Algorithm Analysis I: Asymptotics

CSE 332 Spring 2021

Instructor: Hannah C. Tang

Teaching Assistants:

Aayushi Modi	Khushi Chaudhari	Patrick Murphy
Aashna Sheth	Kris Wong	Richard Jiang
Frederick Huyan	Logan Milandin	Winston Jodjana
Hamsa Shankar	Nachiket Karmarkar	

gradescope

gradescope.com/courses/256241

- Consider the following piece of code to find a value `k` in the array `arr`
- What, if any, assumptions does this code make about its inputs? What happens when those assumptions are violated?

```
boolean f(int[] arr, int k) {
  for(int i=0; i < arr.length; ++i) {
    if(arr[i] == k)
      return true;
    else if(arr[i] > k)
      return false;
  }
  return false;
}
```

Announcements

- Project 1 released!
  - Fill out the partner survey NOW (due by end of lecture)
  - Checkpoint is a Gradescope-administered survey, not released yet
  - `git pull` before you `git push` (we have a bugfix ***just for you***)
- Lecture activities were  graded  and returned
  - Remember that you get the points if the textbox is non-empty
- Office hours released!
  - Reasonable coverage for most timezones
  - Contact us to arrange a one-on-one if you can't find a good time

Lecture Outline

- **A Computational Model for Describing Algorithm Performance**
- Using the Model to Compare Algorithms
- Review: Logarithms and Exponents
- Big-Oh Definitions

Describing Algorithms: What Do We Care About?

- Correctness:
  - Does the algorithm do what is intended
- Performance:
  - Speed — **time complexity**
  - Memory — **space complexity**
- Other attributes:
  - Clarity, security, … equity?!?!

- Why analyze performance?
  - To make good design decisions
  - Enable you to examine an algorithm (or code) and identify bottlenecks

Q: How Should We Describe An Algorithms' Performance?

A: How Should We Describe An Algorithms' Performance?

- Uh, why NOT just run the program and time it??
  - Too much ***variability***; not reliable or ***portable***
    - Hardware: processor(s), memory, etc.
    - Firmware: OS, Java version, libraries, drivers
    - Other: implementation-specific quirks, other programs running, …
  - Choice of input
    - (Non-exhaustive) testing may ***miss*** worst-case input
    - Benchmarks don't ***describe*** or ***predict*** the relationship between input sizes

- Often want to evaluate an ***algorithm***, not an ***implementation***

An ***algorithm*** is more ***performant*** than another when, for sufficiently large inputs, it runs in less time *(our focus)* or less space than the other

Describing An Algorithms' Performance

An ***algorithm*** is more ***performant*** than another when, for sufficiently large inputs, it runs in less time or less space than the other

1. To be ***descriptive of large inputs*** (*n*), we need to understand:
   - What do we consider to be "large"?
     - If *n* is 10, probably any algorithm is fast enough

2. To characterize time (or space) without an implementation and its input, we need a computational model that's:
   - ***Independent*** of CPU, programming language, coding tricks, etc.
   - Rigorous and ***accurate***; able to predict performance without an implementation

A Computational Model for Algorithms (1 of 3)

- We abstract away the computer by counting:
  1. “elements” (space complexity)
  2. “operations” (time complexity)

- Remember: “Independent of CPU, programming language, coding tricks, etc.”

A Computational Model for Algorithms (2 of 3)

2. Basic ***elements*** take "some amount of" ***constant space***
   - Integers in an array
   - Nodes in a linked list
   - Etc.
   - (This is an *approximation of reality*: a very useful "lie".)

A Computational Model for Algorithms (3 of 3)

1. Basic ***operations*** take "some amount of" ***constant time***
   - Arithmetic
   - Assignment
   - Access one Java field **or array index**
   - Etc.
   - (Again, this is an *approximation of reality*)

Consecutive statements	Sum of time of each statement
Loops	Num iterations * time for loop body
Recurrence	Solve recurrence equation
Function Calls	Time of function's body ???
Conditionals	Time of condition + time of {slower/faster} branch

Which Branch To Analyze?

