Math Review

CSE 373
Data Structures & Algorithms
Ruth Anderson
Autumn 2011

Today's Outline

- **Announcements**
    - Assignment #1 due Thurs, Oct 6 at 11pm
    - Midterm date? Please fill out poll by end of day today.

- **Math Review**
    - **Proof by Induction**
    - **Powers of 2**
    - **Binary numbers**
    - **Exponents and Logs**
- **Algorithm Analysis**

When did you take cse 143?

Total responses (N): 92 Did not respond: 0

Numeric value	Answer	Frequency	Percentage
1	0 - summer 11	6	6.52%
2	1 - spring 11	8	8.70%
3	2 - winter 11	21	22.83%
4	3 - autumn 10	13	14.13%
5	4 - summer 10	2	2.17%
6	5 - spring 10	16	17.39%
7	6 - Before spring 10	19	20.65%
8	7 - I did not take cse 143 at UW (AP or transfer credit)	5	5.43%
9	Other:	2	2.17%

Homework 1 – Sound Blaster!

Play your favorite song in reverse!

<u>Aim</u>:
1. Implement stack ADT two different ways
2. Use to reverse a sound file

<u>Due</u>: Thurs, Oct 6, 2011
   Submit via catalyst drop box before: 11pm

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 "base case" or "basis."
2. If P(k), then P(k+1) - this is called the "induction step" or "inductive case"

Note: We prove 2. by assuming P(k) is true.

Putting these together, we show that P(n) is true.

Example

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

1

Example Proof by Induction

Theorem: $P(n)$ holds for all $n \geq 1$

Proof: By induction on n

- Base case, $n=1$: $2^0 = 1 = 2^1 - 1$
- Induction step:
  - Inductive hypothesis: Assume the sum of the first k powers of 2 is 2^k-1
  - Given the hypothesis, show that:
    the sum of the first ($k+1$) powers of 2 is 2^{k+1}-1

From our inductive hypothesis we know:
$$1 + 2 + 4 + ... + 2^{k-1} = 2^k - 1$$

Add the next power of 2 to both sides…
$$1 + 2 + 4 + ... + 2^{k-1} + 2^k = 2^k - 1 + 2^k$$

We have what we want on the left; massage the right a bit:
$$1 + 2 + 4 + ... + 2^{k-1} + 2^k = 2(2^k) - 1 = 2^{k+1} - 1$$

Example: Putting it all together

- *Inductive hypothesis*: (We assumed this was true)
  $1 + 2 + 4 + … + 2^{k-1} = 2^k - 1$
- *Induction step*: (Adding 2^k to both sides)
  $1 + 2 + 4 + … \; 2^{k-1} + 2^k = 2^k - 1 + 2^k = 2(2^k) - 1 = 2^{k+1} - 1$
  Therefore if the equation is valid for n = k, it must also be valid for n = k+1.

Summary: Our theorem is valid for n=1 (base case) and by the induction step it is therefore valid for n=2, n=3, …

Thus, it is valid for all integers greater than or equal to 1.

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 $a_i = 1$

Powers of 2

- A bit is 0 or 1
- A sequence of n bits can represent 2^n distinct things
  - For example, the numbers 0 through 2^n-1
- 2^{10} is 1024 ("about a thousand", kilo in CSE speak)
- 2^{20} is "about a million", mega in CSE speak
- 2^{30} is "about a billion", giga in CSE speak

Java:
- an **int** is 32 bits and signed, so "max int" is "about 2 billion"
- a **long** is 64 bits and signed, so "max long" is 2^{63}-1

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
  $\log_2 256 = 8$

Therefore…

Could give a unique id to…
- Every person in the U.S. with 29 bits
- Every person in the world with 33 bits
- Every person to have ever lived with 38 bits (estimate)
- Every atom in the universe with 250-300 bits

So if a password is 128 bits long and randomly generated, do you think you could guess it?

Signed Numbers?

Logarithms and Exponents

- Since so much is binary in CS, `log` almost always means `log₂`
- Definition: $\log_2 x = y$ if $x = 2^y$
- So, $\log_2 1{,}000{,}000 = $ "a little under 20"
- Just as exponents grow *very* quickly, logarithms grow *very* slowly
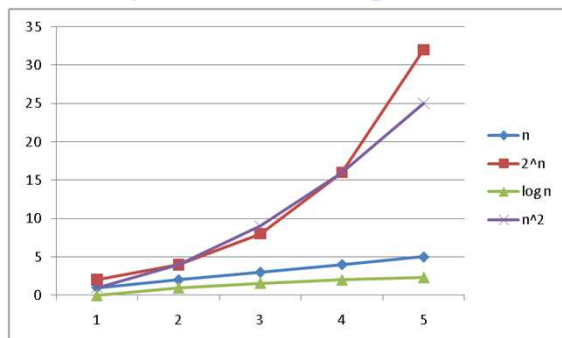
See Excel file
for plot data –
play with it!
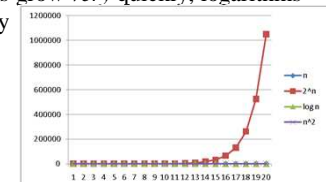
Logarithms and Exponents

