Lecture 5: Algorithm Analysis and Modeling

CSE 373: Data Structures and Algorithms

5 Minutes

Warm Up

Construct a mathematical function modeling the worst case runtime for the following functions

Approach

-> start with basic operations, work inside out for control structures

- Each basic operation = +1
- Conditionals = worst case test operations + branch
- Loop = iterations (loop body)

```
public void mystery1(ArrayList<String> list) {
    for (int i = 0; i < 3000; i++) {
        for (int j = 0; j < 1000; j++) {
+4          int index = (i + j) % list.size();
+2          System.out.println(list.get(index));
        }
        for (int j = 0; j < list.size(); j++) {
+1          System.out.println(":)");
        }
    }
}
```

1000(6)

n(1)

3000(1000(6) + n(1))

Possible answer
T(n) = 3000 (6000 + n)

```
public void mystery2(ArrayList<String> list) {
    for (int i = 0; i < list.size(); i++) {
        for (int j = 0; j < list.size(); j++) {
+1          System.out.println(list.get(0));
        }
    }
}
```

n(1)

n(n(1))

Possible answer
$T(n) = n^2$

Socrative:
www.socrative.com
Room Name: CSE373
Please enter your name as: Last, First

Adminstrivia

HW 1 Due Tonight at 11:59pm

HW 2 goes live Today

Please fill out class survey

Read Pair Programming Doc if you haven’t!

Why don't we care about exact constants?

Not enough information to compute precise constants

Depends on too many factors (underlying hardware, background processes, temperature etc…)

We really care about the growth of the function

Big O…

Comparing Functions

Asymptotic Analysis

asymptotic analysis – the process of mathematically representing runtime of a algorithm in relation to the number of inputs and how that relationship changes as the number of inputs grow

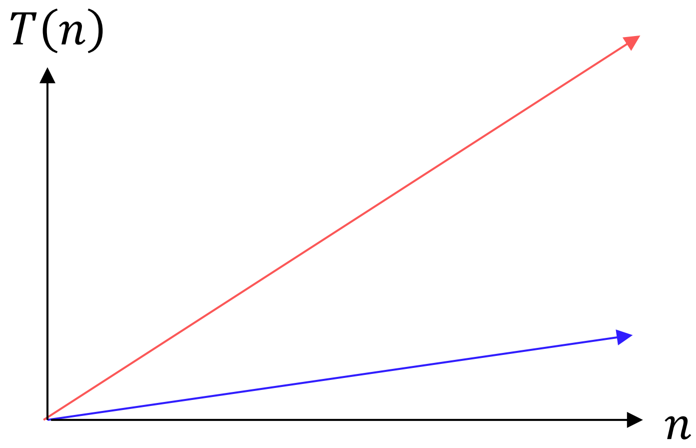
Two step process

1. **Model** – the process of mathematically representing how many operations a piece of code will run in relation to the number of inputs n
2. **Analyze** – compare runtime/input relationship across multiple algorithms
    1. Graph the model of your code where x = number of inputs and y = runtime
    2. For which inputs will one perform better than the other?

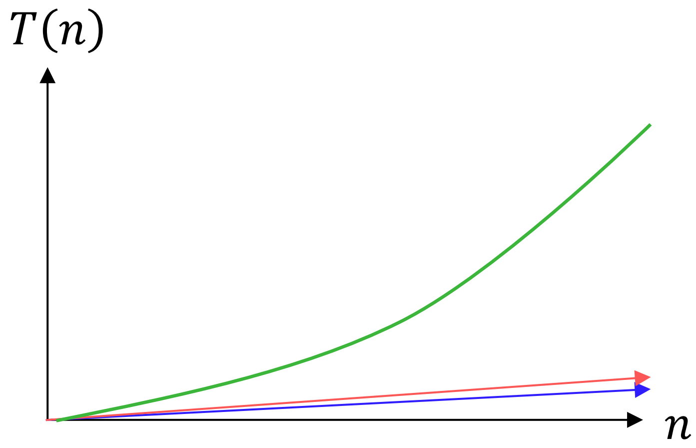
Function growth

Imagine you have three possible algorithms to choose between.
Each has already been reduced to its mathematical model

$f(n) = n$ $g(n) = 4n$ $h(n) = n^2$

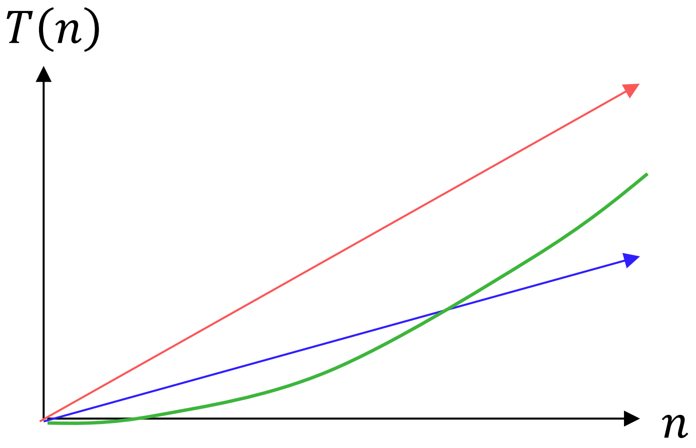


The growth rate for f(n) and g(n) looks very different for small numbers of input