- Case Analysis != Asymptotic Analysis

- We generally talk about two cases:
  - ***Worst-case complexity***: max # steps algorithm takes on "most challenging" input of size N
  - ***Best-case complexity***: min # steps algorithm takes on "easiest" input of size N
  - (there are other cases, but they're harder to reason about)

- Unless otherwise stated, we usually refer to the ***worst case***
  - So we'll analyze the ***slower branch***

Examples: From Code to Our Model

```
b = b + 5
c = b / a
b = c + 100
```

```
for (i = 0; i < n; i++) {
   sum++;
 }
```

```
if (j < 5) {
   sum++;
} else {
   for (i = 0; i < n; i++) {
      sum++;
    }
}
```

Examples: From Code to Our Model

①

```
b = b + 5
c = b / a
b = c + 100
```

2
2
2
6

②

```
for (i = 0; i < n; i++) {
    sum++;
}
```

1 n
1 2
2

$1 + 5n$

③

```
if (j < 5) {
   sum++;
} else {
   for (i = 0; i < n; i++) {
      sum++;
   }
}
```

$1+5n$

$1+1+5n$

Another Example

```
int coolFunction(int n, int sum) {
  int i, j;
  for (i = 0; i < n; i++) {
    for (j = 0; j < n; j++) {
      sum++;
    }
  }
  print "This program is great!";
  for (i = 0; i < n; i++) {
    sum++;
  }
  return sum
}
```

Another Example

```
int coolFunction(int n, int sum) {
  int i, j;
  for (i = 0; i < n; i++) {
    for (j = 0; j < n; j++) {
      sum++;
    }
  }
  print "This program is great!";
  for (i = 0; i < n; i++) {
    sum++;
  }
  return sum
}
```

$25n^2 + 6$

$5n+1$

$5(5n+1)+1$

1

$5n+1$

$5n+1$

1

Analyzing Loops, Formally

- In this model, we use summations to quantify the runtime

```
for (i = 0; i < n; i++) {
  sum++;
}
```

Analyzing Loops, Formally

- In this model, we use summations to quantify the runtime

1 2

```
for (i = 0; i < n; i++) {
    sum++;
}
```

2 n

$$\sum_{i=0}^{n-1} 5 \Rightarrow 5n$$

gradescope

gradescope.com/courses/256241

- What is the precise expression that describes `findSorted()`'s runtime as a function of `n = arr.length`?

```
// Requires arr to be sorted
// Returns whether k is in array
boolean findSorted(int[] arr, int k) {
  for(int i=0; i < arr.length; ++i) {
    if(arr[i] == k)
      return true;
    else if(arr[i] > k)
      return false;
  }
  return false;
}
```

Example Soln: Linear Search

- What is the precise expression that describes `findSorted()`'s runtime as a function of `n = arr.length`?

```
// Requires arr to be sorted
// Returns whether k is in array
boolean findSorted(int[] arr, int k) {
  for (int i=0; i < arr.length; ++i) {
    if (arr[i] == k)
      return true;
    else if (arr[i] > k)
      return false;
  }
  return false;
}
```

2

$8n + 2$

gradescope

gradescope.com/courses/256241

- Assuming the following value for `arr`, what values for `k` yield the best and worst runtimes?

2	3	5	16	37	50	73	75	126

```
// Requires arr to be sorted
// Returns whether k is in array
boolean findSorted(int[] arr, int k) {
  for(int i=0; i < arr.length; ++i) {
    if(arr[i] == k)
      return true;
    else if(arr[i] > k)
      return false;
  }
  return false;
}
```

Best k:

Worst k:

Worst Case = Slower Branch; Best Case = ???

- Assuming the following value for `arr`, what values for `k` yield the best and worst runtimes?

