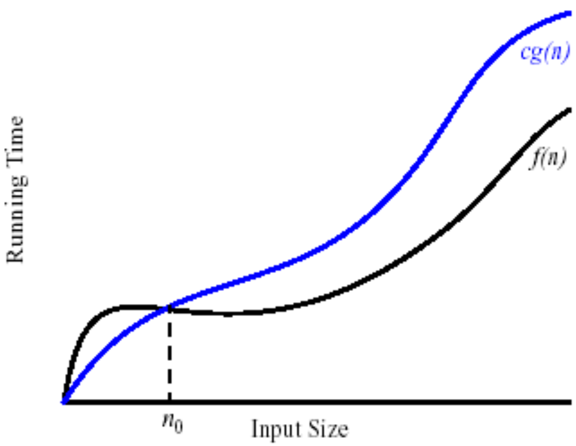


CSSE 230 Day 2

Growable Arrays Continued

Big-O notation

Submit Growable Array exercise

Agenda and goals

- Growable Array recap
- Big-Oh definition

- After today, you'll be able to
  - Use the term *amortized* appropriately in analysis
  - State the formal definition of big-Oh notation

Announcements

- All should do piazza introduction post (a few students left)
- Turn in the GrowableArray exercise now.
- Quiz problems 1-5. Do on your own, then compare with a neighbor.

You must demonstrate programming competence on exams to succeed

- See syllabus for exam weighting and caveats.
- Evening exams (Thu Week 3, Wed Week 8)
- Think of every program you write as a practice test
  - For example, HW4 → Exam 2

Properties of Logarithms & Exponentials

Properties of logarithmic functions

$$log_b(xy) = log_b(x) + log_b(y)$$

$$log_b(x/y) = log_b(x) - log_b(y)$$

$$log_b(x^\alpha) = \alpha\, log_b(x)$$

$$log_b(x) = \frac{log_a(x)}{log_a(b)}$$

$$a^{log_b(n)} = n^{log_b(a)}$$

Properties of exponential functions

$$a^{(b+c)} = a^b a^c$$

$$a^{bc} = (a^b)^c$$

$$a^b/a^c = a^{(b-c)}$$

$$b = a^{log_a(b)}$$

$$b^c = a^{c*log_a(b)}$$

Questions?

- About Homework 1?
  - Aim to complete ASAP, since it is due after next class
  - It is substantial
  - The last problem (the table) is worth lots of points!
- About the Syllabus?

Warm Up and Stretching thoughts

- Short but intense! ~50 lines of code total in our solutions

- Be sure to read the description of how it will be graded. Note how style will be graded.

- **Demo:** Use Git to check out the project

- **Demo:** Running the JUnit tests for test, file, package, and project

Iterative Code Analysis Examples

How many times does `sum++` run?

```
for (i = 4; i < n; i++)
    for (j = 0; j <= n; j++)
        sum++;
```

Why is this one so easy?

What if inner were `for (j = 0; j <= i; j++)` ?

