Lecture 4:
Running-Time Analysis



Join the **Computer Science Club**
this **Friday (Jan 31) at 12:15PM**
in the **CS Lounge** for our first social!

1. Meet and get to know new and old CSers.
2. Learn about the exciting plans CS Club has for this semester (and next!)
3. Find out open board positions and other ways to get more involved!
4. **Eat free pizza!**

See you all there!

Logistics

- HW 2 due Tuesday
- Reading for Tuesday: K&T 3.1–3.3
- Questions?

Asymptotic Growth: Big O(), Ω(), Θ()

Goal: build tools to coarsely classify running times

Examples:

- $f(n) = 4.1n^3 + 23n + n \log(n) + 27(\log(n))^{10}$
  → $f(n)$ is $O(n^3)$

- Ternary search runs in $O(\log(n))$ time

Ternary Search?

Input: sorted array A[] of size n (assume power of 3) , value x
Goal: find i such that A[j] = x

```
initialize low = 0
while (n > 1)
    i = low + ⌊n/3⌋
    j = low + ⌊2*n/3⌋
    if x < A[i]
        // low stays the same
    else if A[i] ≤ x < A[j]
        low = i
    else if  x ≥ A[j]
        low = j
    end if
    n = n/3
end while
return i if A[i] = x, else false
```

How many times through loop?

How many "primitive steps" inside loop?

Total # steps = $c \log_3(n) + d$

Ternary search runs in $O(\log(n))$ time

Logarithm Review

Defn: $\log_b(a)$ is the unique number c s.t. $b^c = a$

Informally, the number of times you can divide a into b parts before each has size one

Facts:

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

$$\log(x^k) = k\log(x)$$

$$\log_b(b^n) = n$$

$$b^{\log_b(n)} = n$$

$$\log_a(n) = \frac{\log_b(n)}{\log_b(a)}$$

O(), Ω(), Θ()

- Exercise: match definitions to pictures (in pairs)

Big O Definition

Definition: T(n) is O(f(n)) if there are constants $c \geq 0$ and $n_0 \geq 0$ such that

$$T(n) \leq c \cdot f(n) \text{ for all } n \geq n_0$$

f is an asymptotic upper bound of T

Big Ω Definition

Definition: $T(n)$ is $\Omega(f(n))$ if there are constants $c \geq 0$ and $n_0 \geq 0$ such that

$$T(n) \geq c \cdot f(n) \text{ for all } n \geq n_0$$

f is an asymptotic lower bound of T

Big Θ Definition

- T(n) is Θ(f(n)) if and only if T(n) is O(n) and T(n) is Ω(n)
- f is an asymptotic tight bound of T

O(), Ω(), Θ()

- O() - upper bound
  $T(n) \leq c \cdot f(n)$, where $c \geq 0$ for all $n \geq n_0$
- Ω() - lower bound
  $T(n) \geq c \cdot f(n)$, where $c \geq 0$ for all $n \geq n_0$
- Θ() - tight bound
  Both O() and Ω()

Properties of $O()$, $\Omega()$, $\Theta()$

Transitivity

Claim:

(a) If $f = O(g)$ and $g = O(h)$, then $f = O(h)$

(b) If $f = \Omega(g)$ and $g = \Omega(h)$, then $f = \Omega(h)$

(b) If $f = \Theta(g)$ and $g = \Theta(h)$, then $f = \Theta(h)$

Prove (a) on board

Additivity

Claim:

(a) If $f = O(h)$ and $g = O(h)$, then $f+g = O(h)$

(b) If $f_1, f_2, \ldots, f_k$ are each $O(h)$, then $f_1 + f_2 + \ldots + f_k$ is $O(h)$

(c) If $f = O(g)$, then $f+g = \Theta(g)$

Prove (a) on board; discuss (c)

Consequences of Additivity

1) OK to drop "lower-order" terms

$f(n) = 4.1n^3 + 23n + n \log(n) + 27(\log(n))^{10}$ 0

$f(n)$ is $O(n^3)$

2) Polynomials: only highest-order term matters

Let $T(n) = a_0 + a_1 n + a_2 n^2 + \ldots + a_d n^d$, where a_d is positive. Then $T(n)$ is $\Theta(n^d)$

