ldots + 2^{n-1} = 2^n - 1.$

Proof by induction:
Basis with $n_0 = 1$:

(left hand side) $2^{1-1} = 2^0 = 1$
(right hand side) $2^1 - 1 = 2 - 1 = 1$

So true for $n_0 = 1$

Example: Inductive Step

- *Induction hypothesis*: (Assume this is true)
  $1 + 2 + 4 + \ldots + 2^{k-1} = 2^k - 1$
- *Induction step*: Now add 2^k to both sides:
  $1 + 2 + 4 + \ldots\ 2^{k-1} + \mathbf{2^k} = 2^k - 1 + \mathbf{2^k}$
  $= 2(2^k) - 1$
  $= 2^{k+1} - 1$

  Therefore if the equation is valid for n = k, it must also be valid for n = k+1.
- *Summary*: It is valid for n=1 (basis) and by the induction step it is therefore valid for n=2, n=3, …
  It is valid for all integers greater than 0.

Powers of 2

- Many of the numbers we use in Computer Science are powers of 2
- Binary numbers (base 2) are easily represented in digital computers
  – each "bit" is a 0 or a 1
  – an n-bit wide field can represent how many different things?

0000000000101011

N bits can represent how many things?

# Bits	Patterns	# of patterns
1		
2		

Unsigned binary numbers

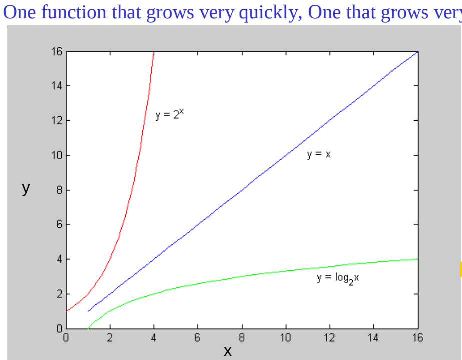
- For **unsigned** numbers in a fixed width field
  - the minimum value is 0
  - the maximum value is 2^n-1, where n is the number of bits in the field
  - The value is $\sum_{i=0}^{i=n-1} a_i 2^i$
- Each bit position represents a power of 2 with $a_i = 0$ or $a_i = 1$

Signed Numbers?

Logarithms and Exponents

- Definition: $\log_2 x = y$ if and only if $x = 2^y$
  - $8 = 2^3$, so $\log_2 8 = 3$
  - $65536 = 2^{16}$, so $\log_2 65536 = 16$
- Notice that $\log_2 n$ tells you how many bits are needed to distinguish among n different values.
  - 8 bits can hold any of 256 numbers, for example: 0 to 2^8-1, which is 0 to 255
  - $\log_2 256 = 8$

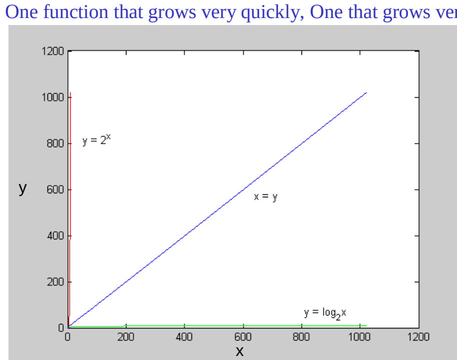
One function that grows very quickly, One that grows very slowly




```
x = 0:.1:4
y = 2.^x
plot(x,y,'r')
hold on
plot(y,x,'g')
plot(y,y,'b')
```

x, 2^x and $\log_2 x$

One function that grows very quickly, One that grows very slowly




```
x = 0:10
y = 2.^x
plot(x,y,'r')
hold on
plot(y,x,'g')
plot(y,y,'b')
```

2^x and $\log_2 x$

Floor and Ceiling

$\lfloor X \rfloor$ Floor function: the largest integer $\leq$ X

$\lfloor 2.7 \rfloor = 2 \qquad \lfloor -2.7 \rfloor = -3 \qquad \lfloor 2 \rfloor = 2$

$\lceil X \rceil$ Ceiling function: the smallest integer $\geq$ X

$\lceil 2.3 \rceil = 3 \qquad \lceil -2.3 \rceil = -2 \qquad \lceil 2 \rceil = 2$

