Asymptotic Analysis II

CSE 373
Data Structures & Algorithms
Ruth Anderson
Autumn 2012

Today's Outline

- **Announcements**
  - Assignment #1, due Thurs, Oct 4 at 11pm
  - Assignment #2, posted later this week, due Fri Oct 12 at BEGINNING of lecture
- **Algorithm Analysis**
  - Big-Oh
  - Analyzing code

Ignoring constant factors

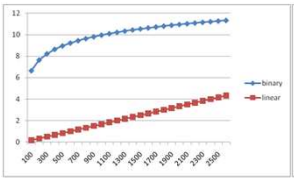
- So binary search is $O(\log n)$ and linear search is $O(n)$
  - But which is faster?
- Could depend on constant factors:
  - How *many* assignments, additions, etc. for each n
    - *E.g.* $T(n) = 5{,}000{,}000n$ vs. $T(n) = 5n^2$
  - And could depend on size of n *(if n is small then constant additive factors could be more important)*
    - *E.g.* $T(n) = 5{,}000{,}000 + \log n$ vs. $T(n) = 10 + n$
- **But** 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…

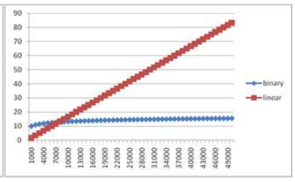
Linear Search vs. Binary Search

Let's try to "help" linear search:

- Run it on a computer 100x as fast (say 2010 model vs. 1990)
- Use a new compiler/language that is 3x as fast
- Be a clever programmer to eliminate half the work
- So doing each iteration is 600x as fast as in binary search

For small n, linear search is faster! But eventually binary search wins.





10/03/2012 4

Asymptotic notation

About to show formal definition of Big-O, which amounts to saying:

1. Eliminate low-order terms
2. Eliminate coefficients

Examples:

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

10/03/2012 5

Examples

True or false?

1. 4+3n is O(n)
2. n+2logn is O(logn)
3. logn+2 is O(1)
4. n^{50} is $O(1.1^n)$

10/03/2012 6

Examples

True or false?

1. 4+3n is O(n) — True
2. n+2logn is O(logn) — False
3. logn+2 is O(1) — False
4. n^{50} is $O(1.1^n)$ — True

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** $O(n^2)$

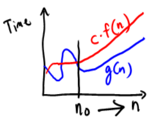
- $3n^2+17$ and n^2 have the same asymptotic behavior

Confusingly, we also say/write:

- $(3n^2+17)$ **is** $O(n^2)$
- $(3n^2+17) \in O(n^2)$
- $(3n^2+17) = O(n^2)$

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

Formally Big-Oh



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 $n \geq n_0$

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"

- Example: Let **g(*n*)** $= 3n^2+17$ and **f(*n*)** $= n^2$

  $c = 5$ and $n_0 = 10$ is more than good enough

This is "less than or equal to"

- So $3n^2+17$ is also $O(n^5)$ and $O(2^n)$ etc.

Using the definition of Big-Oh (Example 1)

Def'n:
$g(n)$ is in $O(f(n))$ iff there exist positive constants c and n_0 s.t. $g(n) \le c\ f(n)$ for all $n \ge n_0$

Given: $g(n) = 1000n$ & $f(n) = n^2$
Prove: $g(n)$ is in $O(f(n))$

- A valid proof is to find valid c & n_0
- Try: $n_0 = 1000$, $c = 1$
- Also: $n_0 = 1$, $c = 1000$

10/07/2011 10

Using the definition of Big-Oh (Example 2)

Def'n:
$g(n)$ is in $O(f(n))$ iff there exist positive constants c and n_0 s.t. $g(n) \le c\ f(n)$ for all $n \ge n_0$

Given: $g(n) = 4n$ & $f(n) = n^2$
Prove: $g(n)$ is in $O(f(n))$

- A valid proof is to find valid c & n_0
- When n=4, $g(n) = 16$ & $f(n) = 16$; this is the crossing over point
- So we can choose $n_0 = 4$, and $c = 1$
- Note: There are many possible choices:
  ex: $n_0 = 78$, and $c = 42$ works fine

10/03/2012 11

Using the definition of Big-Oh (Example 3)

Def'n:
$g(n)$ is in $O(f(n))$ iff there exist positive constants c and n_0 s.t. $g(n) \le c\ f(n)$ for all $n \ge n_0$

Given: $g(n) = n^4$ & $f(n) = 2^n$,
Prove: $g(n)$ is in $O(f(n))$

- A valid proof is to find valid c & n_0
- One possible answer: $n_0 = 20$, and $c = 1$

10/03/2012 12

What's with the c?

- To capture this notion of similar asymptotic behavior, we allow a constant multiplier (called c)
- Consider:
  g(n) = 7n+5
  f(n) = n
- These have the same asymptotic behavior (linear), so g(n) is in O(f(n)) even though g(n) is always larger
- There is no positive n_0 such that g(n) ≤ f(n) for all n ≥ n_0
- The 'c' in the definition allows for that:
  $g(n) \leq c\ f(n)$ for all $n \geq n_0$
- To prove g(n) is in O(f(n)), have c = 12, n_0 = 1

Big Oh: Common Categories

From fastest to slowest:

