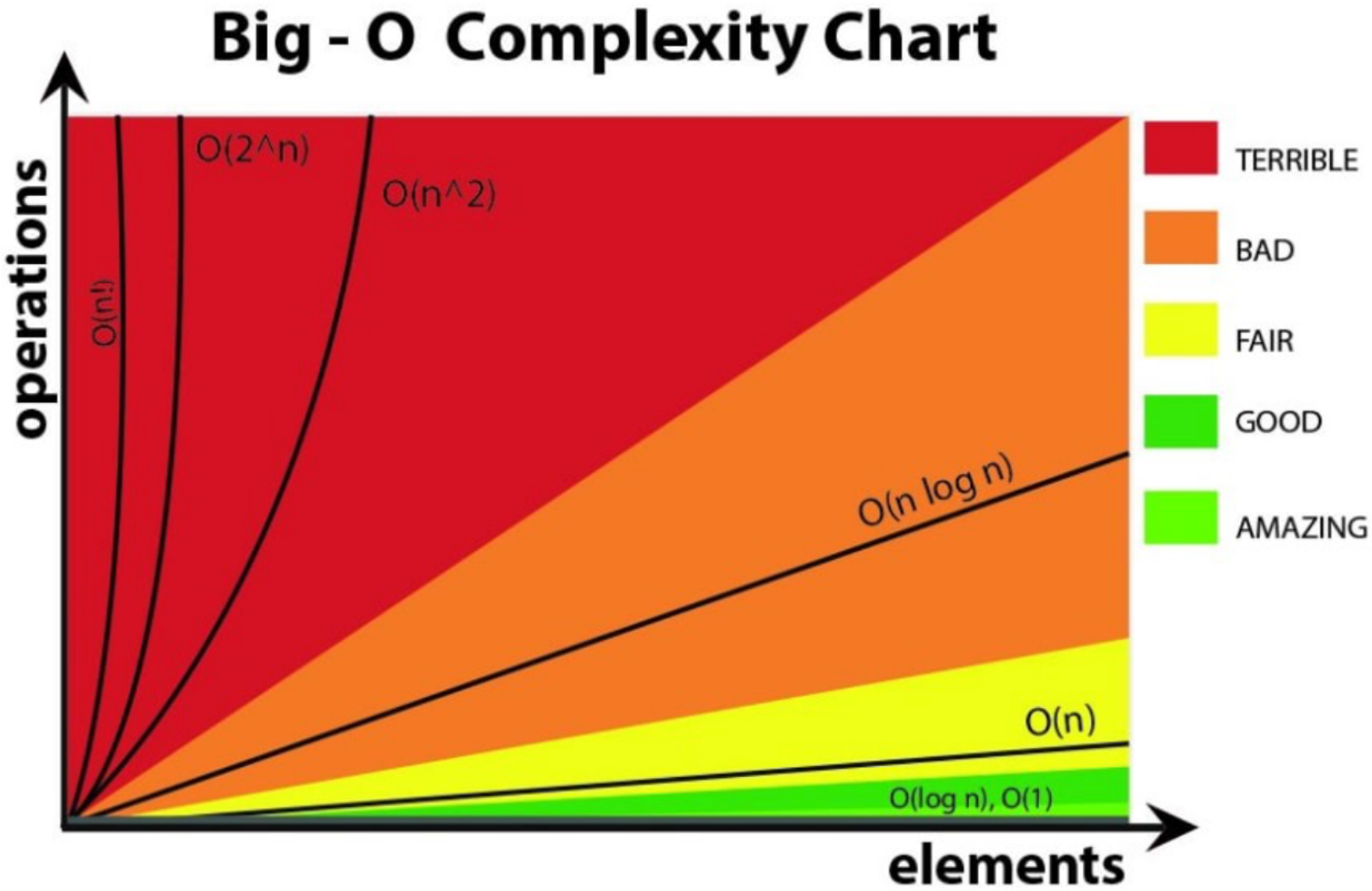
CSCI 3104: Algorithms

Lecture 3: Asymptotic Notation and Analysis

Rachel Cox

Department of Computer Science



Source

Introduction & Logistics

Course Platforms:

Canvas - All links to Zoom Lectures, HW assignments, Quizzes, Grades

Piazza - Q&A from classmates, TAs, CAs, and Instructors

Gradescope - Submission of HW and Exams

What will we learn today?

