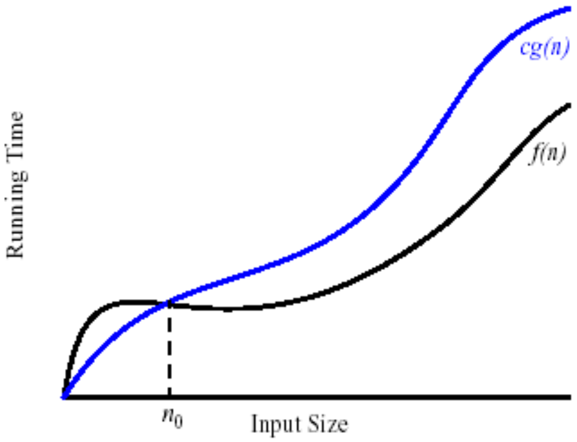


CSSE 230 Day 2

Growable Arrays Continued
Big-Oh and its cousins

Submit Growable Array exercise
Answer Q1-3 from today's in-class quiz.

Agenda and goals

- Finish course intro
- Growable Array recap
- Big-Oh and cousins

- After today, you'll be able to
  - Use the term *amortized* appropriately in analysis
  - explain the meaning of big-Oh, big-Omega (Ω), and big-Theta (θ)
  - apply the definition of big-Oh to prove runtimes of functions

Announcements and FAQ

- You will not usually need the textbook in class

- All should do piazza introduction post (a few students left)

You must demonstrate programming competence on exams to succeed

- See syllabus for exam weighting and caveats.
- Think of every program you write as a practice test
  - Especially HW4 and test 2a

Warm Up and Stretching thoughts

- Short but intense! ~45 lines of code total in our solutions to all but Adder

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

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

  Demo: Run the Adder program

Questions?

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

Homework 1 help

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? (does the inner loop depend on outer loop?)

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

Homework 1 help

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

Daring to double

Growable Arrays Table

N	E_N	**Answers for problem 2**
4	0	0
5	0	0
6	5	5
7	5	5 + 6 = 11
10	5	5 + 6 + 7 + 8 + 9 = 35
11	5 + 10 = 15	5 + 6 + 7 + 8 + 9 + 10 = 45
20	15	sum(i, i=5..19) = 180 **using Maple**
21	5 + 10 + 20 = 35	sum(i, i=5..20) = 200
40	35	sum(i, i=5..39) = 770
41	5 + 10 + 20 + 40 = 75	sum(i, i=5..40) = 810

Doubling the Size

- Doubling each time:
  - Assume that $N = 5 (2^k) + 1$.
- Total # of array elements copied:

k	N	#copies
0	6	5
1	11	5 + 10 = 15
2	21	5 + 10 + 20 = 35
3	41	5 + 10 + 20 + 40 = 75
4	81	5 + 10 + 20 + 40 + 80 = 155
k	$= 5 (2^k) + 1$	$5(1 + 2 + 4 + 8 + \ldots + 2^k)$

Express as a closed-form expression in terms of K, then express in terms of N

Doubling the Size (solution)

- Assume that $N = 5(2^k) + 1$.
- Total # of array elements copied
$= 5(1 + 2 + 4 + 8 + \ldots + 2^k)$
- **Do in terms of k, then in terms of N**

Adding One Each Time

- Total # of array elements copied:

N	#copies
6	5
7	5 + 6
8	5 + 6 + 7
9	5 + 6 + 7 + 8
10	5 + 6 + 7 + 8 + 9
N	???

Express as a closed-form expression in terms of N

Conclusions

- What's the **amortized** cost of adding an additional string…
  - in the doubling case?
  - in the add-one case?

Amortized cost means the "average per-operation cost" while adding to a single GrowableArray over time.

- So which should we use?

Q4-5

Logarithm review

Q6

Review these as needed

- Logarithms and Exponents
  - properties of logarithms:

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

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

$$\log_b x^\alpha = \alpha \log_b x$$

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

  - properties of exponentials:

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

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

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

$$b = a^{\log_a b}$$

$$b^c = a^{c*\log_a b}$$

Practice with exponentials and logs

(Do these with a friend after class, not to turn in)

Simplify: Note that log n (without a specified) base means $\log_2 n$.

Also, log n is an abbreviation for log(n).

1. **log (2 n log n)**
2. **log(n/2)**
3. **log (sqrt (n))**
4. **log (log (sqrt(n)))**

5. **$\log_4 n$**
6. **$2^{2 \log n}$**
7. **if $n=2^{3k} - 1$, solve for k.**

Where do logs come from in algorithm analysis?

