CSE373: Data Structures and Algorithms

Lecture 2: Math Review; Algorithm Analysis

Hunter Zahn

Summer 2016

Today

- Finish discussing stacks and queues

- Review math essential to algorithm analysis
  - Proof by induction
  - Powers of 2
  - Binary numbers
  - Exponents and logarithms

- Begin analyzing algorithms
  - Using asymptotic analysis (continue next time)

Prove that $1 + 2 + 4 + 8 + \ldots + 2^n = 2^{n+1}-1$

Background on Induction

- Type of mathematical proof
- Typically used to establish a given statement for all natural numbers (integers > 0)
- Proof is a sequence of deductive steps
  1. Show the statement is true for the first number.
  2. Show that if the statement is true for any one number, this implies the statement is true for the next number.
  3. If so, we can infer that the statement is true for all numbers.

Think about climbing a ladder

1. Show you can get to the first rung (base case)

2. Show you can get between rungs (inductive step)

3. Now you can climb forever.

5 steps to inductive proofs

1. State what you're trying to prove.
   – Suppose that P(n) is some predicate (mention n)
   – Ex:

     "Let P(n) be …

     Will prove that P(n) is true for every n >= x"
2. Prove the "base case"
   – Show that P(x) is true
3. Inductive Hypothesis (IH)
   – Assume that P(k) is true for some arbitrary integer k in the set of integers you're looking at

4. Inductive Step
   – Show that P(k + 1) is true.
   – Be sure to use the Inductive Hypothesis, and point out where you use it!
5. Conclusion

Why you should care

- Induction turns out to be a useful technique
  - AVL trees
  - Heaps
  - Graph algorithms
  - Can also prove things like $3^n > n^3$ *for* $n \geq 4$
- Exposure to rigorous thinking

Prove that $1 + 2 + 4 + 8 + \ldots + 2^n = 2^{n+1}-1$

Example

P(n) = "the sum of the first *n* powers of 2 (starting at 0) is 2^n-1"

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

Proof: By induction on *n*

- Base case: *n*=1. Sum of first 1 power of 2 is 2^0 , which equals 1.
  And for *n*=1, 2^n-1 equals 1.
- Inductive case:
  - Assume the sum of the first *k* powers of 2 is 2^k-1
  - Show the sum of the first (*k*+1) powers of 2 is 2^{k+1}-1

  Using assumption, sum of the first (*k*+1) powers of 2 is

  $(2^k-1) + 2^{(k+1)-1} = (2^k-1) + 2^k = 2^{k+1}-1$

Powers of 2

- A bit is 0 or 1 (just two different "letters" or "symbols")
- 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

Therefore…

Could give a unique id to…

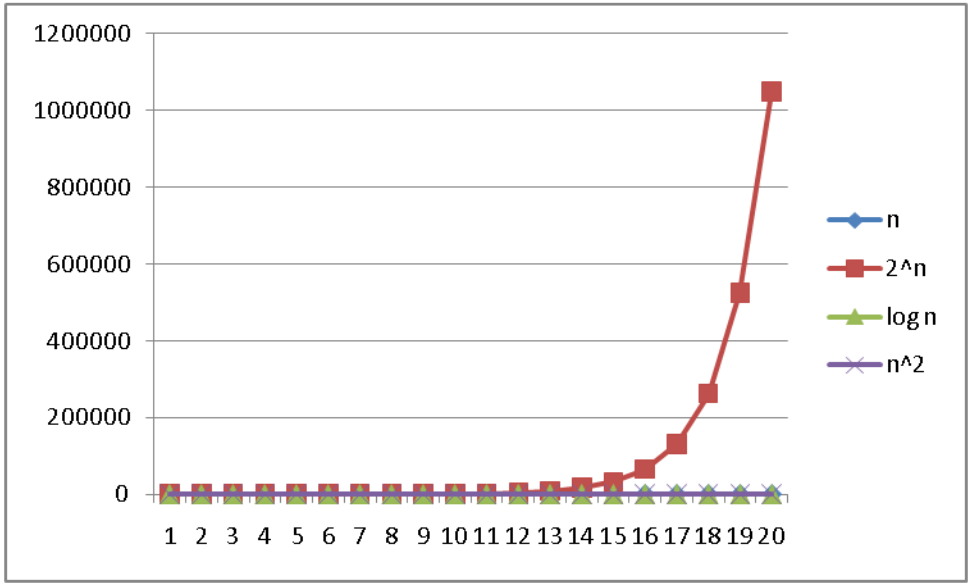
- Every person in the U.S. with 29 bits
- Every person in the world with 33 bits
- Every person to have ever lived with 38 bits (estimate)
- Every atom in the universe with 250-300 bits

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

Logarithms and Exponents

- Since so much is binary **log** in CS almost always means $\log_2$
- 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