- ❑ Efficiency and Complexity
- ❑ Asymptotic Notation
- ❑ Asymptotic Analysis

- ❑ Sections 2.2, 2.3 & Chapter 3 of CLRS

Efficiency

Do we always want the fastest running algorithm?

Efficiency

Reasons to analyze algorithms:

- Predict performance
- Compare algorithms
- Provide guarantees

How do we define and measure efficiency in algorithms?

Are we going to time the algorithm?

- Actual Runtime depends on many factors
  - hardware, internet, ...

} could do this (empirical studies)

- Our goal is to standardize and make our analysis independent of a particular machine

} * complexity + asymptotics analysis

Efficiency

The operation counts that we tally will be **functions** from the input size to the number of operations.

n

❖ **Time complexity is the function**.

An algorithm is **efficient** if it has polynomial runtime. If there are constants $c > 0$ and $d > 0$ such that on every input instance of size n, running time is bounded by cn^d

≈

- Historically, lower order polynomials are efficient. $n, nlogn, n^2, n^3$

Efficiency

How will we count operations? ← operations in a particular algorithm

- We will specify a set of atomic operations and assume that each takes 1 unit of time.
- We will specify a set of atomic data structures and assume each takes 1 unit of space.

* These are simplifying assumptions.

What is an operation?

+, −, ×, ÷

Random Access Machine (RAM)

- motivation behind the name is that indexing and addressing memory takes 1 unit of time
- Accessing memory takes constant time assuming we know the memory location we need

RAM Model of Computation

- theoretical model of a computer hardware or software system

- **Atomic Operations**: simple mathematical operations (e.g. +, ×, −, =) and simple functional operations (e.g. if, call, goto, etc.) operations take 1 unit of time.
  - Not Atomic - Loop, function calls, subroutine calls, sort, reading a file
- **Atomic Data Types**: Simple data types (e.g. boolean, integer, character, float) take 1 unit of space.
  - Not Atomic - array, strings, lists, trees

- RAM model strikes a good balance of making simplifying assumption, allowing complexity to be independent of the hardware, but is still a decent model

Operations Count

$len(A) = n$

Example: Count the operations for the algorithm shown below.

Algorithm 1 A simple function for finding the maximum number in a list of integers.

	cost	times
procedure MAXIMUM(A)		
$temp = 0$	c_1	1
for i in 1..len(A) :	c_2	$n+1$
if $A[i] \geq temp$:	c_3	n
$temp = A[i]$	c_4	t_i
return $temp$	c_5	1

dependent on what A looks like

Let $T(n)$ represent the runtime of this procedure.

$T(n) = c_1 + c_2(n+1) + c_3 n + c_4 \cdot t_i + c_5$ } operations count

$t_i \leq len(A) = n$

$T(n) \leq c_1 + c_2(n+1) + c_3 n + c_4 \cdot n + c_5 = (c_1 + c_2 + c_5) + (c_2 + c_3 + c_4)\, n$

Operations Count

Recall: linear function $f(x) = ax + b$

Example: Count the operations for the algorithm shown below.

Algorithm 1 A simple function for finding the maximum number in a list of integers.

```
procedure MAXIMUM(A)
    temp = 0
    for i in 1..len(A) :
        if A[i] ≥ temp :
            temp = A[i]
    return temp
```

$\Rightarrow T(n) \leq a + bn$ $\Rightarrow T(n)$ is bounded above by a linear function.

Also $T(n) = c_1 + c_2(n+1) + c_3 n + c_4 \cdot t_i + c_5$

$\geq (c_2 + c_3) n$

$\Rightarrow T(n) \geq dn$ $\Rightarrow T(n)$ is bounded below by a linear function

Operations Count

Example: Count the operations for the algorithm shown below.

Algorithm 1 Sort an array in ascending order with insertion sort.

procedure INSERTIONSORT(A)	#	*cost*	*times*
for j in $2..\text{len}(A)$ **:**	#	c_1	n
$k = A[j]$	#	c_2	$n-1$
$i = j - 1$	#	c_3	$n-1$
while $i > 0$ and $A[i] > k$ **:**	#	c_4	$\sum_{j=2}^{n} t_j$
$A[i+1] = A[i]$	#	c_5	$\sum_{j=2}^{n} t_j - 1$
$i = i - 1$	#	c_6	$\sum_{j=2}^{n} t_j - 1$
$A[i+1] = k$	#	c_7	$n-1$

$$T(n) = c_1 n + c_2(n-1) + c_3(n-1) + c_4 \sum_{j=2}^{n} t_j + c_5 \sum_{j=2}^{n} (t_j - 1) + c_6 \sum_{j=2}^{n} (t_j - 1) + c_7 (n-1)$$

↑ operations count!