2	3	5	16	37	50	73	75	126

```
// Requires arr to be sorted
// Returns whether k is in array
boolean findSorted(int[] arr, int k) {
  for(int i=0; i < arr.length; ++i) {
    if(arr[i] == k)
      return true;
    else if(arr[i] > k)
      return false;
  }
  return false;
}
```

Best k: 2

Worst k: 126

Modeling an Algorithm's Cases

- What is the precise expression that describes `findSorted()`'s best and worst runtimes ***independent of its inputs***?

```
// Requires arr to be sorted
// Returns whether k is in array
boolean findSorted(int[] arr, int k) {
  for(int i=0; i < arr.length; ++i) {
    if(arr[i] == k)
      return true;
    else if(arr[i] > k)
      return false;
  }
  return false;
}
```

2

Best case runtime: 8

A constant!

Worst case runtime: 8n+2

A linear function!

Lecture Outline

- A Computational Model for Describing Algorithm Performance
- **Using the Model to Compare Algorithms**
- Review: Logarithms and Exponents
- Big-Oh Definitions

Remember a faster search algorithm?

Comparing Algorithms (1 of 2)

- "Binary search is $O(\texttt{log}\ n)$ and linear is $O(n)$ "
  - But which algorithm is faster?
  - Depending on ***specific case***, ***constant factors***, and ***size of n*** linear search could be faster!

1. ***Specific case:***
   - For now, we'll use worst case
2. ***Constant factors***:
   - How *many* assignments, additions, etc. for each *n*
3. ***Size of n:***
   - Remember: "Descriptive of large inputs"*
   - So we pick n→∞ as our definition of "large"

** slide 9*

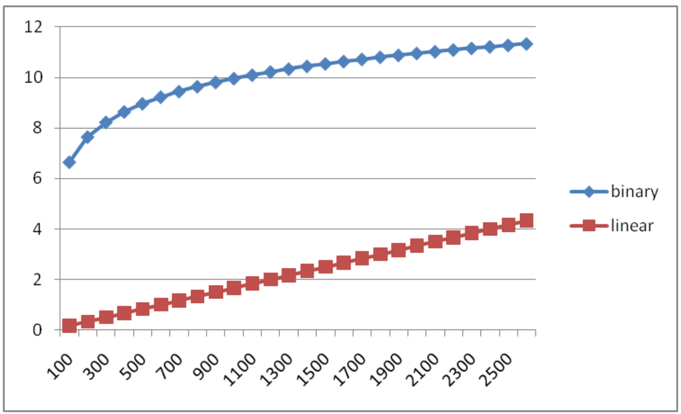
Comparing Algorithms (2 of 2)

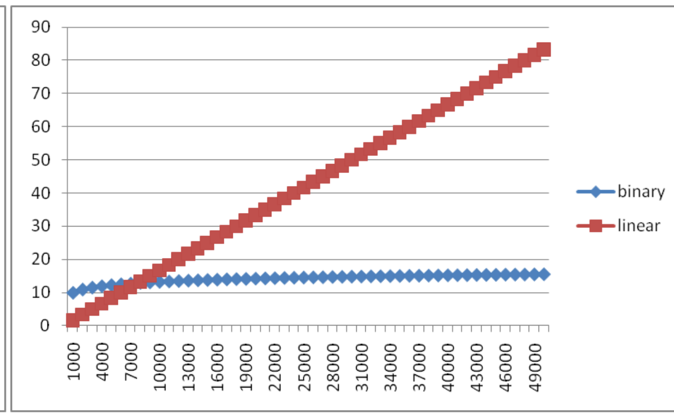
- How formalize the idea of how an algorithm behaves as N→∞?
  - There exists some n_0 such that for all $n > n_0$ ***binary search "wins"***

- Let's play with a couple plots to get some intuition…

Example: Binary Search vs Linear Search

- Let's "help" linear search "win"
  - Run it on a computer 100x as fast (say 2018 model vs. 1990)
  - Use a new compiler/language that is 3x as fast
  - Be a clever programmer to eliminate half the work
  - Each iteration is 600x as fast as in binary search