Iterative Code Analysis Examples

How many times does `sum++` run?

```
for (i = 1; i <= n; i *= 2)
    sum++;
```

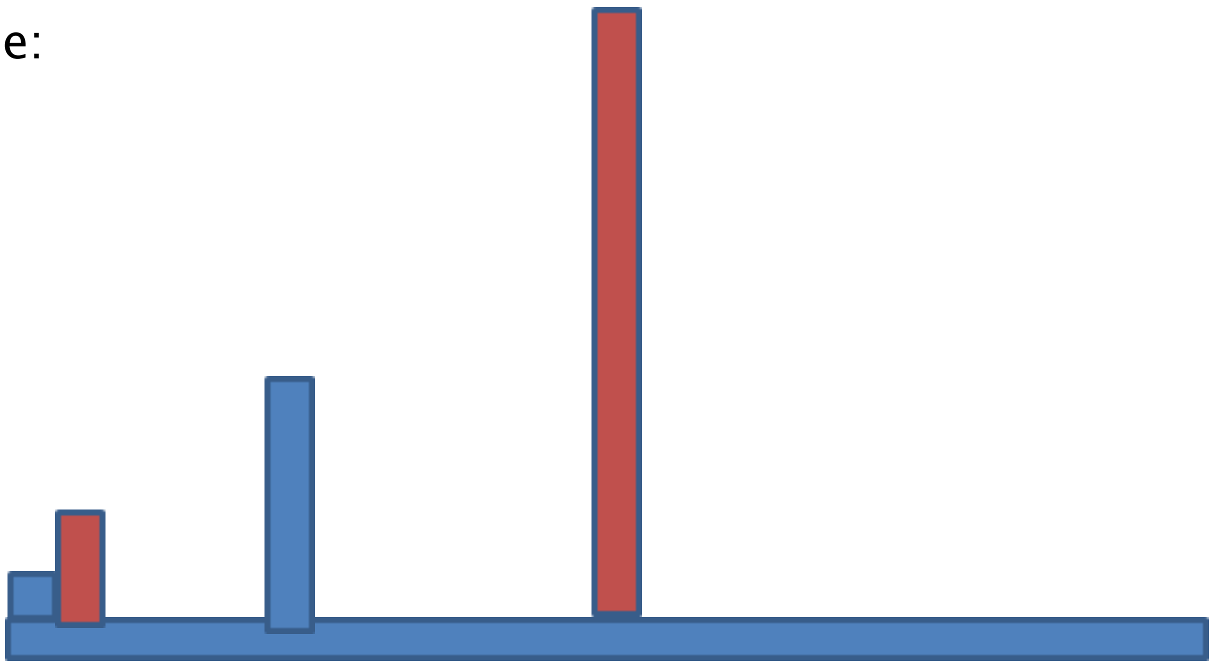
Be precise, using floor/ceiling as needed, to get full credit.

Growable Arrays Exercise

Solution

Worst-case vs amortized cost for adding an element to an array using the doubling scheme

Worst-case:
O(n)



amortized:
O(1)

Note: amortized is not the same as average case!

- average case: averaged over *input domain*.
- amortized cost: per-operation cost when undergoing *a sequence of operations*.

Conclusions

- What's the **amortized** and **worst-case** costs of adding an additional string…
  - in the doubling strategy?
  - in the add-one strategy?

- For which strategy is amortized analysis meaningful?
  - "When …a worst-case bound for a sequence of operations is better than the corresponding bound obtained by considering each operation separately and can be spread evenly to each operation in the sequence…" —Weiss, p.845
  - I.e., when amortized runtime is better than worst-case runtime

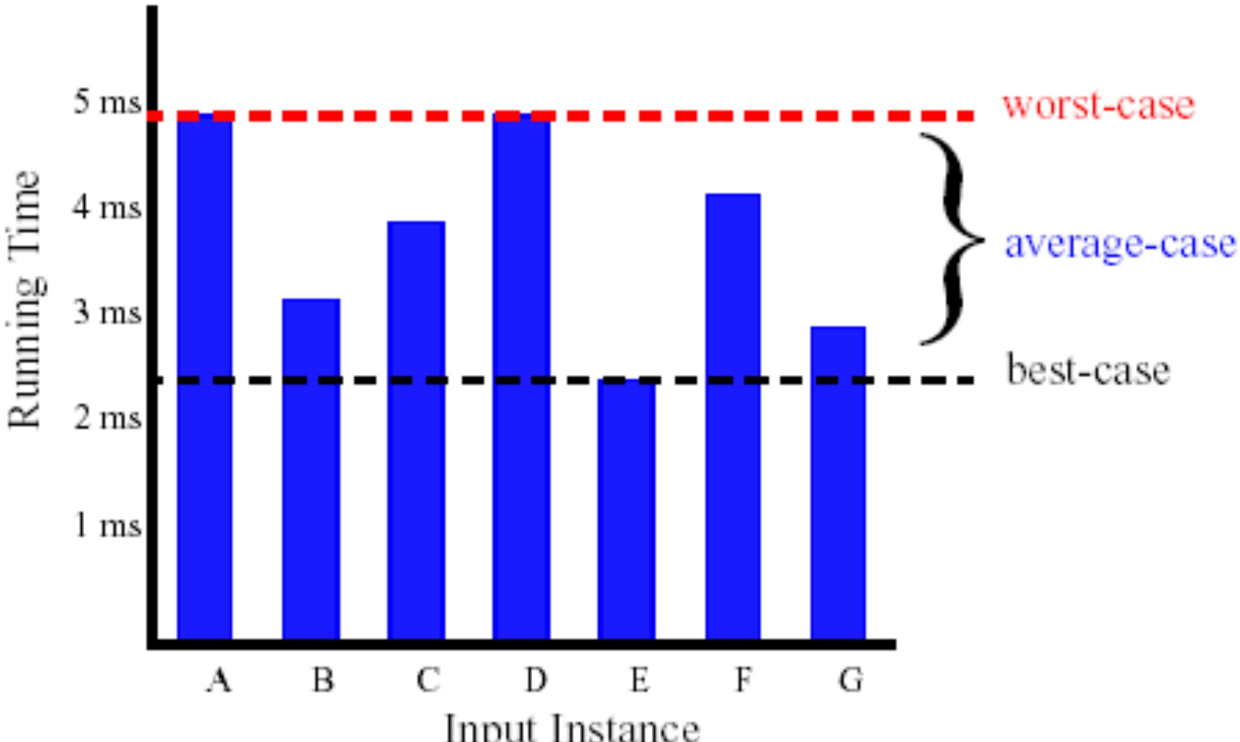
- Are there any hypothetical cases where we would prefer the slower strategy?