Operations Count

Example: What are the best-case and worst-case running times for InsertionSort?.

Algorithm 1 Sort an array in ascending order with insertion sort.

	#	*cost*	*times*
procedure INSERTIONSORT(A)			
for j in 2..len(A) :	#	c_1	n
$k = A[j]$	#	c_2	$n-1$
$i = j - 1$	#	c_3	$n-1$
• **while** $i > 0$ and $A[i] > k$:	#	c_4	$\sum t_j$
$A[i+1] = A[i]$	#	c_5	$\sum t_j - 1$
$i = i - 1$	#	c_6	$\sum t_j - 1$
$A[i+1] = k$	#	c_7	$n-1$

best case : array is already sorted

$\Rightarrow t_j = 1$ for every j

(check condition of while loop, don't Run body of while loop)

$A = [6, 7, 8, 9, 10]$ (J=4)

$i=3$ $A[3]=8$

$k=9$

$$T(n) = c_1 n + c_2(n-1) + c_3(n-1) + c_4 \sum_{j=2}^{n} 1 + 0 + 0 + c_7(n-1)$$

$$= c_1 n + c_2 n - c_2 + c_3 n - c_3 + c_4(n-1) + c_7 n - c_7$$

$$= (c_1 + c_2 + c_3 + c_4 + c_7) n - (c_2 + c_3 + c_4 + c_7)$$

$$= an - b$$

Best case.

$\Rightarrow T(n)$ is a linear function.

Operations Count

Example: What are the best-case and worst-case running times for InsertionSort?.

Algorithm 1 Sort an array in ascending order with insertion sort.

	#	*cost*	*times*
procedure INSERTIONSORT(A)			
for j in 2..len(A) :	#	c_1	n
$k = A[j]$	#	c_2	$n-1$
$i = j - 1$	#	c_3	$n-1$
while $i > 0$ and $A[i] > k$:	#	c_4	$\sum t_j$.
$A[i+1] = A[i]$	#	c_5	$> \sum t_{j-1}$
$i = i - 1$	#	c_6	
$A[i+1] = k$	#	c_7	$n-1$

Worst case: reverse sorted list

$\Rightarrow t_j = j$ for every j

$$\sum_{j=2}^{n} t_j = \sum_{j=2}^{n} j$$

$$= 1 - 1 + \sum_{j=2}^{n} j$$

$$= 1 + \sum_{j=2}^{n} j - 1$$

$$= \sum_{j=1}^{n} j - 1$$

$$= \frac{n(n+1)}{2} - 1$$

Re-indexing !

$$\sum_{j=2}^{n} t_j - 1 = \sum_{j=2}^{n} j - 1 = \sum_{j=1}^{n-1} j$$

$$= \frac{(n-1)n}{2}$$

$\Rightarrow$

$$T(n) = c_1 n + c_2(n-1) + c_3(n-1) + c_4\left(\frac{n(n+1)}{2} - 1\right) + c_5\left(\frac{(n-1)n}{2}\right) + c_6\left(\frac{(n-1)n}{2}\right) + c_7(n-1)$$

Operations Count

Example: What are the best-case and worst-case running times for InsertionSort?.

Algorithm 1 Sort an array in ascending order with insertion sort.

	#	*cost*	*times*
procedure INSERTIONSORT(A)	#		
for j in 2..len(A) **:**	#		
$k = A[j]$	#		
$i = j - 1$	#		
while $i > 0$ and $A[i] > k$ **:**	#		
$A[i+1] = A[i]$	#		
$i = i - 1$	#		
$A[i+1] = k$	#		

$$T(n) = c_1 n + c_2(n-1) + c_3(n-1) + c_4\left(\frac{n(n+1)}{2} - 1\right) + c_5\left(\frac{(n-1)n}{2}\right) + c_6\left(\frac{(n-1)n}{2}\right) + c_7(n-1)$$

$$= c_1 n + c_2 n - c_2 + c_3 n - c_3 + c_4 \frac{n^2}{2} + c_4 \cdot \frac{n}{2} - c_4 + c_5 \frac{n^2}{2} - c_5 \frac{n}{2} + c_6 \frac{n^2}{2} - c_6 \frac{n}{2} + c_7 n - c_7$$

$$= \left(\frac{c_4}{2} + \frac{c_5}{2} + \frac{c_6}{2}\right) n^2 + \left(c_1 + c_2 + c_3 + \frac{c_4}{2} - \frac{c_5}{2} - \frac{c_6}{2} + c_7\right) n + \left(-c_2 - c_3 - c_4 - c_7\right)$$

$$= a_1 n^2 + b_1 n + e_1$$

$\Rightarrow$ $T(n)$ has (quadratic) Runtime in the worst case.