When we're dealing with infinity, constants and lower-order terms don't meaningfully add to the final result

Intuitive Simplifications

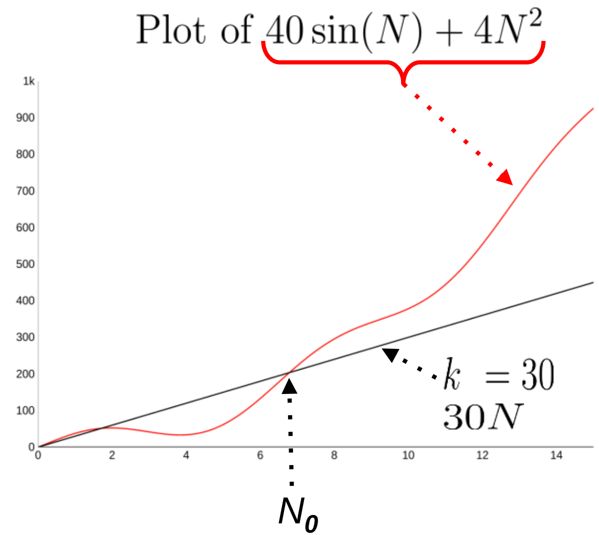
- When we're dealing with infinity, constants and lower-order terms don't meaningfully add to the final result

- (1) Eliminate lower-order terms
  - $6 + \frac{1}{2}N^2 + \frac{3}{2}N + 1 + \frac{1}{2}N^2 + \frac{1}{2}N + \frac{1}{2}N^2 - \frac{1}{2}N + N^2 + N$
  - $\cancel{6} + \frac{1}{2}N^2 + \cancel{\frac{3}{2}N} + 1 + \frac{1}{2}N^2 + \cancel{\frac{1}{2}N} + \frac{1}{2}N^2 - \cancel{\frac{1}{2}N} + N^2 + \cancel{N}$
  - $\frac{5}{2}N^2$

- (2) Ignore multiplicative constants
  - $\cancel{\frac{5}{2}}N^2$
  - N^2

Why Does This Work?

Demo: https://www.desmos.com/calculator/rl25eewwe3



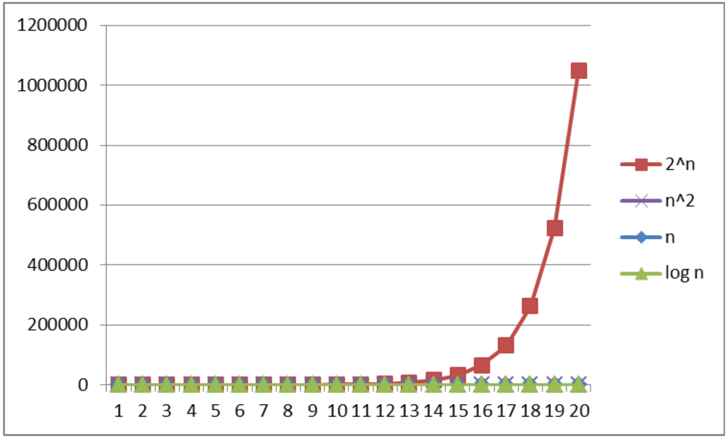
Lecture Outline

- A Computational Model for Describing Algorithm Performance
- Using the Model to Compare Algorithms
- **Review: Logarithms and Exponents**
- Big-Oh Definitions

Logarithms and Exponents

- Definition: $\log_2 x = y$ if $x = 2^y$
  - Note: since so much is binary in CS, $\log$ almost always means $\log_2$

- Just as exponents grow *very* quickly, logarithms grow *very* slowly
  - So, $\log_2$ 1,000,000 = "a little under 20"