Facts about Floor and Ceiling

1. $X - 1 < \lfloor X \rfloor \leq X$
2. $X \leq \lceil X \rceil < X + 1$
3. $\lfloor n/2 \rfloor + \lceil n/2 \rceil = n$ if n is an integer

Properties of logs

- We will assume logs to base 2 unless specified otherwise.
- $8 = 2^3$, so $\log_2 8 = 3$, so $2^{(\log_2 8)} =$ ________

Show:

log (A • B) = log A + log B

$A = 2^{\log_2 A}$ and $B = 2^{\log_2 B}$

$A \bullet B = 2^{\log_2 A} \bullet 2^{\log_2 B} = 2^{\log_2 A + \log_2 B}$

So: $\log_2 AB = \log_2 A + \log_2 B$

- Note: $\log AB \neq \log A \bullet \log B$!!

Other log properties

- log A/B = log A – log B
- $\log (A^B) = B \log A$
- log log X < log X < X for all X > 0
  - log log X = Y means: $2^{2^Y} = X$
  - log X grows more slowly than X
    - called a "sub-linear" function

Note: $\log \log X \neq \log^2 X$

$\log^2 X = (\log X)(\log X)$ aka "log-squared"

A log is a log is a log

- "Any base B log is equivalent to base 2 log within a constant factor."

$B = 2^{\log_2 B}$

$x = 2^{\log_2 x}$

$$\log_B X = \log_B X$$

substitution $B^{\log_B X} = X$ $B^{\log_B X} = X$ by def. of logs

$$(2^{\log_2 B})^{\log_B X} = 2^{\log_2 X}$$

$$2^{\log_2 B \log_B X} = 2^{\log_2 X}$$

$$\log_2 B \log_B X = \log_2 X$$

$$\boxed{\log_B X = \frac{\log_2 X}{\log_2 B}}$$

Arithmetic Sequences

N = {0, 1, 2, ... } = natural numbers

[0, 1, 2, ...] is an infinite arithmetic sequence

[a, a+d, a+2d, a+3d, ...] is a general infinite arith. sequence.

There is a *constant difference* between terms.

$$1 + 2 + 3 + \ldots + N = \sum_{i=1}^{N} i = \frac{N(N+1)}{2}$$

Algorithm Analysis Examples

- Consider the following program segment:

```
x:= 0;
for i = 1 to N do
  for j = 1 to i do
    x := x + 1;
```

- What is the value of x at the end?

03/31/10 21

Analyzing the Loop

- Total number of times x is incremented is executed =

$$1+2+3+\ldots+N=\sum_{i=1}^{N} i=\frac{N(N+1)}{2}$$

- Congratulations - You've just analyzed your first program!
  - Running time of the program is proportional to N(N+1)/2 for all N
  - Big-O ??

03/31/10 22

Asymptotic Analysis

What we want

- Rough Estimate
- Ignores Details

03/31/10 25

Big-O Analysis

- Ignores "details"

03/31/10 26

Analysis of Algorithms

- Efficiency measure
  - how long the program runs time complexity
  - how much memory it uses space complexity
    - For today, we'll focus on time complexity only
- *Why analyze at all?*

03/31/10 27

Asymptotic Analysis

- Complexity as a function of input size *n*

  T(*n*) = 4*n* + 5

  T(*n*) = 0.5 *n* log *n* - 2*n* + 7

  T(*n*) = $2^n + n^3 + 3n$

- *What happens as n grows?*

03/31/10 28

Why Asymptotic Analysis?

- Most algorithms are fast for small *n*
  - Time difference too small to be noticeable
  - External things dominate (OS, disk I/O, …)
- BUT *n* is often large in practice
  - Databases, internet, graphics, …
- Time difference really shows up as *n* grows!

03/31/10 29

Big-O: Common Names

– constant:	O(1)	
– logarithmic:	O(log n)	
– linear:	O(n)	
– quadratic:	$O(n^2)$	
– cubic:	$O(n^3)$	
– polynomial:	$O(n^k)$	(k is a constant)
– exponential:	$O(c^n)$	(c is a constant > 1)

03/31/10 30