Solutions

No peeking!

Simplify: Note that log n (without a specified) base means $\log_2 n$.
Also, log n is an abbreviation for log(n).

1. $1+\log n + \log \log n$
2. $\log n - 1$
3. $\frac{1}{2} \log n$
4. $-1 + \log \log n$

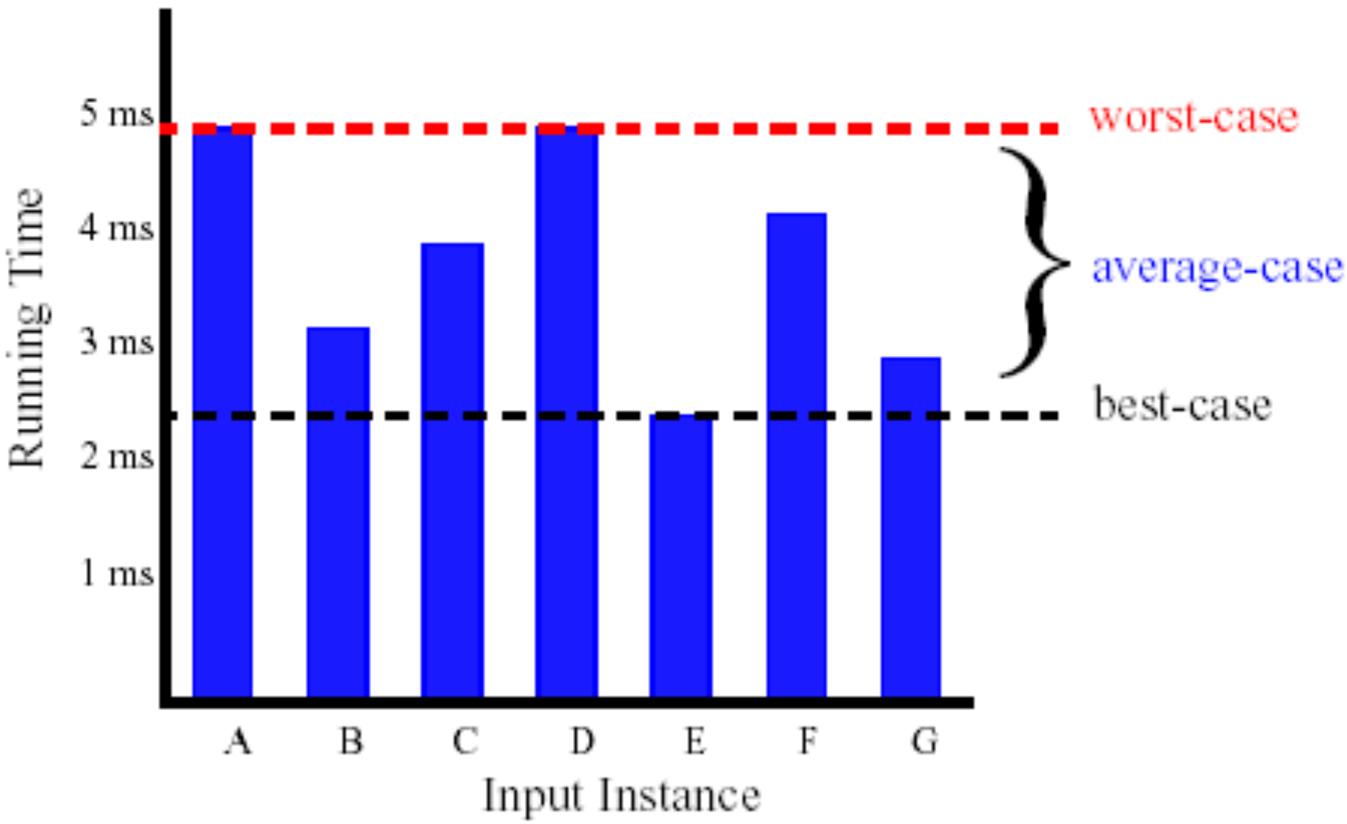
5. $(\log n) / 2$
6. n^2
7. $n+1=2^{3k}$
   $\log(n+1)=3k$
   $k= \log(n+1)/3$

A: Any time we cut things in half at each step
(like binary search or mergesort)