...but since both are linear eventually look similar at large input sizes

whereas h(n) has a distinctly different growth rate

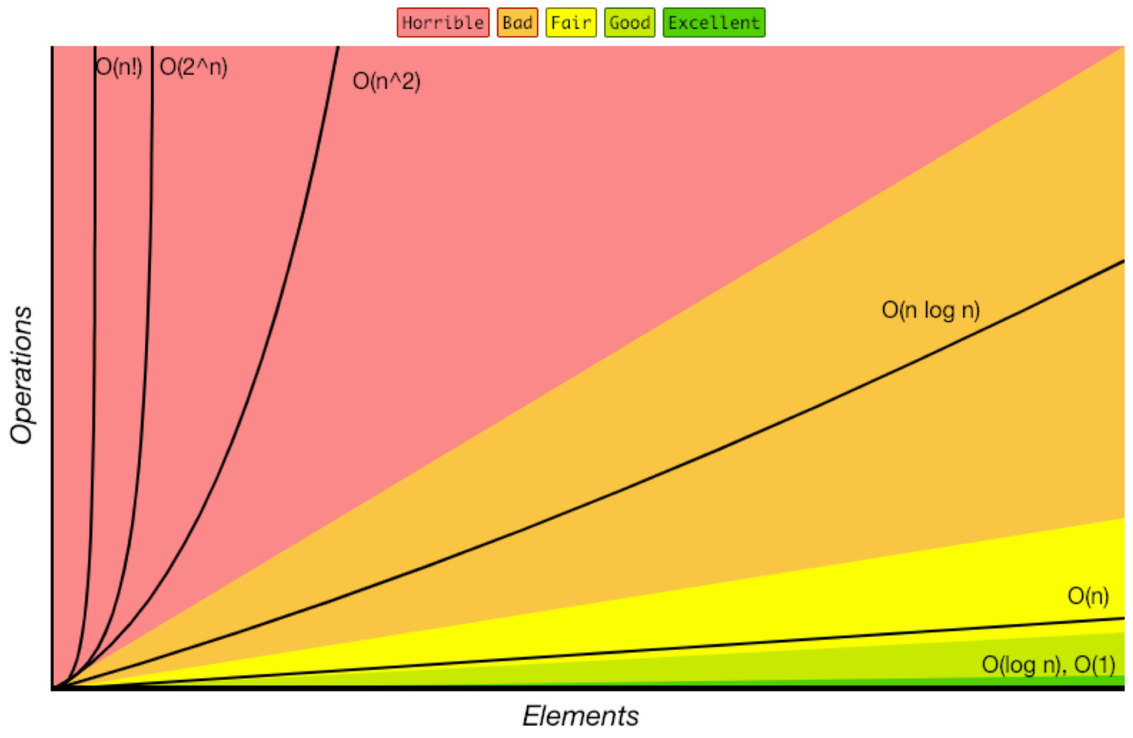


But for very small input values h(n) actually has a slower growth rate than either f(n) or g(n)

Review: Complexity Classes

complexity class – a category of algorithm efficiency based on the algorithm's relationship to the input size N

Class	Big O	If you double N...	Example algorithm
constant	O(1)	unchanged	Add to front of linked list
logarithmic	$O(\log_2 n)$	Increases slightly	Binary search
linear	O(n)	doubles	Sequential search
log-linear	$O(n\log_2 n)$	Slightly more than doubles	Merge sort
quadratic	$O(n^2)$	quadruples	Nested loops traversing a 2D array
cubic	$O(n^3)$	Multiplies by 8	Triple nested loop
polynomial	$O(n^c)$		
exponential	$O(c^n)$	Multiplies drastically	



http://bigocheatsheet.com/

Moving from Model to Complexity Class

Say an algorithm runs **$0.4N^3 + 25N^2 + 8N + 17$** statements

Consider the runtime when N is *extremely large*

17 is quickly dwarfed in the context of thousands of inputs

Lower order terms don't matter – delete them.

We ignore constants like 25 because they are tiny next to N

Multiplying by constant factors has little effect on growth rate

N^3 is so powerful it dominates the overall runtime

Highest order term dominates the overall rate of change

$O(N^3)$

Gives us a "simplified big-O"

$10n \log n + 3n$	$O(n \log n)$
$5n^2 \log n + 13n^3$	$O(n^3)$
$20n \log \log n + 2\, n \log n$	$O(n \log n)$
2^{3n}	$O(8^n)$

Definition: Big-O

We wanted to find an upper bound on our algorithm's running time, but

- We don't want to care about constant factors.
- We only care about what happens as n gets large.

