

CS221: Algorithms and  
Data Structures  
Lecture #6  
Big-O Proofs  
Analyzing Runtime

Alan J. Hu

(Borrowing slides from Steve Wolfman)

# Programming Project #1

- First “Milestone” due Monday, Jan 23, 9pm
- Milestone 2 due Monday, Jan 30, 9pm
- Final submission due Monday, Feb 6, 9pm
- Can work in two-person teams
  - No larger teams allowed.

# Today's Learning Goals

- Define big-O, big-Omega, and big-Theta:  $O(\bullet)$ ,  $\Omega(\bullet)$ ,  $\Theta(\bullet)$
- Explain intuition behind their definitions.
- Prove one function is big-O/Omega/Theta of another function.
- Simplify algebraic expressions using the rules of asymptotic analysis.
- List common asymptotic complexity orders, and how they compare.
- Work some examples.

# What was the definition of Big-O?

- What about Big-Omega and Big-Theta?

# Order Notation – Big-Theta and Big-Omega

- $T(n) \in O(f(n))$  iff there are constants  $c$  and  $n_0$  such that  $T(n) \leq c f(n)$  for all  $n \geq n_0$
- $T(n) \in \Omega(f(n))$  iff  $f(n) \in O(T(n))$
- $T(n) \in \Theta(f(n))$  iff  $T(n) \in O(f(n))$  and  $T(n) \in \Omega(f(n))$

# Proofs?

$$10,000 n^2 + 25 n \in \Theta(n^2)$$

$$10^{-10} n^2 \in \Theta(n^2)$$

$$n \log n \in O(n^2)$$

$$n \log n \in \Omega(n)$$

$$n^3 + 4 \in O(n^4) \text{ but not } \Theta(n^4)$$

$$n^3 + 4 \in \Omega(n^2) \text{ but not } \Theta(n^2)$$

How do you prove a big-O? a big- $\Omega$  ? a big- $\Theta$  ?

# Proving a Big-O

Formally, to prove  $T(n) \in O(f(n))$ , you must show:

$$\exists c, n_0 \forall n > n_0 [T(n) \leq cf(n)]$$

So, we have to come up with specific values of  $c$  and  $n_0$  that “work”, where “work” means that for any  $n > n_0$  that someone picks, the formula holds:

$$[T(n) \leq cf(n)]$$

# Proving Big-O -- Example

$$10,000 n^2 + 25 n \in \Theta(n^2)$$

$$10^{-10} n^2 \in \Theta(n^2)$$

$$n \log n \in O(n^2)$$

$$n \log n \in \Omega(n)$$

$$n^3 + 4 \in O(n^4) \text{ but not } \Theta(n^4)$$

$$n^3 + 4 \in \Omega(n^2) \text{ but not } \Theta(n^2)$$



## Prove $n \log n \in O(n^2)$

- Guess or figure out values of  $c$  and  $n_0$  that will work.

(Let's assume base-10 logarithms.)

- Turns out  $c=1$  and  $n_0 = 1$  works!
- Now, show that  $n \log n \leq n^2$ , for all  $n > 1$
- This is fairly trivial:  $\log n \leq n$  (for  $n > 1$ )

Multiply both sides by  $n$  (OK, since  $n > 1 > 0$ )

# Aside: Writing Proofs

- In lecture, my goal is to give you intuition.
  - I will just sketch the main points, but not fill in all details.
- When you *write* a proof (homework, exam, reports, papers), be sure to write it out formally!
  - Standard format makes it much easier to write!
    - Class website has links to notes with standard tricks, examples
    - Textbook has good examples of proofs, too.
    - Copy the style, structure, and format of these proofs.
  - On exams and homeworks, you'll get more credit.
  - In real life, people will believe you more.

# To Prove $n \log n \in O(n^2)$

Proof:

By the definition of big-O, we must find values of  $c$  and  $n_0$  such that for all  $n \geq n_0$ ,  $n \log n \leq cn^2$ .

Consider  $c=1$  and  $n_0 = 1$ .

For all  $n \geq 1$ ,  $\log n \leq n$ .

Therefore,  $\log n \leq cn$ , since  $c=1$ .

Multiplying both sides by  $n$  (and since  $n \geq n_0 = 1$ ), we have  $n \log n \leq cn^2$ .

Therefore,  $n \log n \in O(n^2)$ .

QED

(This is more detail than you'll use in the future, but until you learn what you can skip, fill in the details.)

# Proving Big- $\Omega$

- Just like proving Big-O, but backwards...