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

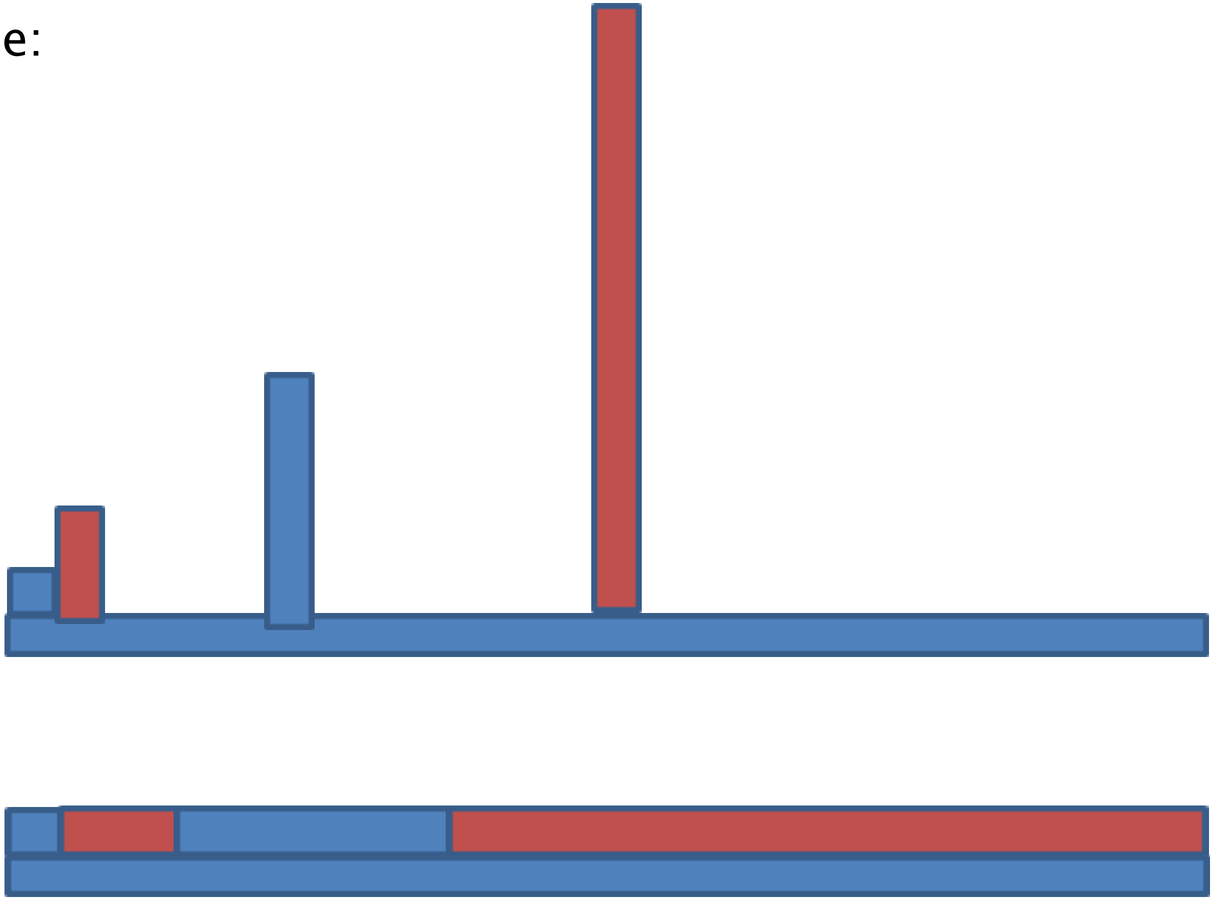
Average Case and Worst Case



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

Worst-case:
O(n)

amortized:
O(1)



Note: average case means averaged over *inputs*, amortized cost means averaged over *time*.

Asymptotics: The "Big" Three

Big-Oh
Big-Omega
Big-Theta

Asymptotic Analysis

- We only care what happens when N gets large
- Is the function linear? quadratic? exponential?