Asymptotic Notation

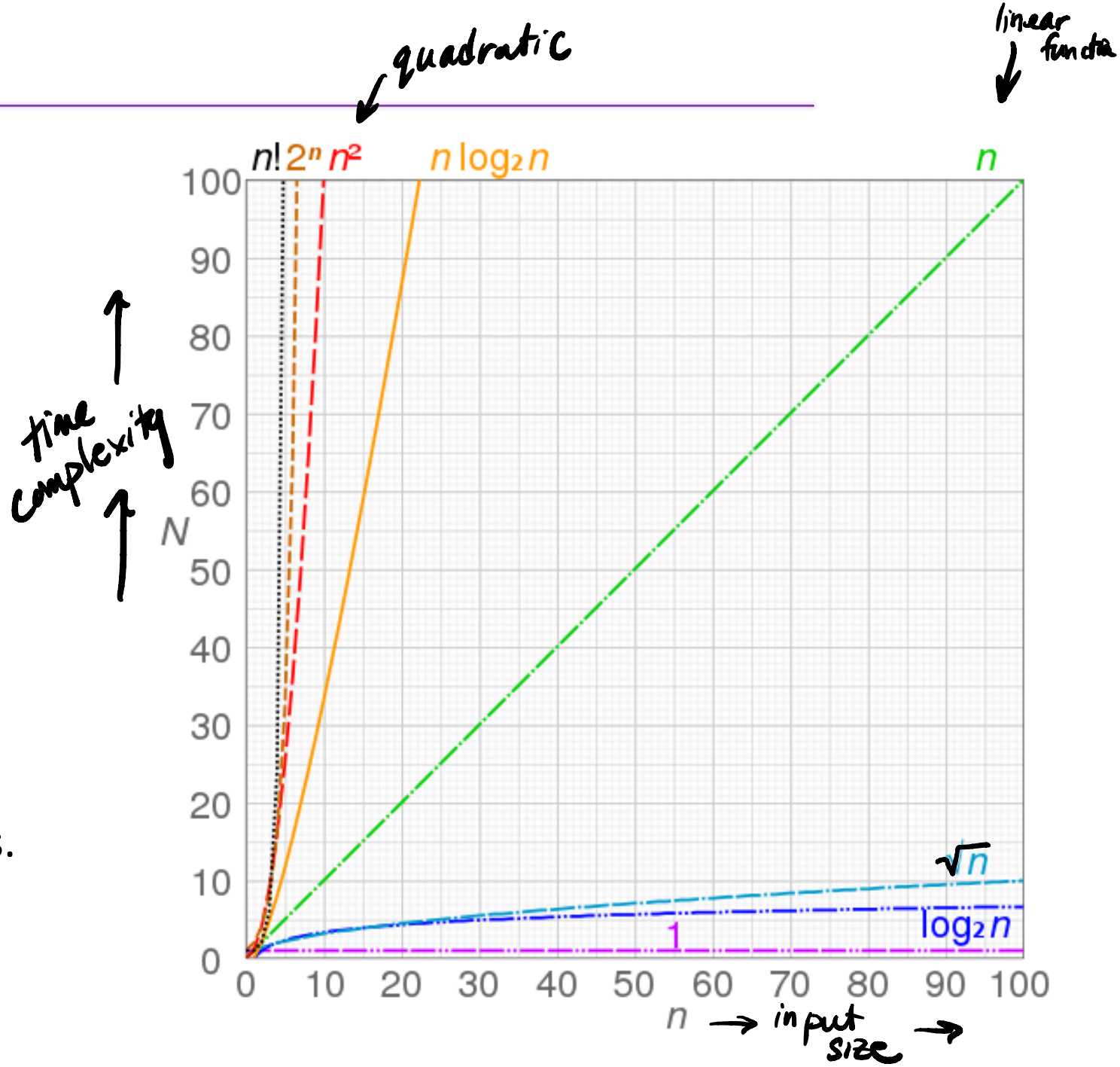
Practice versus Theory

In practice:

- Every operation counts.
- Only care about likely input.
- Can handle a few edge cases.