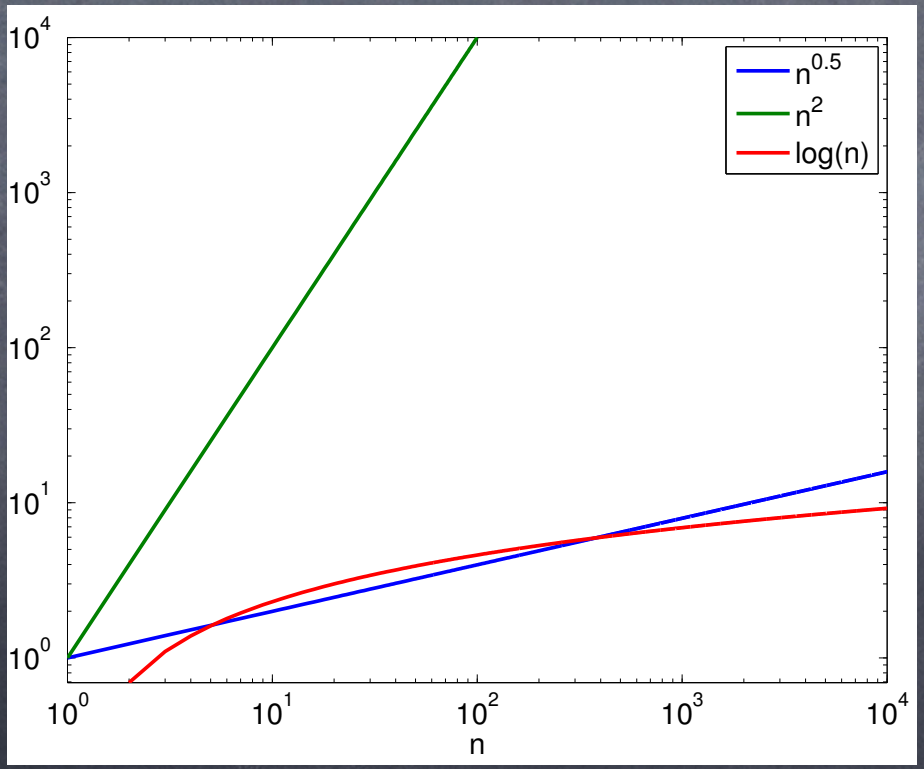
3) Definition of polynomial-time

An algorithm runs in polynomial-time if its running time $T(n)$ is $O(n^d)$ for some constant d

Asymptotic Bounds For Common Functions (i.e., sorting running-times)

Log vs Poly

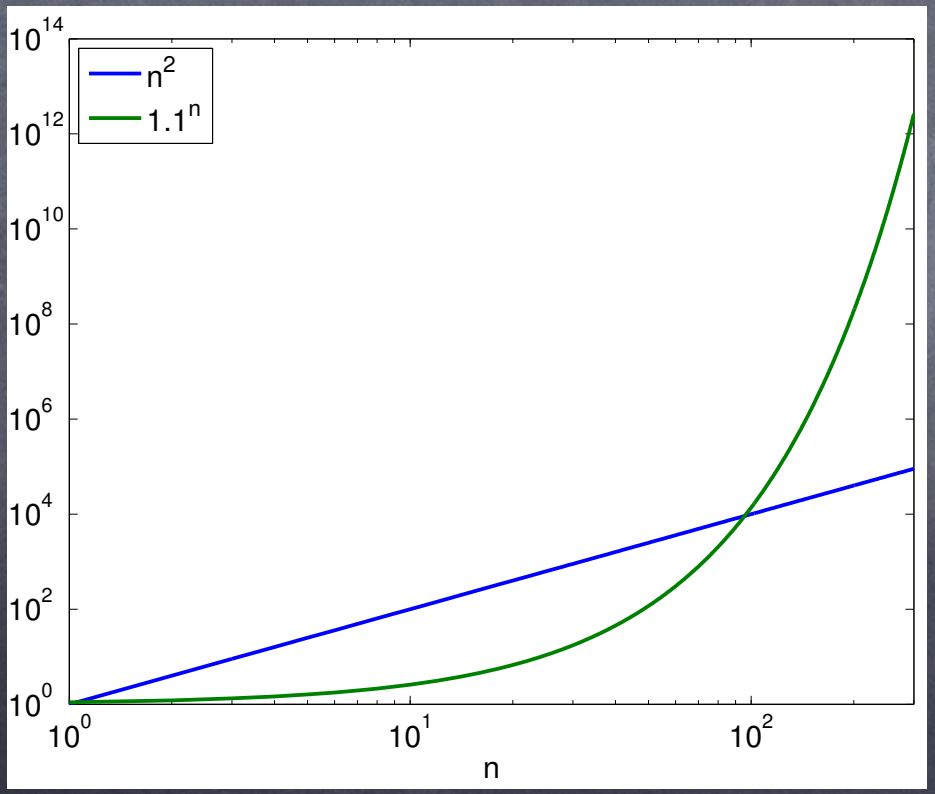
- Fact: $\log_a n$ is $O(n^d)$, for all a and d
  → Polynomials grow faster than logarithms



Poly vs Exp

Fact: n^d is $O(r^n)$ when $r > 1$

➔ Exponential functions grow faster than polynomials



A Harder Example

- Which of these grows faster?
  $n^{4/3}$
  $n(\log n)^3$

Recap

What you should know

- Polynomial time = efficient
- Definitions of O(), Ω(), Θ()
- Transitivity and additivity. Basic proof techniques
- How to "sort" functions: log(n), polynomials, n*log(n), exponentials