Figure 5.1

Running times for small inputs

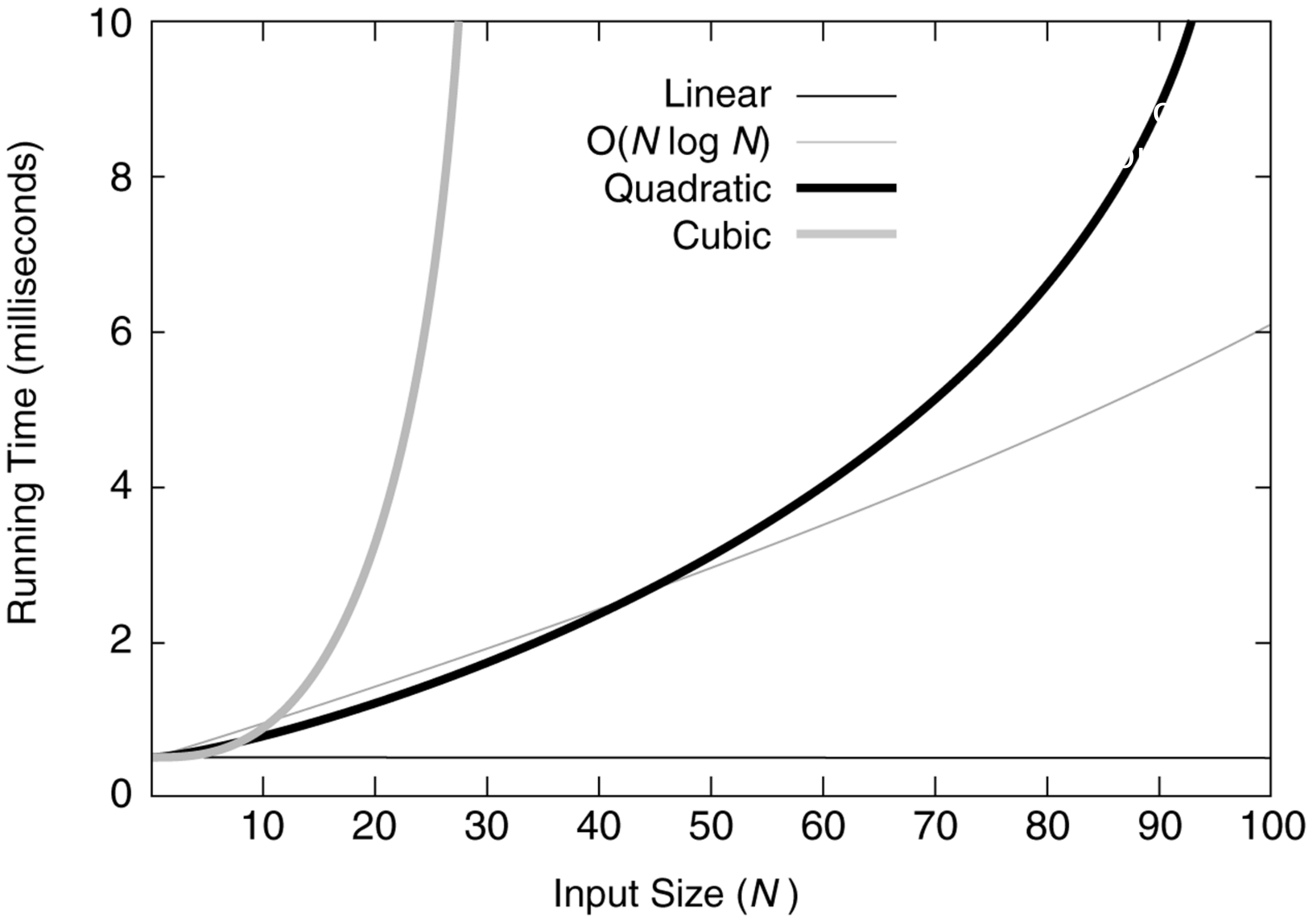


Figure 5.2

Running times for moderate inputs

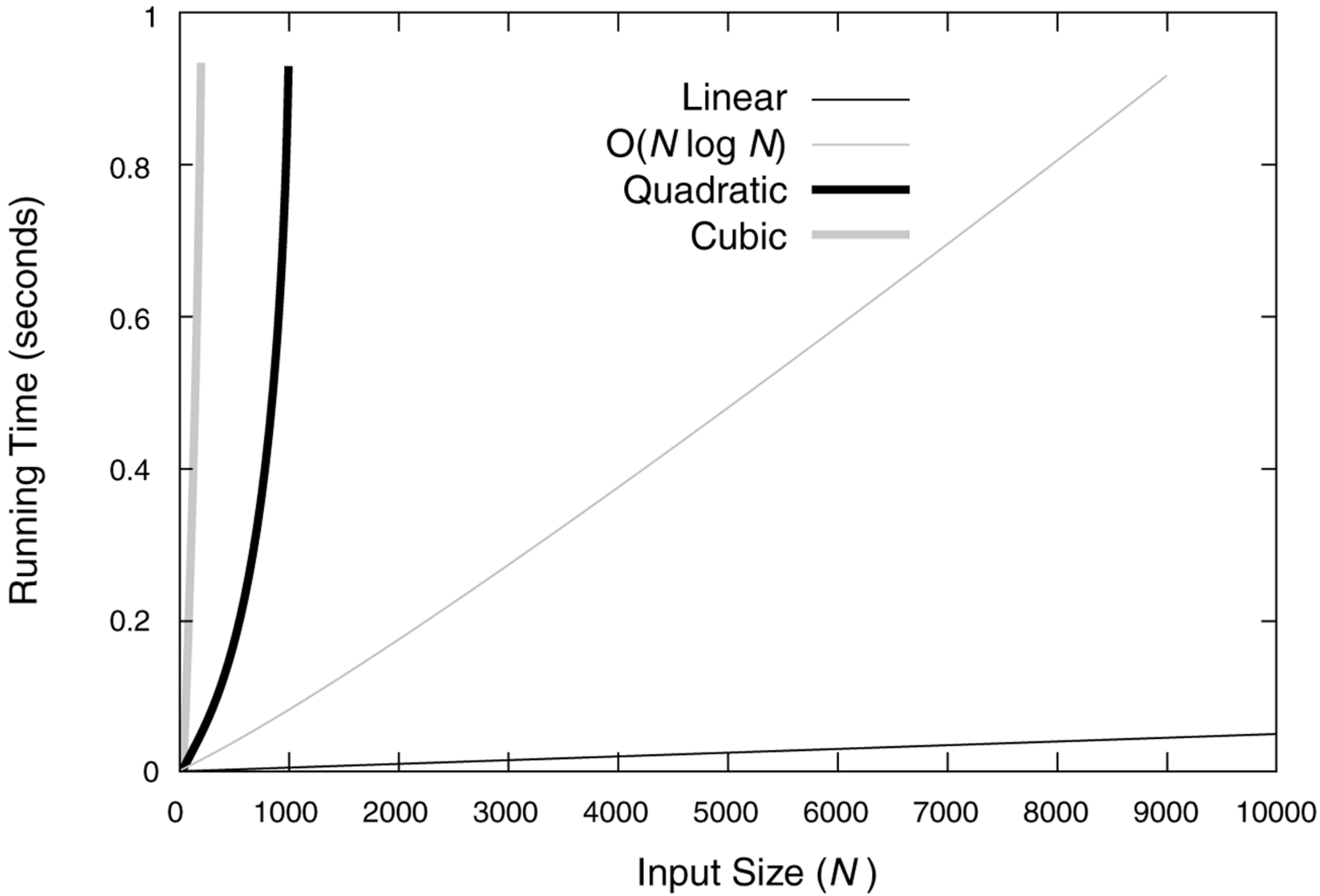