In theory:

- Lazy with operations.
- All inputs are important.
- Need to consider the edge cases.



Asymptotic Analysis

Asymptotics is a way to describe limiting behavior.

Example: Consider $f(n) = n^3 + 5n$ and $g(n) = n^3$. These two functions are what we would call "**asymptotically equivalent**".

- We will be using asymptotics to help us gain an understanding how an algorithm performs for an arbitrarily large input.

- We will be looking for an upper bound and a lower bound of the time complexity. We will also try to find a simple function to which our given algorithm complexity is asymptotically equal.

Asymptotic Notation

"Big O": $\mathcal{O}$ used to describe asymptotic $\leq$

asymptotic upper bound

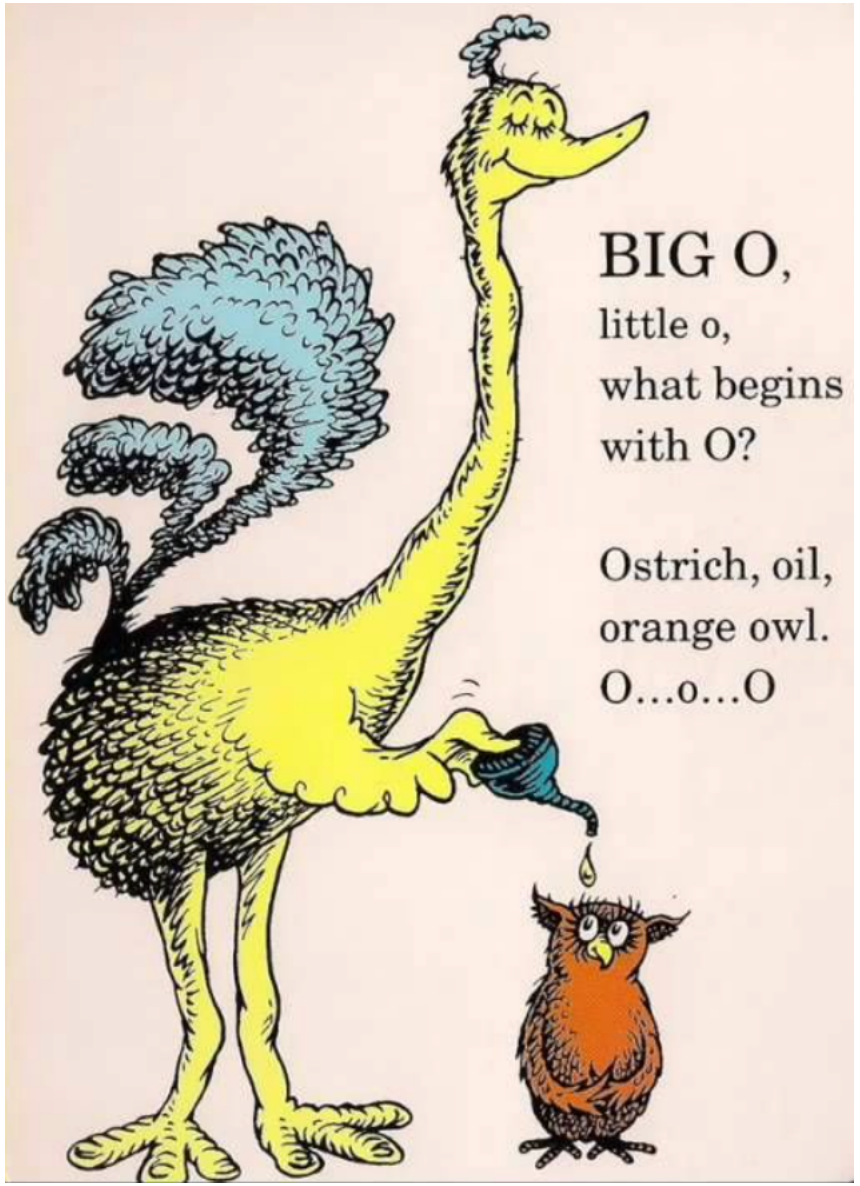
"Little o": o used to describe the asymptotic $<$