# Proving Big- $\Theta$

- Just prove Big-O and Big- $\Omega$

# Proving Big- $\Theta$ -- Example

$$10,000 n^2 + 25 n \in \Theta(n^2)$$

$$10^{-10} n^2 \in \Theta(n^2)$$

$$n \log n \in O(n^2)$$

$$n \log n \in \Omega(n)$$

$$n^3 + 4 \in O(n^4) \text{ but not } \Theta(n^4)$$

$$n^3 + 4 \in \Omega(n^2) \text{ but not } \Theta(n^2)$$

Prove  $10,000 n^2 + 25 n \in O(n^2)$

- What values of  $c$  and  $n_0$  work?

(Lots of answers will work...)

Prove  $10,000 n^2 + 25 n \in O(n^2)$

- What values of  $c$  and  $n_0$  work?

I'll use  $c=10025$  and  $n_0 = 1$ .

$$\begin{aligned} 10,000 n^2 + 25 n &\leq 10,000 n^2 + 25 n^2 \\ &\leq 10,025 n^2 \end{aligned}$$



Prove  $10,000 n^2 + 25 n \in \Omega(n^2)$

- What is this in terms of Big-O?

Prove  $n^2 \in O(10,000 n^2 + 25 n)$

- What values of  $c$  and  $n_0$  work?

Prove  $n^2 \in O(10,000 n^2 + 25 n)$

- What values of  $c$  and  $n_0$  work?

I'll use  $c=1$  and  $n_0 = 1$ .

$$n^2 \leq 10,000 n^2$$

$$\leq 10,000 n^2 + 25 n$$

Therefore,  $10,000 n^2 + 25 n \in \Theta(n^2)$

# Mounties Find Silicon Downs Fixed

- The fix sheet (typical growth rates in order)
  - constant:  $O(1)$
  - logarithmic:  $O(\log n)$  ( $\log_k n, \log n^2 \in O(\log n)$ )
  - poly-log:  $O(\log^k n)$  ( $k$  is a constant  $> 1$ )
  - linear:  $O(n)$
  - (log-linear):  $O(n \log n)$  (usually called “ $n \log n$ ”)
  - (superlinear):  $O(n^{1+c})$  ( $c$  is a constant,  $0 < c < 1$ )
  - quadratic:  $O(n^2)$
  - cubic:  $O(n^3)$
  - polynomial:  $O(n^k)$  ( $k$  is a constant  $> 0$ ) “tractable”
  - exponential:  $O(c^n)$  ( $c$  is a constant  $> 1$ )  
“<sup>20</sup>intractable”

# Asymptotic Analysis Hacks

- These are quick tricks to get big- $\Theta$  category.
- Eliminate low order terms
  - $4n + 5 \Rightarrow 4n$
  - $0.5 n \log n - 2n + 7 \Rightarrow 0.5 n \log n$
  - $2^n + n^3 + 3n \Rightarrow 2^n$
- Eliminate coefficients
  - $4n \Rightarrow n$
  - $0.5 n \log n \Rightarrow n \log n$
  - $n \log (n^2) = 2 n \log n \Rightarrow n \log n$

# USE those cheat sheets!

- Which is faster,  $n^3$  or  $n^3 \log n$ ?
- Which is faster,  $n^3$  or  $n^{3.01}/\log n$ ?  
(Split it up and use the “dominance” relationships.)

# Rates of Growth

- Suppose a computer executes  $10^{12}$  ops per second:

n =	10	100	1,000	10,000	$10^{12}$
n	$10^{-11}$ s	$10^{-10}$ s	$10^{-9}$ s	$10^{-8}$ s	1s
n log n	$10^{-11}$ s	$10^{-9}$ s	$10^{-8}$ s	$10^{-7}$ s	40s
$n^2$	$10^{-10}$ s	$10^{-8}$ s	$10^{-6}$ s	$10^{-4}$ s	$10^{12}$ s
$n^3$	$10^{-9}$ s	$10^{-6}$ s	$10^{-3}$ s	1s	$10^{24}$ s
$2^n$	$10^{-9}$ s	$10^{18}$ s	$10^{289}$ s		