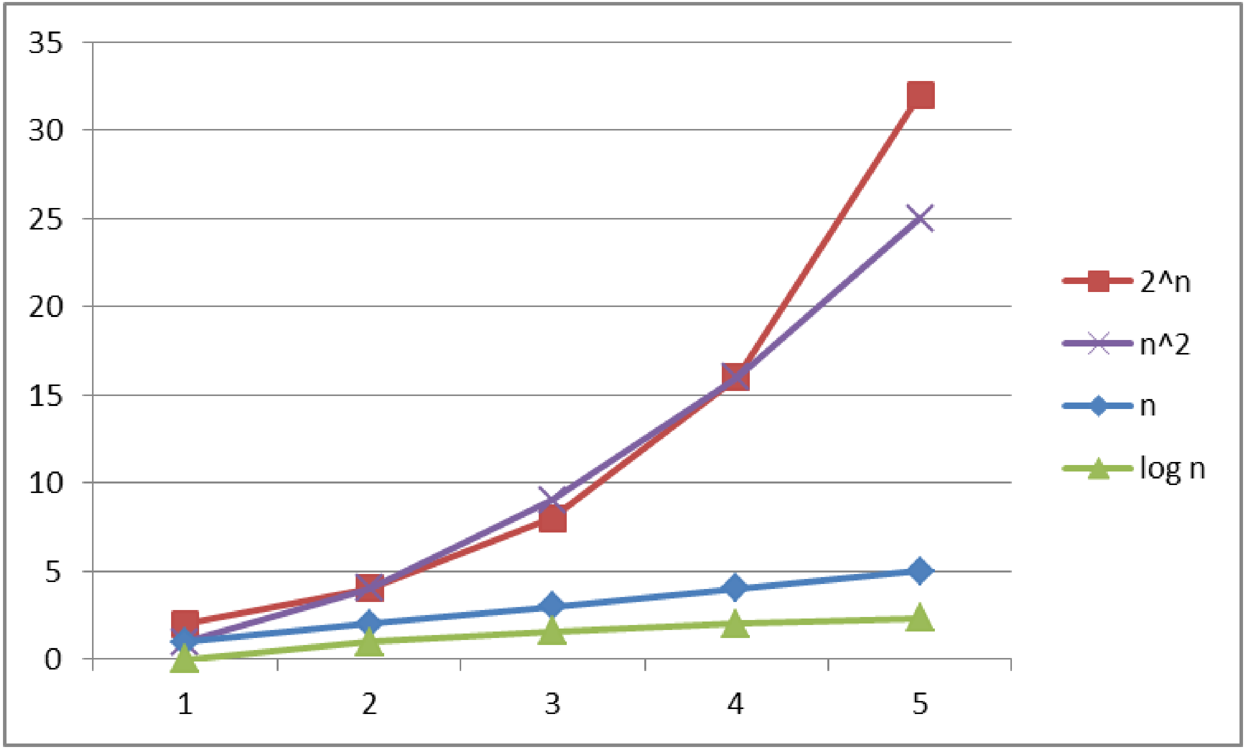
Log base doesn't matter (much)

- "Any base *B* log is equivalent to base 2 log within a constant factor"
  - ***And we are about to prove constant factors don't matter!***
  - In particular, $\log_2 x = 3.22 \log_{10} x$
- Why a constant multiplier ?
  - $\log_B x = (\log_A x) / (\log_A B)$