Logarithms and Exponents

Logarithms and Exponents

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')
```
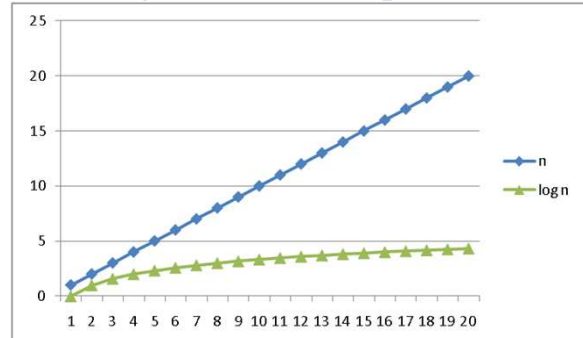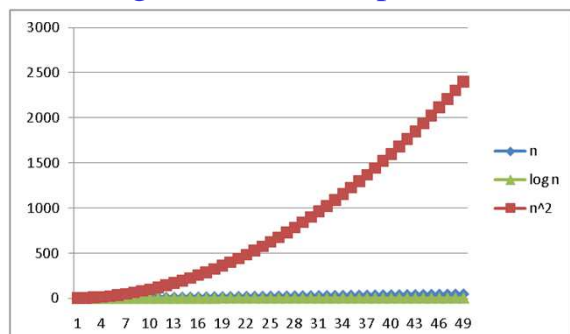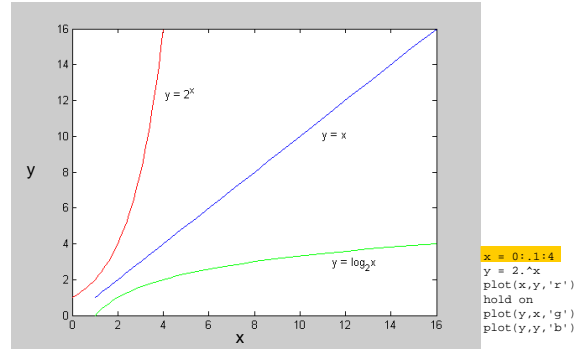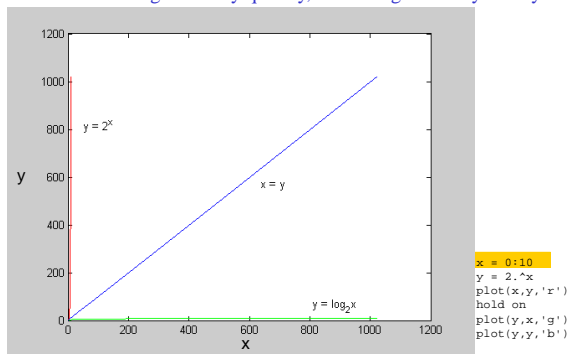
2^x and $\log_2 x$

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.
- $x = \log_2 2^x$
- $8 = 2^3$, so $\log_2 8 = 3$, so $2^{(\log_2 8)} = $ _____

Show:

$\log (A \cdot B) = \log A + \log B$

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

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

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

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

Also, it follows that `log(N^k)= k log N`

4

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: $2^{2^Y} = X$
  - Ex. $\log_2 \log_2 4\,billion \sim \log_2 \log_2 2^{32} = \log_2 32 = 5$
- $\log X$ grows more slowly than X
  - called a "sub-linear" function
- $(\log x)(\log x)$ is written $\log^2 x$ (aka "log-squared")
  - It is greater than $\log x$ for all $x > 2$
- Note: $\log \log X \neq \log^2 X$

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

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

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

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

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

Log base doesn't matter (much)

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

- And we are about to stop worrying about constant factors!
- In particular, $\texttt{log}_2 \texttt{ x = 3.22 log}_{10} \texttt{ x}$
- In general, we can convert log bases via a constant multiplier
- To convert from base B to base A:

  $$\texttt{log}_B \texttt{ x = (log}_A \texttt{ x) / (log}_A \texttt{ B)}$$

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.

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

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?

Analyzing the Loop

- Total number of times x is incremented is executed =

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

- 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 ??