"Big Omega": Ω used to describe the asymptotic $\geq$

asymptotic lower bound

"Little omega": ω used to describe the asymptotic $>$

"Theta": Θ used to describe the asymptotic $=$



Asymptotic Notation

Asymptotic upper bound

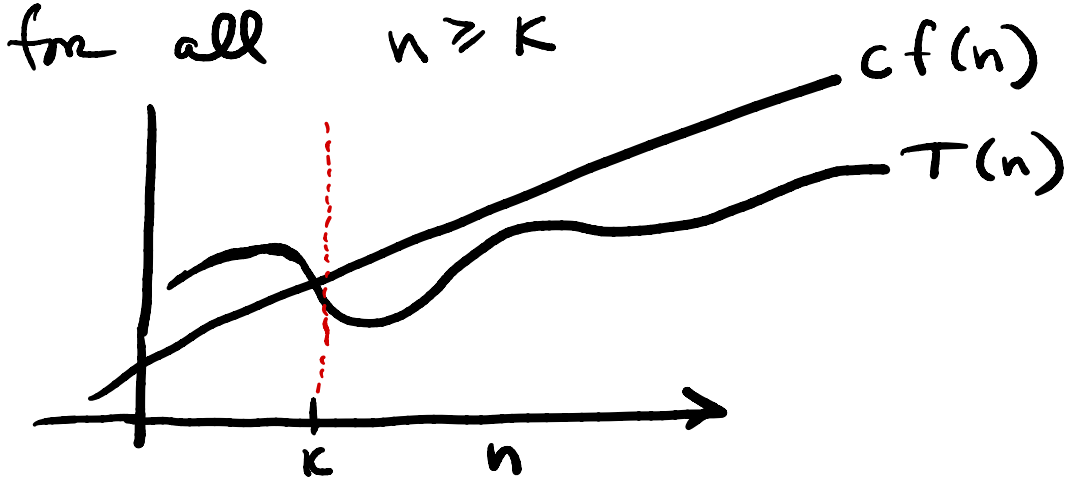
"Big O": $\mathcal{O}$ used to describe asymptotic $\leq$

- We write $f(n) = \mathcal{O}(g(n))$ to indicate that a function $f(n)$ is a member of the set $\mathcal{O}(g(n))$.

- $\mathcal{O}(g(n)) = \{f(n)$: $\exists$ positive constants C and k such that $0 \leq f(n) \leq Cg(n)$ for all $n \geq k\}$

Let T(n) be the worst case running time of an algorithm of input size n.

T(n) is O(f(n)) if $\exists c > 0$ and $k \geq 0$ such that $T(n) \leq cf(n)$ for all $n \geq k$



Common notations:

- T(n) is in O(f(n))
- T(n) is O(f(n))
- T(n) = O(f(n))

Asymptotic Notation

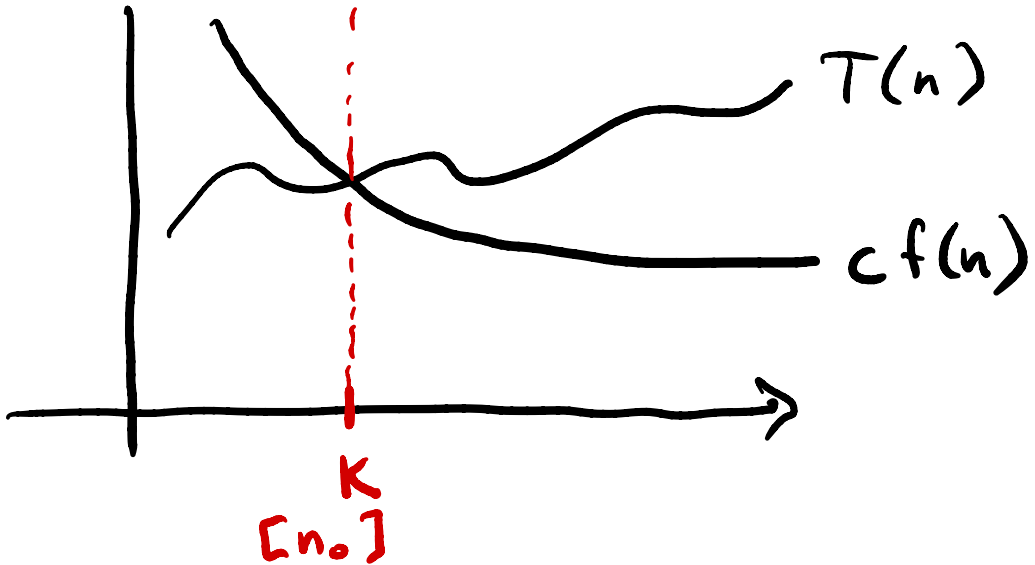
Asymptotic Lower Bound

"Big Omega": Ω used to describe the asymptotic $\geq$

We write $f(n) = \Omega(g(n))$ to indicate that a function $f(n)$ is a member of the set $\Omega(g(n))$.