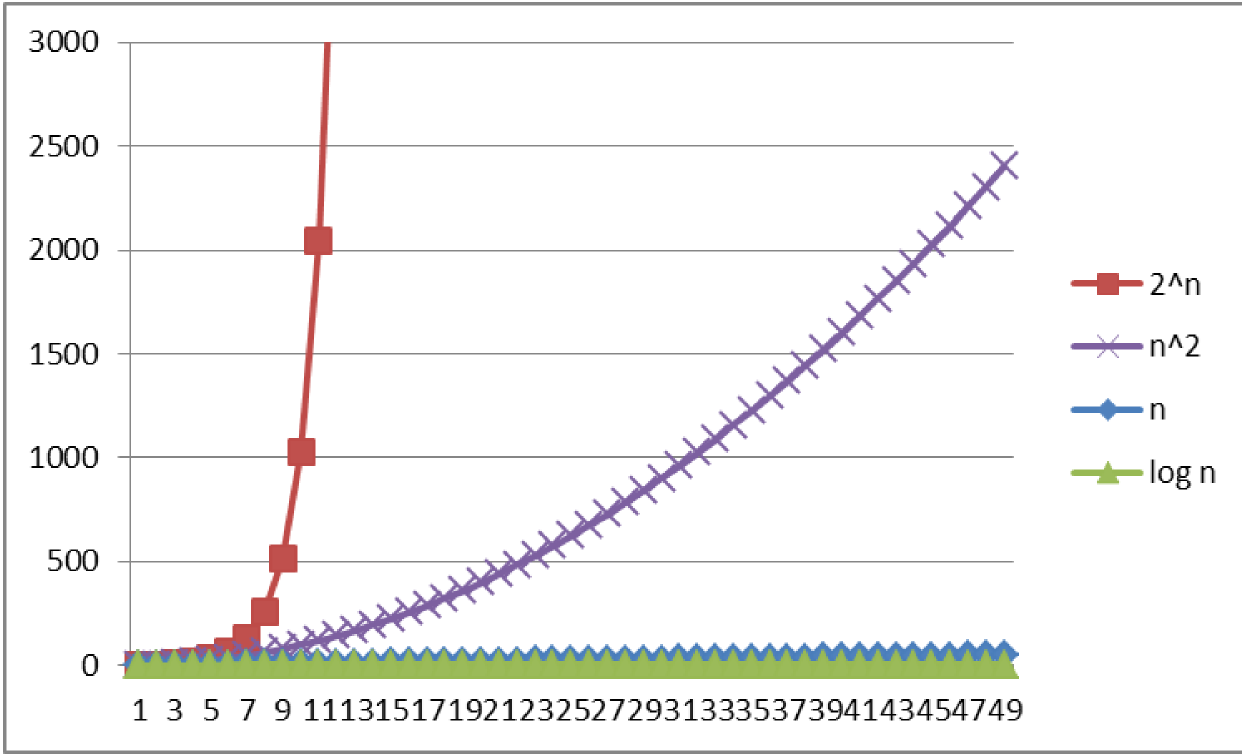
Review: Properties of logarithms

- **`log(A*B) = log A + log B`**
  - So **`log(N`**^{k}**`)= k log N`**
- **`log(A/B) = log A – log B`**
- **`x =`** $\log_2 2^x$
- **`log(log x)`** is written **`log log x`**
  - Grows as slowly as 2^{2^y} grows fast
  - Ex: $\log_2 \log_2 4billion \sim \log_2 \log_2 2^{32} = \log_2 32 = 5$
- **`(log x)(log x)`** is written **`log`**^{2}**`x`**
  - It is greater than **`log x`** for all **`x > 2`**