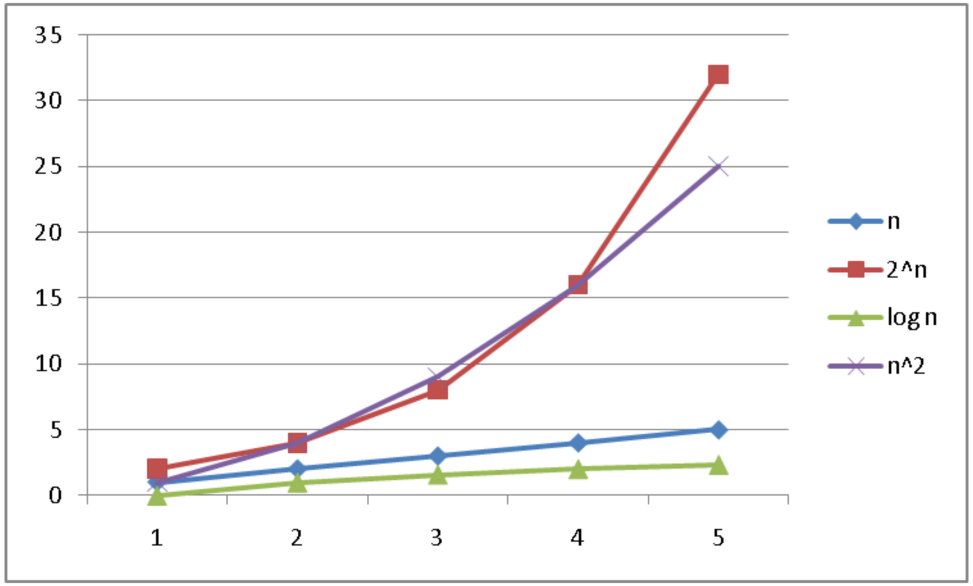
See Excel file for plot data – play with it!



Logarithms and Exponents

- Since so much is binary **log** in CS almost always means $\log_2$
- 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

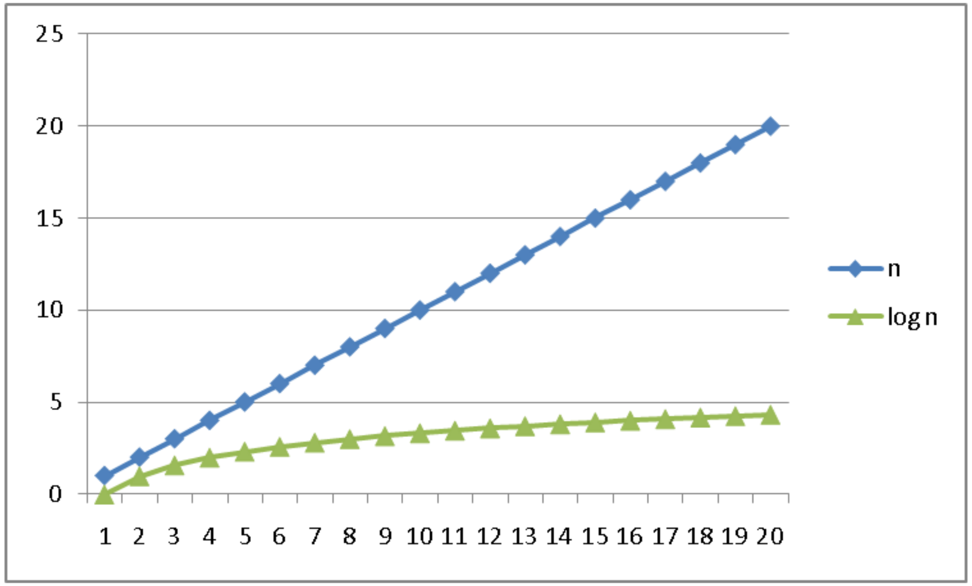
See Excel file for plot data – play with it!



Logarithms and Exponents

- Since so much is binary **`log`** in CS almost always means $\log_2$
- 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