$\Omega(g(n)) = \{f(n)$: $\exists$ positive constants C and k such that $0 \leq Cg(n) \leq f(n)$ for all $n \geq k\}$

$T(n)$ is $\Omega(f(n))$ if $\exists$ constants $c > 0$ and $k \geq 0$ such that $\forall n \geq k$, $T(n) \geq cf(n)$



Question: If $T(n)$ is in $\Omega(n^3)$ is $T(n)$ also in $\Omega(n)$?

Yes but its not helpful

Asymptotic Notation

two functions have the same order

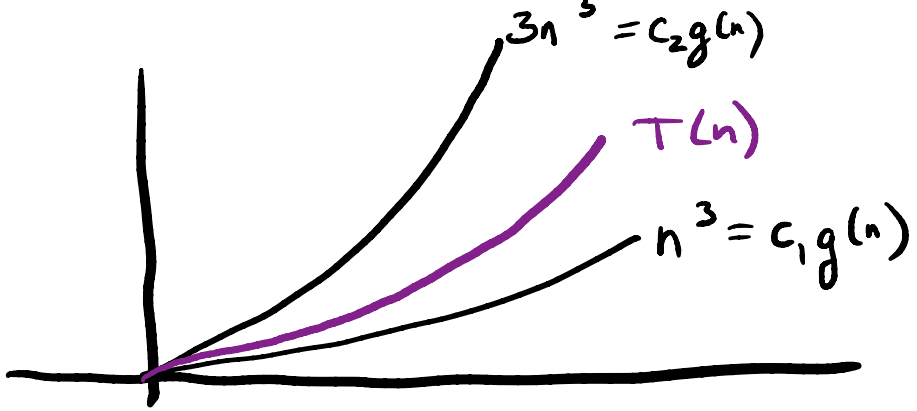
"**Theta**": Θ used to describe the asymptotic $=$

We write $f(n) = \Theta(g(n))$ to indicate that a function $f(n)$ is a member of the set $\Theta(g(n))$.

$\Theta(g(n)) = \{f(n)$: $\exists$ positive constants $C_1, C_2,$ and k such that
$0 \leq C_1 g(n) \leq f(n) \leq C_2 g(n)$ for all $n \geq k\}$

$T(n)$ is $\Theta(g(n))$ if $\exists$ constants c_1, c_2, k such that

$0 \leq c_1 g(n) \leq T(n) \leq c_2 g(n)$



Asymptotic Notation

represents an upper bound that is not asymptotically tight.

"Little o": o used to describe the asymptotic $<$

We write $f(n) = o(g(n))$ to indicate that a function $f(n)$ is a member of the set $o(g(n))$.

$o(g(n)) = \{f(n): \forall \text{ positive constants } C, \exists\ k \text{ such that } 0 \leq f(n) < Cg(n) \text{ for all } n \geq k\ \}$

e.g. $2n^2$ is $O(n^2)$ ← tight bound

$2n^2$ is $O(n^3)$ ← upper bound but not tight

$2n^2$ is $o(n^3)$

$\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0 \Rightarrow f(n)$ is $o(g(n))$ } Limit def. of little o.

Asymptotic Notation

represents a lower bound that is not asymptotically tight.

"Little omega": ω used to describe the asymptotic $>$

We write $f(n) = \omega(g(n))$ to indicate that a function $f(n)$ is a member of the set $\omega(g(n))$.

$\omega(g(n)) = \{f(n): \forall$ positive constants $C, \exists\ k$ such that $0 \leq Cg(n) < f(n)$ for all $n \geq k\ \}$

e.g. $\frac{n^2}{2} = \omega(n)$ but $\frac{n^2}{2} \neq \omega(n^2)$

$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty \Rightarrow f(n)$ is $\omega(g(n))$ } Limit def. of little omega

END of PART 1

Begin Part 2

Asymptotic Analysis

Example: Suppose that $T(n) = \frac{1}{2}n^2 - 3n$. Prove that $T(n)$ is $\Theta(n^2)$.