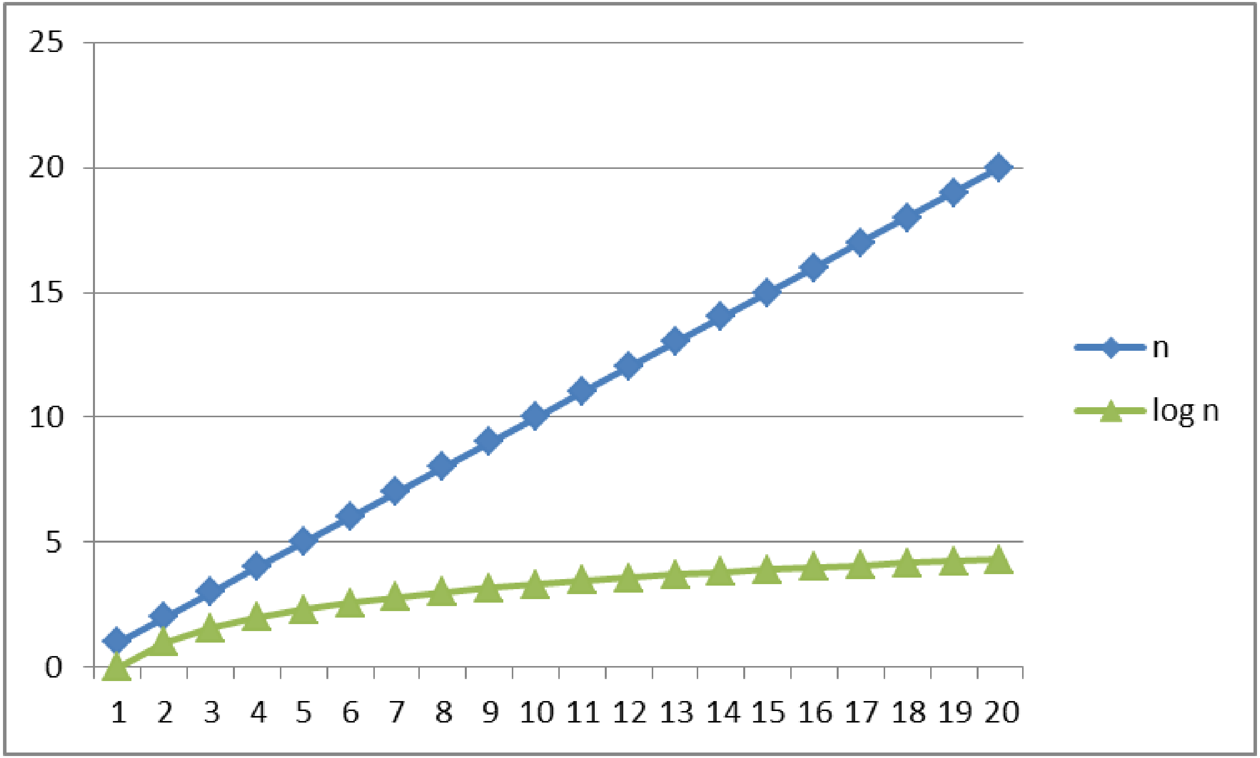
Logarithms and Exponents



Logarithms and Exponents



Logarithms and Exponents



Lecture Outline

- A Computational Model for Describing Algorithm Performance
- Using the Model to Compare Algorithms
- Review: Logarithms and Exponents
- **Big-Oh Definition**

Introduction: Asymptotic Notation

- About to show formal definition, which amounts to our earlier intuitive simplifications:
  - Eliminate lower-order terms
  - Ignore multiplicative constants

- Examples:
  - $4n + 5$
  - $0.5n \log n + 2n + 7$
  - $n^3 + 2^n + 3n$
  - $n \log (10n^2)$

Big-Oh relates functions

- We use O on a function f(n) (for example n^2) to mean *the set of functions with asymptotic behavior* ***less than or equal to*** f(n)

- So ($3n^2$+17) **is in** $O(n^2)$
  - $3n^2$+17 and n^2 have the same **asymptotic behavior**

- Confusingly, we also say/write:
  - ($3n^2$+17) **is** $O(n^2)$
  - ($3n^2$+17) $\in$ $O(n^2)$
  - ($3n^2$+17) = $O(n^2)$ ← *least ideal*

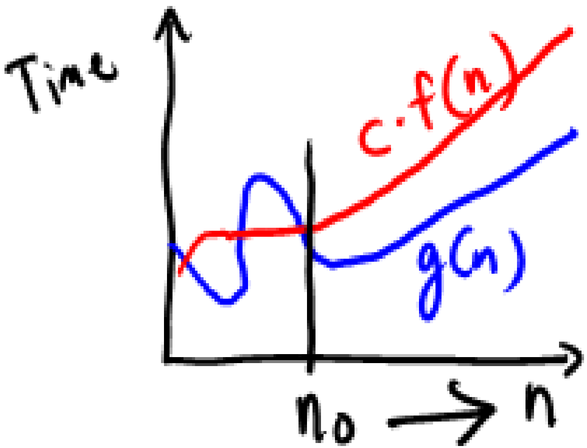
- But we would never say $O(n^2)$ = ($3n^2$+17)