Algorithm Analysis: Running Time

Running Times

- Algorithms may have different *time complexity* on different data sets
- What do we mean by "Worst Case"?
- What do we mean by "Average Case"?
- What are some application domains where knowing the Worst Case time complexity would be important?
- http://cacm.acm.org/magazines/2013/2/160173-the-tail-at-scale/fulltext

Average Case and Worst Case



Note: amortized is not the same as average case!
- average case: averaged over *input domain*. "Expected runtime"
- amortized cost: per-operation cost when undergoing *a sequence of operations*. "Guaranteed runtime, when amortized to a per-operation basis"

Notation for Asymptotic Analysis

Big-O

Asymptotic Analysis

- Rule of thumb: we only care what happens as N (input size) gets large

- Is the runtime linear? quadratic? exponential? in N

Figure 5.1

Running times for small inputs

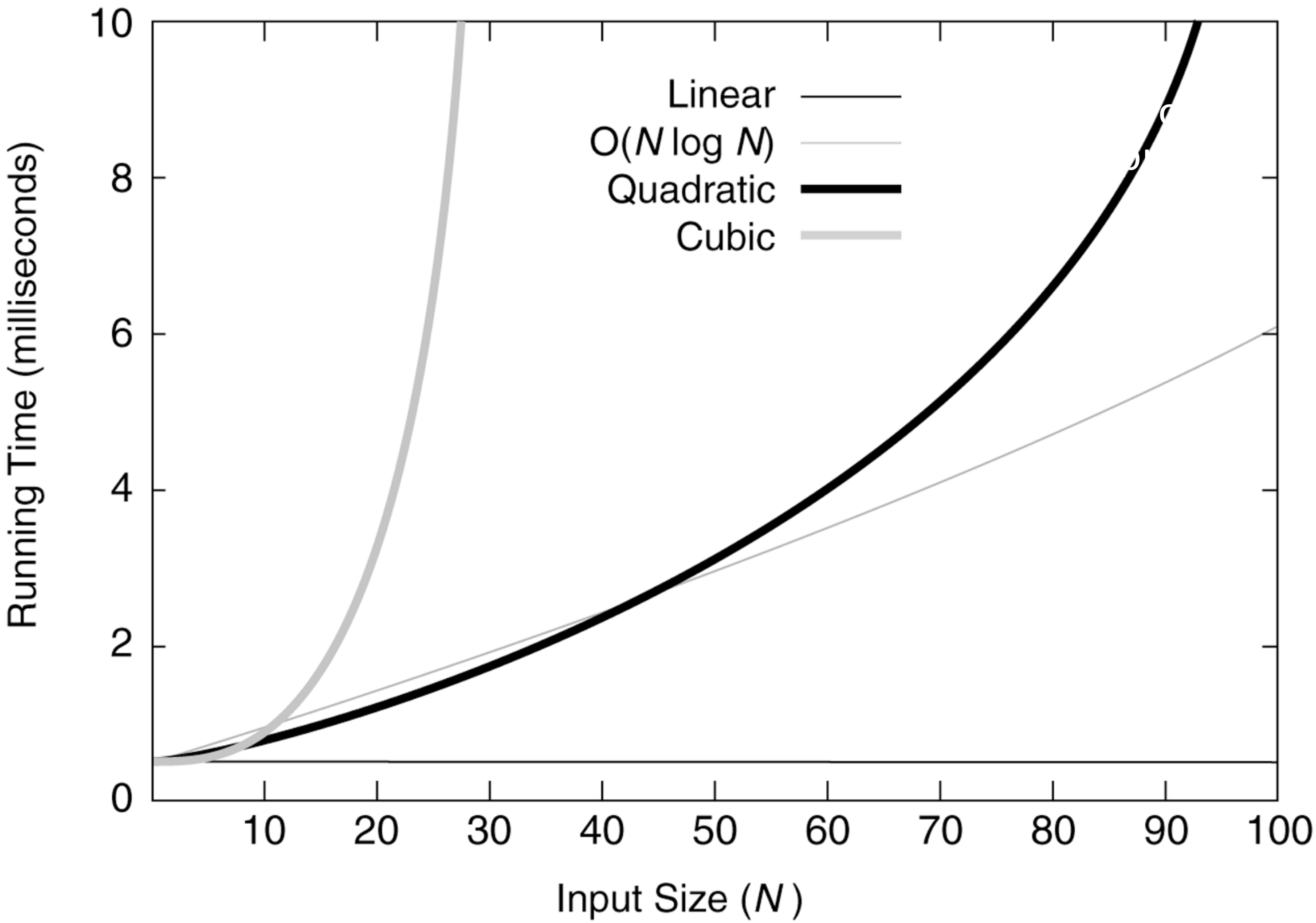


Figure 5.2

Running times for moderate inputs

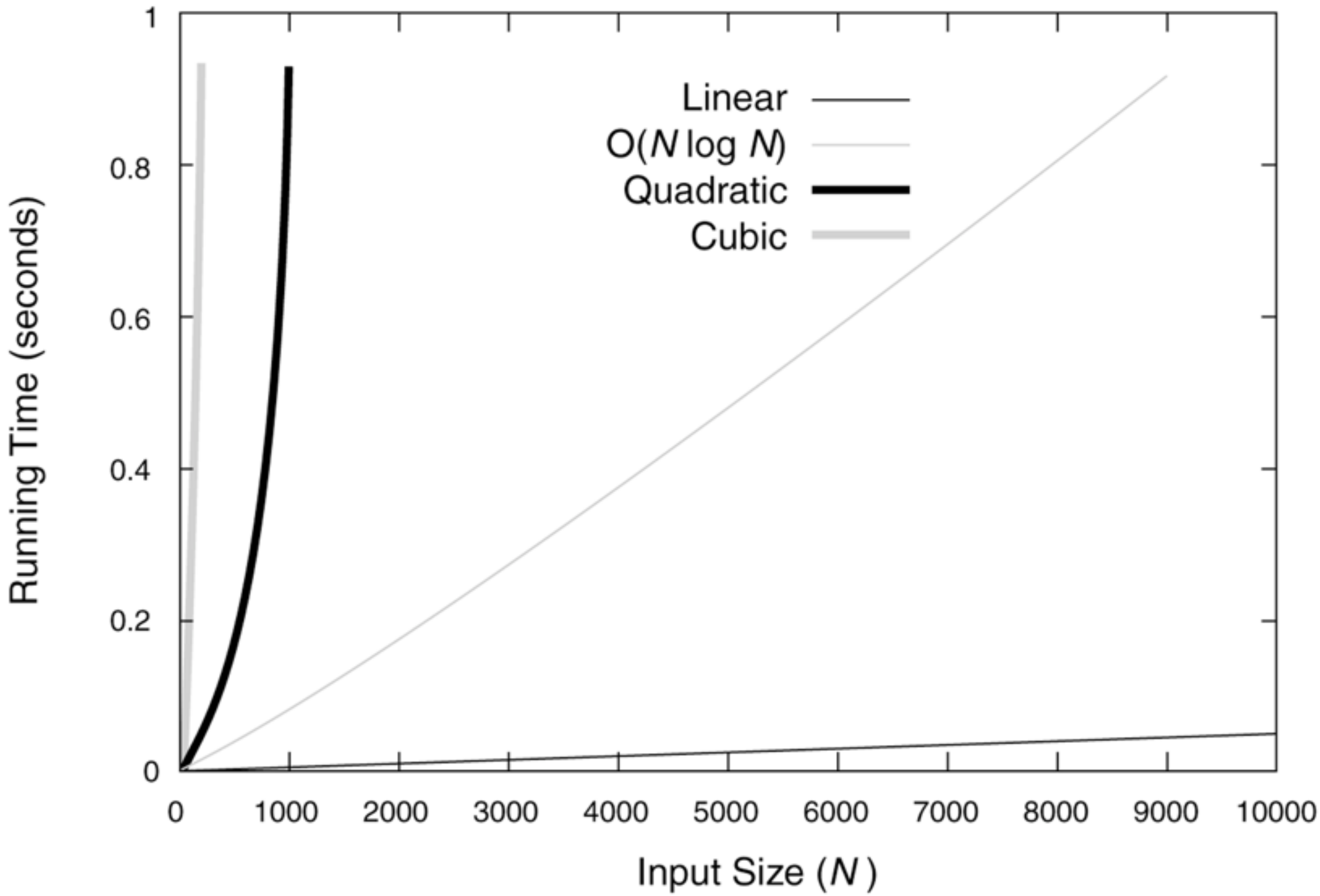


Figure 5.3

Functions in order of increasing growth rate

FUNCTION	NAME
c	Constant
$\log N$	Logarithmic
$\log^2 N$	Log-squared
N	Linear
$N \log N$	N log N
N^2	Quadratic
N^3	Cubic
2^N	Exponential

The answer to most big-O questions is one of these functions

a.k.a "log linear"

Simple Rule for Big-O (informal)

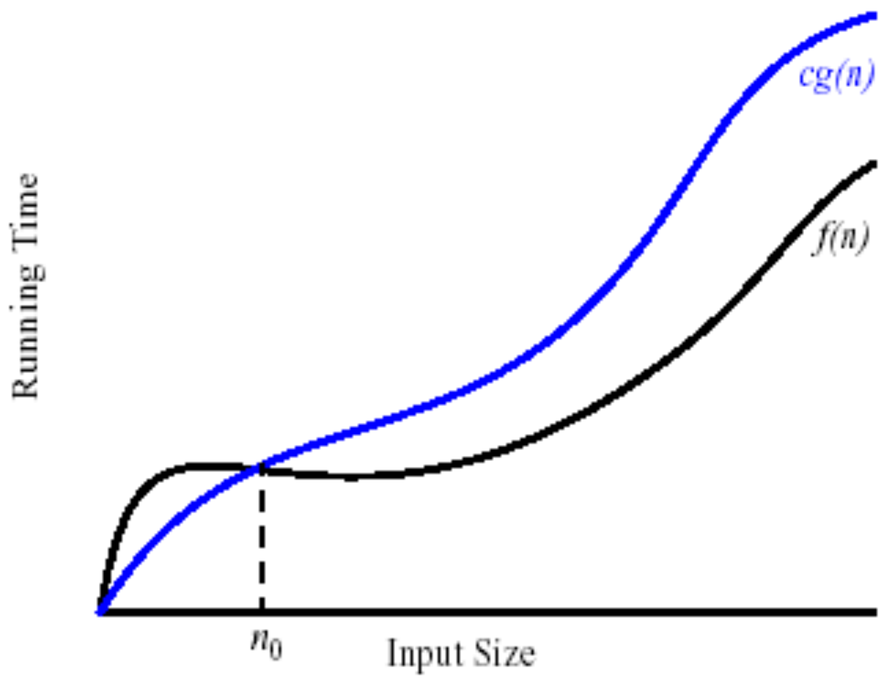
- Drop lower order terms and constant factors
- **$7n - 3$** is **$O(n)$**
- **$8n^2\log n + 5n^2 + n$** is **$O(n^2\log n)$**

Formal Definition of Big-O

- Given functions f(n) and g(n), we say that f(n) is O(g(n)) if and only if there exist constants $c > 0$ and $n_0 \geq 0$ such that

$$f(n) \leq c\, g(n) \text{ for all } n \geq n_0.$$

- For this to make sense, f(n) and g(n) should be functions over non-negative integers, and f(n), g(n) ≥ 0 on this range.



More formally:
"f(n) is **in** O(g(n))".

O(g(n)) is actually a *set*
(of what?)

Proving a Big-O relationship

- f(n) is O(g(n)) if there exist two positive constants c and n_0 such that $f(n) \le c\, g(n)$ for all $n \ge n_0$.

- Q: How to prove that f(n) is O(g(n))?
  A: Give c and n_0 and show the condition holds.

- Ex1: $f(n) = 4n + 15$. $g(n) = ???$
- Ex2: $f(n) = 5n^2 + 2n - 4$. $g(n) = ???$