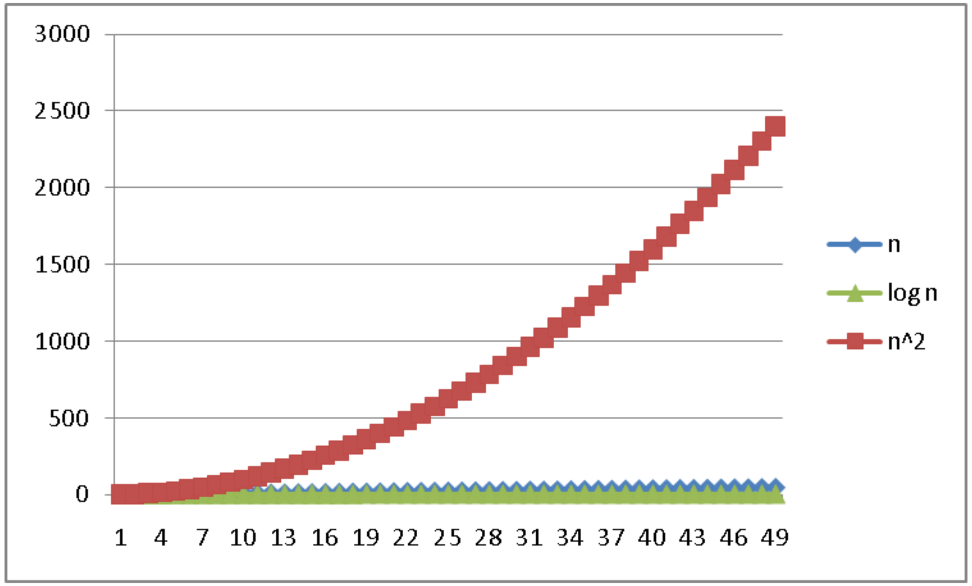
See Excel file for plot data – play with it!



Logarithms and Exponents

- Since so much is binary **`log`** in CS almost always means $\log_2$
- 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!



Properties of logarithms

- $\log(A*B) = \log A + \log B$
  – So $\log(N^k) = k \log N$

- $\log(A/B) = \log A - \log B$

- $\log(\log x)$ is written $\log \log x$
  – Grows as slowly as 2^{2^y} grows quickly

- $(\log x)(\log x)$ is written $\log^2 x$
  – It is greater than $\log x$ for all $x > 2$
  – It is not the same as $\log \log x$

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, $\log_2 x = 3.22 \log_{10} x$
- In general,

$$\log_B x = (\log_A x) / (\log_A B)$$

Floor and ceiling

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

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

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

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

Floor and ceiling properties

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

Algorithm Analysis

As the "size" of an algorithm's input grows
(integer, length of array, size of queue, etc.):

- How much longer does the algorithm take (time)
- How much more memory does the algorithm need (space)

Because the curves we saw are so different, often care about only "which curve we are like"

Separate issue: Algorithm *correctness* – does it produce the right answer for all inputs

- Usually more important, naturally

Example

- What does this pseudocode return?

```
x := 0;
for i=1 to N do
  for j=1 to i do
     x := x + 3;
return x;
```

- Correctness: For any N ≥ 0, it returns…

Example

- What does this pseudocode return?

```
x := 0;
for i=1 to N do
  for j=1 to i do
     x := x + 3;
return x;
```

- Correctness: For any N ≥ 0, it returns 3N(N+1)/2
- Proof: By induction on *n*
  - *P(n)* = after outer for-loop executes *n* times, **`x`** holds 3n(n+1)/2
  - Base: n=0, returns 0
  - Inductive: From *P(k)*, **`x`** holds 3k(k+1)/2 after *k* iterations. Next iteration adds 3(k+1), for total of 3k(k+1)/2 + 3(k+1) = (3k(k+1) + 6(k+1))/2 = (k+1)(3k+6)/2 = 3(k+1)(k+2)/2

Example

- How long does this pseudocode run?

```
x := 0;
for i=1 to N do
  for j=1 to i do
     x := x + 3;
return x;
```

- Running time: For any N ≥ 0,
  - Assignments, additions, returns take “1 unit time”
  - Loops take the sum of the time for their iterations

- So: 2 + 2*(number of times inner loop runs)
  - And how many times is that…

Example

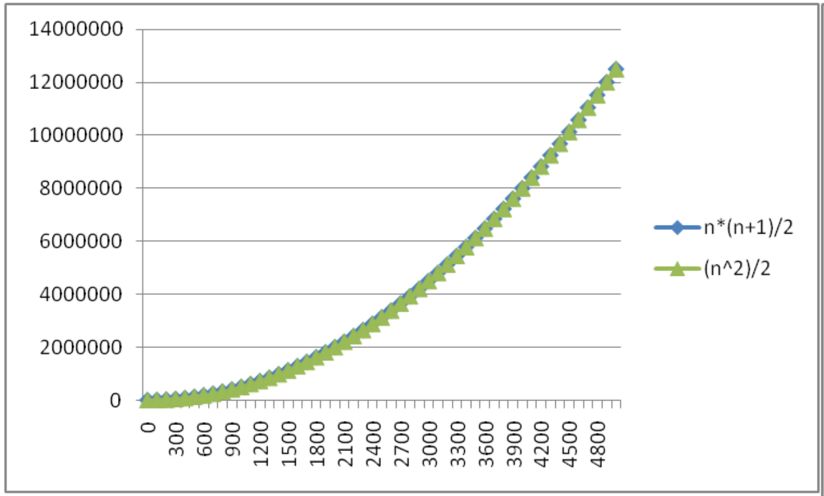
- How long does this pseudocode run?