Big-Oh, Formally (1 of 3)

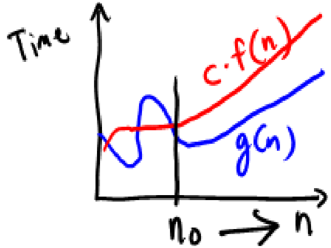
Definition: $g(n)$ is in $O(\, f(n)\,)$ iff there exist positive constants c and n_0 such that

$g(n) \leq c\, f(n)$ for all $n \geq n_0$

Note: $n_0 \geq 1$ (and a natural number) and $c > 0$



Big-Oh, Formally (2 of 3)



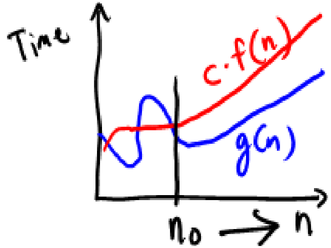
Definition: g(n) is in O(f(n)) iff there exist positive constants c and n_0 such that

g(n) ≤ c f(n)   for all $n \geq n_0$

Note: $n_0 \geq 1$ (and a natural number) and $c > 0$

To show g(n) is in O(f(n)), pick a c large enough to "cover the constant factors" and n_0 large enough to "cover the lower-order terms"

Big-Oh, Formally (3 of 3)



Definition: g(n) is in O(f(n)) iff there exist positive constants c and n_0 such that

g(n) ≤ c f(n) for all $n \geq n_0$

Note: $n_0 \geq 1$ (and a natural number) and $c > 0$

Example: Let g(n) = 3n + 4 and f(n) = n

c = 4 and n_0 = 5 is one possibility

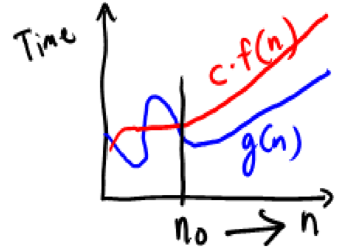
Example: Let g(n) = 3n + 4 and f(n) = n^5

c = 3 and n_0 = 2 is one possibility

Example: Let g(n) = 3n + 4 and f(n) = 2^n

c = 100000000 and n_0 = 1 is one possibility

Big-Oh, Formally (3 of 3)



> Definition: g(n) is in O(f(n)) iff there exist positive constants c and n_0 such that
>
> $g(n) \le c\, f(n)$ for all $n \ge n_0$

Note: $n_0 \ge 1$ (and a natural number) and $c > 0$

Example: Let g(n) = 3n + 4 and f(n) = n

c = 4 and n_0 = 5 is one possibility

$3n+4 \le 4n \quad \forall n \ge 5$

$\therefore 3n+4 \in O(n)$

Example: Let g(n) = 3n + 4 and f(n) = n^5

c = 3 and n_0 = 2 is one possibility

$3n+4 \le 3n^5 \quad \forall n \ge 2$

$\therefore 3n+4 \in O(n^5)$

Example: Let g(n) = 3n + 4 and f(n) = 2^n

c = 100000000 and n_0 = 1 is one possibility

$3n+4 \le 100000000 \cdot 2^n \quad \forall n \ge 1$

$\therefore 3n+4 \in O(2^n)$

Summary

- Complexity analyses use simplified cost models

- Asymptotic analysis can take liberties with mathematical expressions because it deals with infinity
  - Eg, dropping lower-order terms and constants
  - But it gives us a common "frame of reference" with which to compare algorithms, too!

- Case Analysis != Asymptotic Analysis
  - Case analysis is a different axis on which to evaluate runtime and space

- Review your log rules!