— Formal Defn of Big-O \_\_\_\_\_

# CS 362, Lecture 1

Jared Saia University of New Mexico

- Recall the formal definition of Big-O notation:
- A function f(n) is O(g(n)) if there exist positive constants c and n<sub>0</sub> such that f(n) ≤ cg(n) for all n ≥ n<sub>0</sub>

Today's Outline \_\_\_\_\_ Example \_\_\_\_\_

- Administrative Info
- Asymptotic Analysis Review
- Recurrence Relation Review

- Let's show that f(n) = 10n + 100 is O(g(n)) where g(n) = n
- We need to give constants c and  $n_0$  such that  $f(n) \leq cg(n)$  for all  $n \geq n_0$
- $\bullet$  In other words, we need constants c and  $n_0$  such that 10 $n+100 \leq cn$  for all  $n \geq n_0$

2

\_ Relatives of big-O \_\_\_\_\_

• We can solve for appropriate constants:

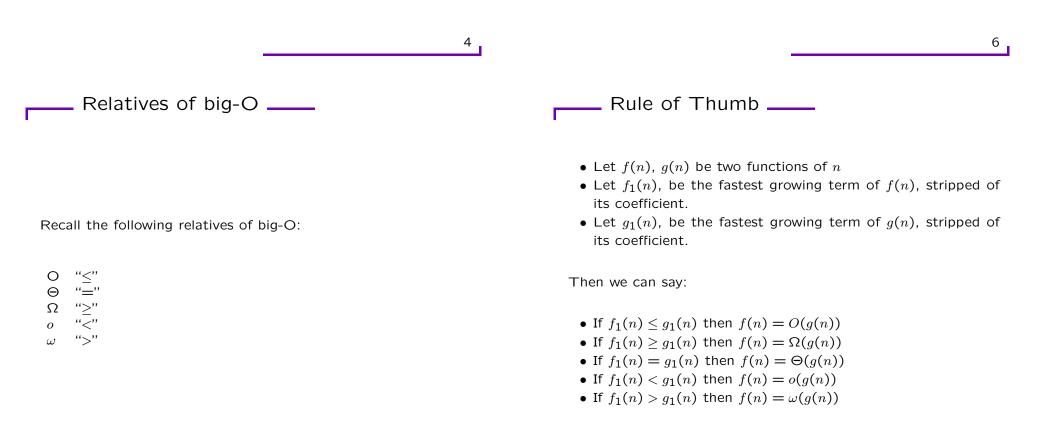
$$10n + 100 \leq cn \tag{1}$$

$$10 + 100/n \leq c$$
 (2)

- So if n > 1, then c should be greater than 110.
- In other words, for all n > 1,  $10n + 100 \le 110n$
- So 10n + 100 is O(n)

When would you use each of these? Examples:

- O " $\leq$ " This algorithm is  $O(n^2)$  (i.e. worst case is  $\Theta(n^2)$ )
- $\Theta$  "=" This algorithm is  $\Theta(n)$  (best and worst case are  $\Theta(n)$ )
- $\Omega$  " $\geq$ " Any comparison-based algorithm for sorting is  $\Omega(n \log n)$
- o "<" Can you write an algorithm for sorting that is  $o(n^2)$ ?
- $\omega$  ">" This algorithm is not linear, it can take time  $\omega(n)$

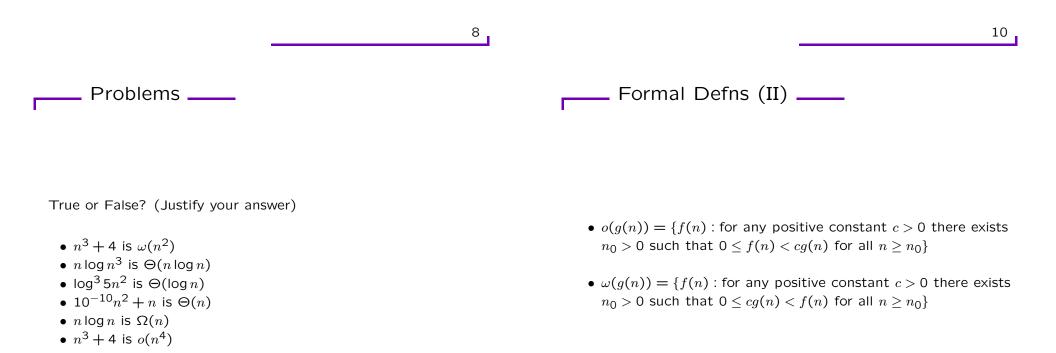


## \_\_\_ Formal Defns \_\_\_\_\_

The following are all true statements:

- $\sum_{i=1}^{n} i^2$  is  $O(n^3)$ ,  $\Omega(n^3)$  and  $\Theta(n^3)$
- $\log n$  is  $o(\sqrt{n})$
- $\log n$  is  $o(\log^2 n)$
- $10,000n^2 + 25n$  is  $\Theta(n^2)$