Asymptotic Analysis

Example: Suppose we run an algorithm to determine whether an array A contains the integer t. What is the running time?

Algorithm 1

```
1: for i = 1 to n do
2:     if A[i] == t then
3:         Return TRUE
4: Return FALSE
```

Asymptotic Analysis

Example: Suppose we run an algorithm to determine whether arrays A or B contain the integer t. What is the running time?

Algorithm 2

```
1: for i = 1 to n do
2:     if A[i] == t then
3:         Return TRUE
4: for i = 1 to n do
5:     if B[i] == t then
6:         Return TRUE
7: Return FALSE
```

Asymptotic Analysis

Example: Suppose we run an algorithm to determine whether arrays A and B have a number in common. What is the running time?

Algorithm 3

```
1: for i = 1 to n do
2:     for j = 1 to n do
3:         if A[i] == B[j] then
4:             Return TRUE
5: Return FALSE
```

Asymptotic Analysis

Example: Let $f(n) = 3n^2 + 17nlog_2(n) + 1000$. Which of the following are true?

A. $f(n)$ is $\mathcal{O}(n^2)$.

B. $f(n)$ is $\mathcal{O}(n^3)$.

C. Both A and B.

D. Neither A nor B.

Asymptotic Analysis

Example: Which is an equivalent definition of big Omega notation?

A. $f(n)$ is $\Omega(g(n))$ iff $g(n)$ is $\mathcal{O}\big(f(n)\big)$.

B. $f(n)$ is $\Omega\big(g(n)\big)$ iff there exist constants $c > 0$ such that $f(n) \geq c \cdot g(n)$ for infinitely many n.

C. Both A and B.

D. Neither A nor B.

Asymptotic Analysis

Example: Which is an equivalent definition of big Theta notation?

A. $f(n)$ is $\Theta\big(g(n)\big)$ iff $f(n)$ is both $\mathcal{O}\big(g(n)\big)$ and $\Omega\big(g(n)\big)$.

B. $f(n)$ is $\Theta\big(g(n)\big)$ iff $\lim_{n\to\infty} \frac{f(n)}{g(n)} = c$ for some constant $0 < c < \infty$.

C. Both A and B.

D. Neither A nor B.

Asymptotic Analysis

$\mathcal{O}(1)$: Functions bounded above by a constant.

$\Omega(1)$: Functions bounded below by a constant.

$o(1)$: Function that converges to 0.

$\omega(1)$: Function that converges to ∞.

Limits at Infinity – Review!

$\log^q(n) < n^p < a^n < n! < n^n$

L'hospital's Rule: $\lim_{x\to c} \frac{f(x)}{g(x)} = \lim_{x\to c} \frac{f'(x)}{g'(x)}$

- $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0 \implies f(n) < o\big(g(n)\big)$
- $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty \implies f(n) > \omega\big(g(n)\big)$
- $\lim_{n\to\infty} \frac{f(n)}{g(n)} = c \implies f(n) = \Theta\big(g(n)\big)$

Logs and Exponents– Review!

$$(x^y)^z = x^{yz}$$

$$x^y x^z = x^{y+z}$$

$$\log_x y = z \text{ iff } y = x^z$$

$$x^{\log_x y} = y$$

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

$$\log x^c = c \log x$$

$$\log \frac{x}{y} = \log x - \log y$$

$$\log_c x = \frac{\log x}{\log c}$$

Common Functions

function	description
1	constant, independent of n
$\log n$	sublinear, specifically logarithmic
n^c for $c < 1$	sublinear
n	linear
$n \log n$	super-linear, but much less than quadratic
n^2	quadratic
n^3	cubic
n^c for $c > 1$	polynomial, super-linear
c^n for $c > 1$	exponential
$n!$ or n^n	extremely fast-growing functions

Asymptotic Analysis

Example: Show that $nlogn + n \leq o(n^2)$.

Order of Growth

Example: Put the growth rates in order, from slowest growing to fastest.

$\log_4^2(n)$ $\log_5(n)$ $\log_7(n)$ $\log_{3.1}(n)$ $\log_4(n^3)$ $\sqrt[5]{n}$

Next Time

- More Asymptotic Analysis
- Begin Divide and Conquer