Figure 5.3

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-Oh questions is one of these functions

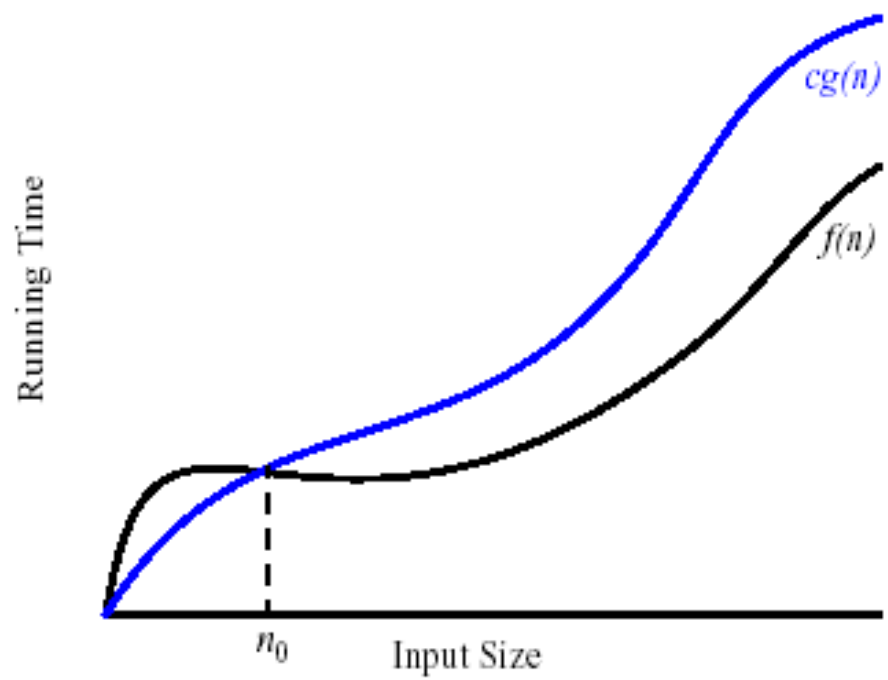
a.k.a "log linear"