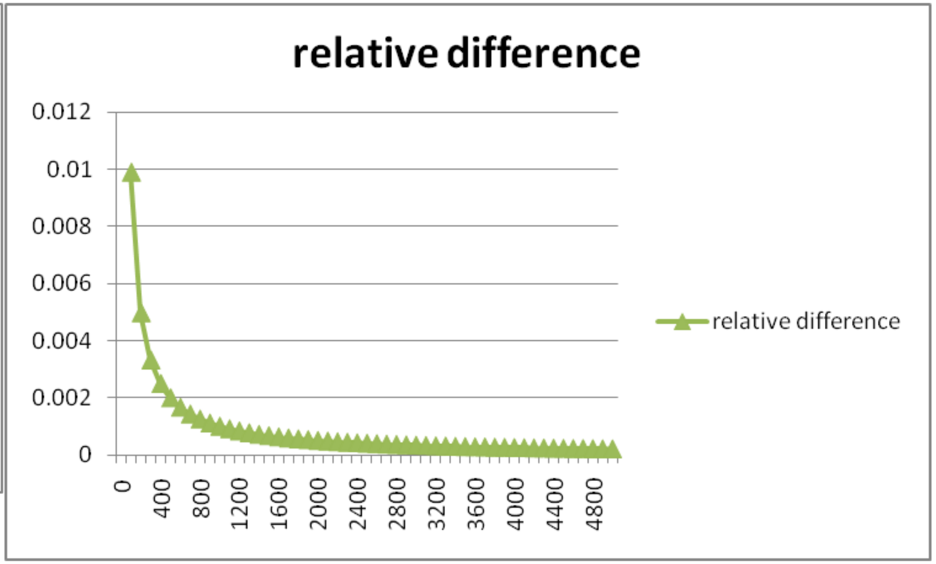
```
x := 0;
for i=1 to N do
  for j=1 to i do
     x := x + 3;
return x;
```

- The total number of loop iterations is N*(N+1)/2
  - This is a very common loop structure, worth memorizing
  - Proof is by induction on N, known for centuries
  - This is *proportional to* N^2 , and we say $O(N^2)$, "big-Oh of"
    - For large enough N, the N and constant terms are irrelevant, as are the first assignment and return
    - See plot… N*(N+1)/2 vs. just $N^2/2$

Lower-order terms don't matter

N*(N+1)/2 vs. just $N^2/2$

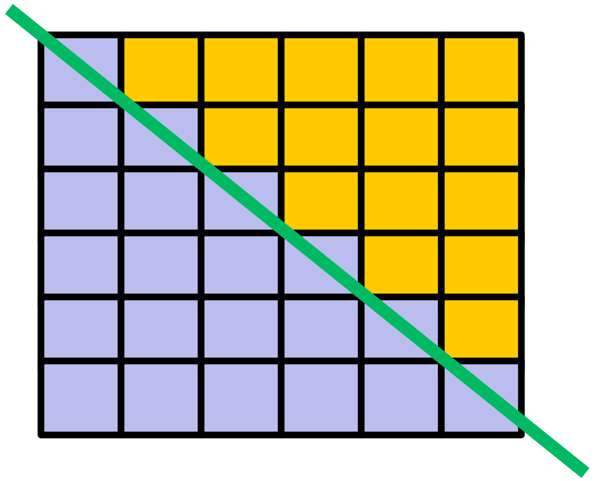




Geometric interpretation

$$\sum_{i=1}^{N} i = N*N/2+N/2$$

```
for i=1 to N do
  for j=1 to i do
     // small work
```



- Area of square: N*N
- Area of lower triangle of square: N*N/2
- Extra area from squares crossing the diagonal: N*1/2
- As N grows, fraction of “extra area” compared to lower triangle goes to zero (becomes insignificant)

Big-O: Common Names

$O(1)$	constant (same as $O(k)$ for constant k)
$O(\log n)$	logarithmic
$O(n)$	linear
$O(n \log n)$	"$n \log n$"
$O(n^2)$	quadratic
$O(n^3)$	cubic
$O(n^k)$	polynomial (where is k is any constant)
$O(k^n)$	exponential (where k is any constant > 1)

Note: "exponential" does not mean "grows really fast", it means "grows at rate proportional to k^n for some k>1"

- A savings account accrues interest exponentially (k=1.01?)
- If you don't know k, you probably don't know it's exponential

Announcements

- TA office hours have been decided
  - Held at the 4th floor breakouts in CSE
    - Whiteboard area near the stairs/elevator

- HW1 released
  - Due Friday, July 2 at 11:00PM
  - See late day policy

- Optional *section* Thursdays 2:00 – 3:00pm
  - Room *TBD*
  - Getting started on HW1, Induction, Eclipse
  - Bring Questions!
  - Materials will be posted online