$$10^4\text{s} = 2.8 \text{ hrs}$$

$$10^{18}\text{s} = 30 \text{ billion years}$$

# Types of analysis

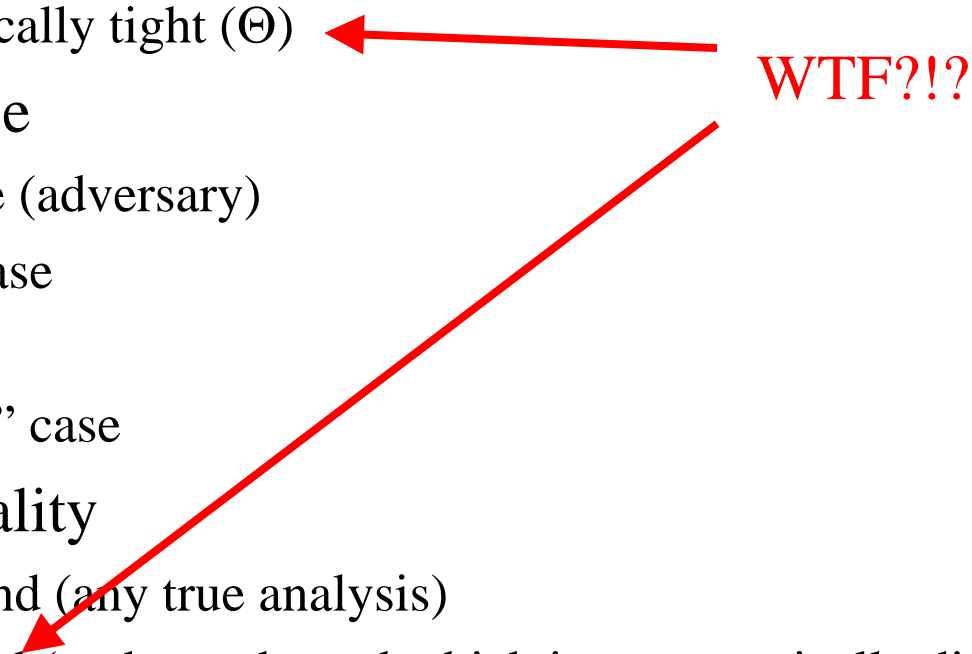
## Orthogonal axes

- bound flavor
  - upper bound ( $O$ )
  - lower bound ( $\Omega$ )
  - asymptotically tight ( $\Theta$ )
- analysis case
  - worst case (adversary)
  - average case
  - best case
  - “common” case
- analysis quality
  - loose bound (any true analysis)
  - tight bound (no better bound which is asymptotically different)<sup>24</sup>



# Types of analysis

## Orthogonal axes

- bound flavor
    - upper bound ( $O$ )
    - lower bound ( $\Omega$ )
    - asymptotically tight ( $\Theta$ )
  - analysis case
    - worst case (adversary)
    - average case
    - best case
    - “common” case
  - analysis quality
    - loose bound (any true analysis)
    - tight bound (no better bound which is asymptotically different)<sup>25</sup>
- 
- The diagram consists of two red arrows and a red text label. One arrow points from the text 'WTF?!?' to the item 'asymptotically tight ( $\Theta$ )' in the 'bound flavor' list. The other arrow points from the same text to the item 'tight bound (no better bound which is asymptotically different)<sup>25</sup>' in the 'analysis quality' list.

# “Tight” Bound

There are at least three common usages for calling a bound “tight”:

1. Big-Theta, “asymptotically tight”
2. “no better bound which is asymptotically different”
3. Big-O upper bound on run time of an algorithm matches provable worst-case lower-bound on any solution algorithm.

# “Tight” Bound – Def. 1

## 1. Big-Theta, “asymptotically tight”

This definition is formal and clear:

$T(n) \in \Theta(f(n))$  if  $T(n) \in O(f(n))$  and  $T(n) \in \Omega(f(n))$

but it is too rigid to capture practical intuition.

For example, what if  $T(n) = (n \% 2 == 0) ? n * n : 0$

Is  $T(n) \in O(n^2)$  ?

Is  $T(n) \in \Theta(n^2)$  ?

## “Tight” Bound – Def. 2

2. “no better bound which is asymptotically different”

This is the most common definition, and captures what people usually want to say, but it’s not formal.

E.g., given same  $T(n)$ , we want  $T(n) \in O(n^2)$  to be considered “tight”, but not  $T(n) \in O(n^3)$

But,  $T(n)$  is NOT  $\Theta(n^2)$ , so isn’t  $T(n) \in O(T(n))$  a tighter bound?

## “Tight” Bound – Def. 2

2. “no better **reasonable** bound which is asymptotically different”

This is the most common definition, and captures what people usually want to say, but it’s not formal.

E.g., given same  $T(n)$ , we want  $T(n) \in O(n^2)$  to be considered “tight”, but not  $T(n) \in O(n^3)$

But,  $T(n)$  is NOT  $\Theta(n^2)$ , so isn’t  $T(n) \in O(T(n))$  a tighter bound?

## “Tight” Bound – Def. 2

2. “no better **reasonable** bound which is asymptotically different”

This is the most common definition, and captures what people usually want to say, but it’s not formal.