Simple Rule for Big-Oh

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

Definition of Big-Oh

- Given functions f(n) and g(n), we say that f(n) is O(g(n)) if and only if $f(n) \leq c\, g(n)$ for all $n \geq n_0$.
- Two constants: $c > 0$ is a real number and $n_0 \geq 0$ is an integer.
- f(n) and g(n) are functions over non-negative integers.



To *prove* Big Oh, find 2 constants

- A function f(n) is (in) O(g(n)) if there exist two positive constants **c** and $\mathbf{n_0}$ such that *for all* $n \geq n_0$, $f(n) \leq c\, g(n)$
- **Q: How to prove that f(n) is O(g(n))?**
  A: Give c and $\mathbf{n_0}$

Assume that all functions have non-negative values, and that we only care about $n \geq 0$. For any function g(n), O(g(n)) is a set of functions.

- Ex: f(n) = 4n + 15, g(n) = ???.

Q8–9

To *prove* Big Oh, find 2 constants

- A function f(n) is (in) O(g(n)) if there exist two positive constants **c** and $\mathbf{n_0}$ such that for all $n \geq n_0$, $f(n) \leq c\, g(n)$
- **Q: How to prove that f(n) is O(g(n))?**
  A: Give c and $\mathbf{n_0}$

- Ex 2: $f(n) = n + \sin(n)$, $g(n) = ???$

Q8–9

Hidden: Answers to examples

- $f(\mathbf{n}) = n + 12$, $g(n) = ???$.
  - **$g(n) = n$. Then $c = 3$ and $n_0 = 6$, or $c = 4$ and $n_0 = 4$, etc.**
- $f(x) = x + \sin(x)$: **$g(n) = n$, $c = 2$, $n_0 = 1$**
- $f(x) = x^2 + \mathrm{sqrt}(x)$: **$g(n) = n^2$, $c=2$, $n_0 = 1$**

Big-Oh, Big-Omega and Big-Theta

O( ) Ω( ) θ( )

- f(n) is O(g(n)) if $f(n) \leq cg(n)$ for all $n \geq n_0$
  - So big-Oh (O) gives an upper bound
- f(n) is Ω(g(n)) if $f(n) \geq cg(n)$ for all $n \geq n_0$
  - So big-omega (Ω) gives a lower bound
- f(n) is θ(g(n)) if it is both O(g(n)) and Ω(g(n))
  Or equivalently:
- f(n) is θ(g(n)) if $c_1 g(n) \leq f(n) \leq c_2 g(n)$ for all $n \geq n_0$
  - So big-theta (θ) gives a tight bound
- True or false: $3n+2$ is $O(n^3)$
- True or false: $3n+2$ is $\Theta(n^3)$

Q7b,c, 10

- True or false: $3n+2$ is $O(n^3)$
- True or false: $3n+2$ is $\Theta(n^3)$

Uses of O, Ω, Θ

- By definition, applied to *functions.*

  “$f(n) = n^2/2 + n/2 - 1$ is $\Theta(n^2)$”

- Can also be applied to an *algorithm*, referencing its **running time**: e.g., when f(n) describes the number of executions of the most-executed line of code.

  “selection sort is $\Theta(n^2)$”

- Finally, can be applied to a *problem*, referencing its **complexity**: the running time of the best algorithm that solves it.

  “The sorting problem is $O(n^2)$”

Q7b,c, 10

Big–Oh Style

- Give tightest bound you can
  - Saying $3n+2$ is $O(n^3)$ is true, but not as useful as saying it's $O(n)$
  - On a test, we'll ask for Θ to be clear.
- Simplify:
  - You could also say: $3n+2$ is $O(5n-3\log(n) + 17)$
  - And it would be technically correct…
  - It would also be poor taste … and your grade will reflect that.

Efficiency in context

- There are times when one might choose a higher-order algorithm over a lower-order one.

- Brainstorm some ideas to share with the class

C.A.R. Hoare, inventor of quicksort, wrote:
Premature optimization is the root of all evil.

Q11