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

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

More Definitions

- **Upper bound**: O(**f(n)**) is the set of all functions asymptotically less than or equal to **f(n)**
  - g(n) is in O(**f(n)**) if there exist positive constants c and n_0 such that
    $g(n) \leq c\ f(n)$ for all $n \geq n_0$
- **Lower bound**: Ω(**f(n)**) is the set of all functions asymptotically greater than or equal to **f(n)**
  - g(n) is in Ω(**f(n)**) if there exist positive constants c and n_0 such that
    $g(n) \geq c\ f(n)$ for all $n \geq n_0$
- **Tight bound**: θ(**f(n)**) is the set of all functions asymptotically equal to **f(n)**
  - g(n) is in θ(**f(n)**) if **both**: g(n) is in O(**f(n)**) AND
    g(n) is in Ω(**f(n)**)

Even More Definitions…

O(**f(n)**) is the set of all functions asymptotically less than or equal to **f(n)**

- g(n) is in *O*(**f(n)**) if there exist positive constants c and n_0 such that

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

o(**f(n)**) is the set of all functions asymptotically less than **f(n)**

- g(n) is in o(**f(n)**) if for any positive constant c, there exists a positive constant n_0 such that

  g(n) $<$ **c f(n)** for all $n \geq n_0$

Ω(**f(n)**) is the set of all functions asymptotically greater than or equal to **f(n)**

- g(n) is in Ω(**f(n)**) if there exist positive constants c and n_0 such that

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

ω(**f(n)**) is the set of all functions asymptotically greater than **f(n)**

- g(n) is in ω(**f(n)**) if for any positive constant c, there exists a positive constant n_0 such that

  g(n) $>$ **c f(n)** for all $n \geq n_0$

Big-Omega et al. Intuitively

Asymptotic Notation	Mathematics Relation
O	≤
Ω	≥
Θ	=
o	<
ω	>

Types of Analysis

Two orthogonal axes:

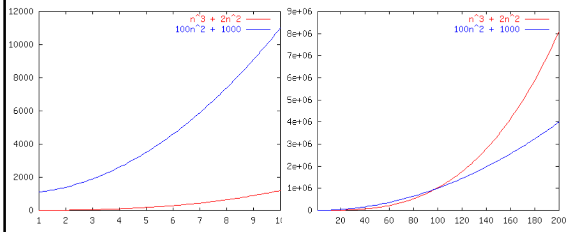
- bound flavor (usually we talk about upper or tight)
  - upper bound (O, o)
  - lower bound (Ω, ω)
  - asymptotically tight (Θ)
- analysis case (usually we talk about worst)
  - worst case (adversary)
  - average case
  - best case
  - "amortized"

Which Function Grows Faster?

$n^3 + 2n^2$ vs. $100n^2 + 1000$

Which Function Grows Faster?

$n^3 + 2n^2$ vs. $100n^2 + 1000$

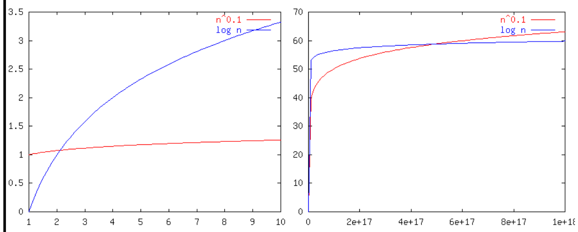


Which Function Grows Faster?

$n^{0.1}$ vs. $\log n$

Which Function Grows Faster?

$n^{0.1}$ vs. $\log n$



10/03/2012 22

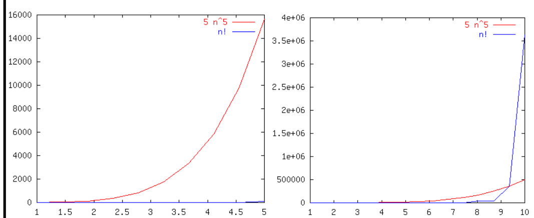
Which Function Grows Faster?

$5n^5$ vs. $n!$

10/03/2012 23

Which Function Grows Faster?

$5n^5$ vs. $n!$



10/03/2012 24

Nested Loops

```
for i = 1 to n do
  for j = 1 to n do
    sum = sum + 1

for i = 1 to n do
  for j = 1 to n do
    sum = sum + 1
```


More Nested Loops

```
for i = 1 to n do
  for j = 1 to n do
    if (cond) {
            do_stuff(sum)
      } else {
            for k  = 1 to n*n
                  sum += 1
```


Big-Oh Caveats

- Asymptotic complexity (Big-Oh) focuses on behavior for **large *n*** and is independent of any computer / coding trick
  - But you can "abuse" it to be misled about trade-offs
  - Example: $n^{1/10}$ vs. `log` *n*
    - Asymptotically $n^{1/10}$ grows more quickly
    - But the "cross-over" point is around $5 * 10^{17}$
    - So if you have input size less than 2^{58}, prefer $n^{1/10}$
- Comparing O() for ***small n*** values can be misleading
  - Quicksort: O(nlogn) (expected)
  - Insertion Sort: $O(n^2)$ (expected)
  - Yet in reality Insertion Sort is faster for small n's
  - We'll learn about these sorts later

Addendum: Timing vs. Big-Oh?

- At the core of CS is a backbone of theory & mathematics
  - Examine the algorithm itself, mathematically, not the implementation
  - Reason about performance as a function of n
  - Be able to mathematically prove things about performance
- Yet, timing has its place
  - In the real world, we do want to know whether implementation A runs faster than implementation B on data set C
  - Ex: Benchmarking graphics cards
  - We will do some timing in our homeworks
- Evaluating an algorithm? Use asymptotic analysis
- Evaluating an implementation of hardware/software? Timing can be useful

10/03/2012 28