- $O(g(n)) = \{f(n) : \text{there exist positive constants } c \text{ and } n_0$ such that  $0 \le f(n) \le cg(n)$  for all  $n \ge n_0\}$
- $\Theta(g(n)) = \{f(n) : \text{there exist positive constants } c_1, c_2, \text{ and } n_0 \text{ such that } 0 \le c_1 g(n) \le f(n) \le c_2 g(n) \text{ for all } n \ge n_0 \}$
- $\Omega(g(n)) = \{f(n) : \text{there exist positive constants } c \text{ and } n_0$ such that  $0 \le cg(n) \le f(n)$  for all  $n \ge n_0\}$



#### Another Example \_\_\_\_\_

- Let  $f(n) = 10 \log^2 n + \log n$ ,  $g(n) = \log^2 n$ . Let's show that  $f(n) = \Theta(g(n))$ .
- We want positive constants  $c_1, c_2$  and  $n_0$ such that  $0 \le c_1 g(n) \le f(n) \le c_2 g(n)$  for all  $n \ge n_0$

$$0 \le c_1 \log^2 n \le 10 \log^2 n + \log n \le c_2 \log^2 n$$

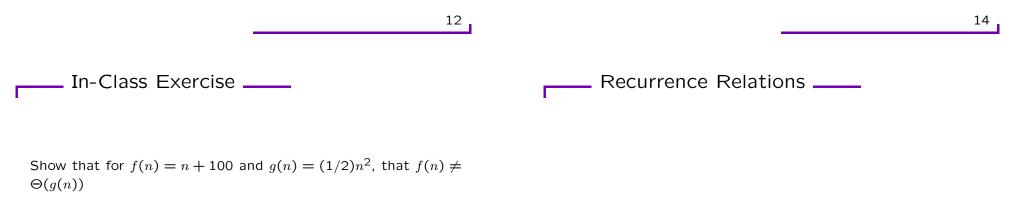
Dividing by  $\log^2 n$ , we get:

$$0 \le c_1 \le 10 + 1/\log n \le c_2$$

• If we choose  $c_1 = 1$ ,  $c_2 = 11$  and  $n_0 = 2$ , then the above inequality will hold for all  $n \ge n_0$ 

"Oh how should I not lust after eternity and after the nuptial ring of rings, the ring of recurrence" - Friedrich Nietzsche, Thus Spoke Zarathustra

- T(n) = 2 \* T(n/2) + n is an example of a *recurrence* relation
- A *Recurrence Relation* is any equation for a function *T*, where *T* appears on both the left and right sides of the equation.
- We always want to "solve" these recurrence relation by getting an equation for *T*, where *T* appears on just the left side of the equation



- What statement would be true if  $f(n) = \Theta(g(n))$  ?
- Show that this statement can not be true.

- Whenever we analyze the run time of a recursive algorithm, we will first get a recurrence relation
- To get the actual run time, we need to solve the recurrence relation

# Substitution Method \_\_\_\_\_

"guess and check"

# \_\_\_\_ Proof \_\_\_\_\_

• We prove this by induction, By I.H.:  $T(n/2) \leq cn/2\log(n/2)$ 

$$T(n) = 2T(n/2) + n$$
 (3)

$$\leq 2(cn/2\log(n/2)) + n \tag{4}$$

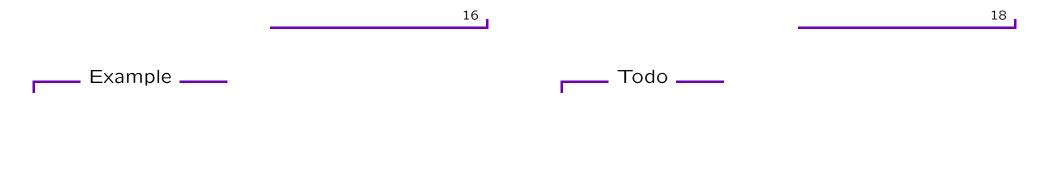
$$= cn \log(n/2) + n \tag{5}$$

$$= cn(\log n - \log 2) + n \tag{6}$$

$$= cn \log n - cn + n \tag{7}$$

$$\leq cn \log n$$
 (8)

last step holds for all n > 0 if  $c \ge 1$ 



• Let's guess that the solution to T(n) = 2 \* T(n/2) + n is  $T(n) = O(n \log n)$ 

• One way to solve recurrences is the substitution method aka

• What we do is make a good guess for the solution to T(n), and then try to prove this is the solution by induction

- In other words,  $T(n) \leq cn \log n$  for all  $n \geq n_0$ , for some positive constants  $c, n_0$
- We can prove that  $T(n) \leq cn \log n$  is true by plugging back into the recurrence

- Read Syllabus
- Visit the class web page: www.cs.unm.edu/~saia/362/
- Sign up for the class mailing list (cs362)
- Read Chapter 3 and 4 in the text