“Reasonable” is defined subjectively, but it basically means a simple combination of normal, common functions, i.e., anything on our list of common asymptotic complexity categories (e.g.,  $\log n$ ,  $n$ ,  $n^k$ ,  $2^n$ ,  $n!$ , etc.). There should be no lower-order terms, and no unnecessary coefficients.

## “Tight” Bound – Def. 2

2. “no better **reasonable** bound which is asymptotically different”

This is the most common definition, and captures what people usually want to say, but it’s not formal.

E.g., given same  $T(n)$ , we want  $T(n) \in O(n^2)$  to be considered “tight”, but not  $T(n) \in O(n^3)$

This is the definition we’ll use in CPSC 221 unless stated otherwise.

## “Tight” Bound – Def. 3

3. Big-O upper bound on run time of an algorithm matches provable lower-bound on **any** algorithm.

The definition used in more advanced, theoretical computer science:

- Upper bound is on a specific algorithm.
- Lower bound is on the problem in general.
- If the two match, you can't get an asymptotically better algorithm.

This is beyond this course, for the most part.

(Example: Sorting...)



## “Tight (Def. 3)” Bound for Sorting

- We'll see later today that you can sort  $n$  numbers in  $O(n \log n)$  time. Is it possible to do better?
- The answer is no (if you know nothing about the numbers and rely only on comparisons):
  - How many different ways can you arrange  $n$  numbers?
  - A sorting algorithm must distinguish between these  $n!$  choices (because any of them *might* be the input).
  - Each comparison can cut the set of possibilities in half.
  - So, to distinguish which of the  $n!$  orders you were input requires  $\lg(n!)$  comparisons.
  - $\lg(n!)$  is  $\Theta(n \log n)$

## “Tight” Bound – Def. 2

2. “no better **reasonable** bound which is asymptotically different”

This is the most common definition, and captures what people usually want to say, but it’s not formal.

E.g., given same  $T(n)$ , we want  $T(n) \in O(n^2)$  to be considered “tight”, but not  $T(n) \in O(n^3)$

This is the definition we’ll use in CPSC 221 unless stated otherwise.

# Analyzing Code

- C++ operations                      - constant time
- consecutive stmts                  - sum of times
- conditionals                        - max/sum of branches,  
   plus condition
- loops                                  - sum of iterations
- function calls                       - cost of function body

*Above all, use your head!*

# Analyzing Code

```
// Linear search  
find(key, array)  
  for i = 1 to length(array) - 1 do  
    if array[i] == key  
      return i  
  return -1
```

Step 1: What's the input size **n**?

# Analyzing Code

```
// Linear search
find(key, array)
  for i = 1 to length(array) - 1 do
    if array[i] == key
      return i
  return -1
```

Step 2: What kind of analysis should we perform?

Worst-case? Best-case? Average-case?

*Expected-case, amortized, ...*

# Analyzing Code

```
// Linear search  
find(key, array)  
  for i = 1 to length(array) - 1 do  
    if array[i] == key  
      return i  
  return -1
```

Step 3: How much does each line cost? (Are lines the right unit?)

# Analyzing Code

```
// Linear search
find(key, array)
  for i = 1 to length(array) - 1 do
    if array[i] == key
      return i
  return -1
```

Step 4: What's **T(n)** in its raw form?

# Analyzing Code

```
// Linear search
find(key, array)
    for i = 1 to length(array) - 1 do
        if array[i] == key
            return i
    return -1
```

Step 5: Simplify  $T(n)$  and convert to order notation.  
(Also, which order notation:  $O$ ,  $\Theta$ ,  $\Omega$ ?)



# Analyzing Code

```
// Linear search
find(key, array)
  for i = 1 to length(array) - 1 do
    if array[i] == key
      return i
  return -1
```

Step 6: Casually name-drop the appropriate terms in order to sound bracingly cool to colleagues: “Oh, linear search? That’s tractable, polynomial time. What polynomial? Linear, duh. See the name?! I hear it’s sub-linear on quantum computers, though. Wild, eh?”

# Analyzing Code

```
// Linear search
find(key, array)
  for i = 1 to length(array) - 1 do
    if array[i] == key
      return i
  return -1
```

Step 7: **Prove** the asymptotic bound by finding constants **c**  
and **n<sub>0</sub>** such that for all **n** ≥ **n<sub>0</sub>**, **T(n) ≤ cn**.

*You usually won't do this in practice.*<sup>42</sup>

# More Examples Than You Can Shake a Stick At (#0)

```
// Linear search
find(key, array)
  for i = 1 to length(array) - 1 do
    if array[i] == key
      return i
  return -1
```

Here's a whack-load of examples for us to:

1. find a function  $\mathbf{T}(\mathbf{n})$  describing its runtime
2. find  $\mathbf{T}(\mathbf{n})$ 's asymptotic complexity
3. find  $\mathbf{c}$  and  $\mathbf{n}_0$  to prove the complexity

# METYCSEA (#1)

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

Time complexity:

- a.  $O(n)$
- b.  $O(n \lg n)$
- c.  $O(n^2)$
- d.  $O(n^2 \lg n)$
- e. None of these

# METYC SSA (#2)

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

Time complexity:

- a.  $O(n)$
- b.  $O(n \lg n)$
- c.  $O(n^2)$
- d.  $O(n^2 \lg n)$
- e. None of these

# METYCSEA (#3)

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

Time complexity:

- a.  $O(n)$
- b.  $O(n \lg n)$
- c.  $O(n^2)$
- d.  $O(n^2 \lg n)$
- e. None of these

# METYC SSA (#4)

- Conditional

if  $C$  then  $S_1$  else  $S_2$

- Loops

while  $C$  do  $S$

# METYC SSA (#5)

- Recursion almost always yields a *recurrence*
- Recursive max:

```
if length == 1: return arr[0]
else: return larger of arr[0] and max(arr[1..length-1])
```

$$T(1) \leq b$$

$$T(n) \leq c + T(n - 1) \quad \text{if } n > 1$$

- Analysis

$$T(n) \leq c + c + T(n - 2) \quad (\text{by substitution})$$

$$T(n) \leq c + c + c + T(n - 3) \quad (\text{by substitution, again})$$

$$T(n) \leq kc + T(n - k) \quad (\text{extrapolating } 0 < k \leq n)$$

$$T(n) \leq (n - 1)c + T(1) = (n - 1)c + b \quad (\text{for } k = n - 1)$$

- $T(n) \in$



# METYC SSA (#6): Mergesort

- Mergesort algorithm
  - split list in half, sort first half, sort second half, merge together

- $T(1) \leq b$

$$T(n) \leq 2T(n/2) + cn \quad \text{if } n > 1$$

- Analysis

$$T(n) \leq 2T(n/2) + cn$$

$$\leq 2(2T(n/4) + c(n/2)) + cn$$

$$= 4T(n/4) + cn + cn$$

$$\leq 4(2T(n/8) + c(n/4)) + cn + cn$$

$$= 8T(n/8) + cn + cn + cn$$

$$\leq 2^k T(n/2^k) + kcn \quad (\text{extrapolating } 1 < k \leq n)$$

$$\leq nT(1) + cn \lg n \quad (\text{for } 2^k = n \text{ or } k = \lg n)$$

- $T(n) \in$

# METYCSEA (#7): Fibonacci

- Recursive Fibonacci:

```
int Fib(n)
    if (n == 0 or n == 1) return 1
    else return Fib(n - 1) + Fib(n - 2)
```

- *Lower* bound analysis

- $T(0), T(1) \geq b$

$$T(n) \geq T(n-1) + T(n-2) + c \quad \text{if } n > 1$$

- Analysis

let  $\phi$  be  $(1 + \sqrt{5})/2$  which satisfies  $\phi^2 = \phi + 1$

show by induction on  $n$  that  $T(n) \geq b\phi^{n-1}$

## Example #7 continued

- Basis:  $T(0) \geq b > b\phi^{-1}$  and  $T(1) \geq b = b\phi^0$
- Inductive step: Assume  $T(m) \geq b\phi^m - 1$  for all  $m < n$   
$$\begin{aligned} T(n) &\geq T(n-1) + T(n-2) + c \\ &\geq b\phi^{n-2} + b\phi^{n-3} + c \\ &\geq b\phi^{n-3}(\phi + 1) + c \\ &= b\phi^{n-3}\phi^2 + c \\ &\geq b\phi^{n-1} \end{aligned}$$
- $T(n) \in$
- Why? Same recursive call is made numerous times.

# Final Concrete Example (#8): Longest Common Subsequence

- Problem: given two strings ( $m$  and  $n$ ), find the longest sequence of characters which appears in order in both strings
  - lots of applications, DNA sequencing, blah, blah, blah
- Example:
  - “**s**earch **m**e” and “**i**nsane **m**ethod” = “**s**ame”

# Abstract Example (#9): It's Log!

Problem: find a tight bound on  $T(n) = \lg(n!)$

Time complexity:

- a.  $O(n)$
- b.  $O(n \lg n)$
- c.  $O(n^2)$
- d.  $O(n^2 \lg n)$
- e. None of these