Big-O

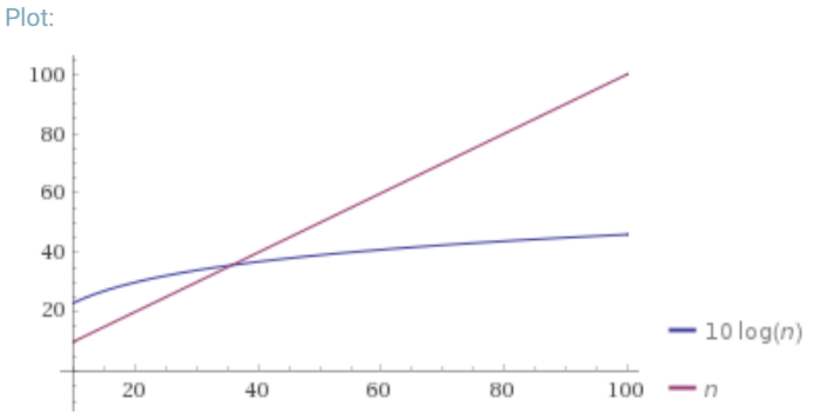
$f(n)$ is $O(g(n))$ if there exist positive constants c, n_0 such that for all $n \geq n_0$,

$$f(n) \leq c \cdot g(n)$$

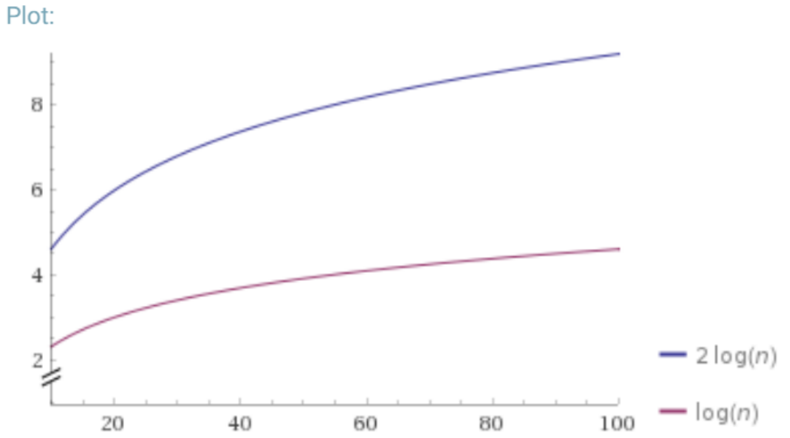
We also say that $g(n)$ "dominates" $f(n)$

O(g(n)) is the "family" or "set" of all functions that are dominated by g(n)

Why n_0?



Why c?



Applying Big O Definition

Big-O

$f(n)$ is $O(g(n))$ if there exist positive constants c, n_0 such that for all $n \geq n_0$,

$$f(n) \leq c \cdot g(n)$$

Show that $f(n) = 10n + 15$ is $O(n)$

Apply definition term by term

$$10n \leq c \cdot n \; when \; c = 10 \; for \; all \; values \; of \; n$$

$$15 \leq c \cdot n \; when \; c = 15 \; for \; n \geq 1$$

Add up all your truths

$$10n + 15 \leq 10n + 15n \leq 25n \; for \; n \geq 1$$

Select values for c and n0 and prove they validate the definition

Take $c = 25$ and $n_0 = 1$

$10n \leq 25n \; for \; all \; values \; of \; n$

$15 \leq 25n \; for \; n \geq 1$

Thus because a c and n0 exist, f(n) is O(n)

3 Minutes

Exercise: Proving Big O

Demonstrate that $5n^2 + 3n + 6$ is dominated by n^3 by finding a c and n_0 that satisfy the definition of domination

$5n^2 + 3n + 6 \le 5n^2 + 3n^2 + 6n^2$ when $n \ge 1$

$5n^2 + 3n^2 + 6n^2 = 14n^2$

$5n^2 + 3n + 6 \le 14n^2$ for $n \ge 1$

$14n^2 \le c*n^3$ for c = ? n >= ?

$\frac{14}{n}$ -> **c = 14 & n >= 1**

Big-O

$f(n)$ is $O(g(n))$ if there exist positive constants c, n_0 such that for all $n \ge n_0$,

$$f(n) \le c \cdot g(n)$$

Edge Cases

True or False: $10n^2 + 15n$ is $O(n^3)$

It's true – it fits the definition

$10n^2 \leq c \cdot n^3 \; when \; c = 10 \; for \; n \geq 1$

$15n \leq c \cdot n^3 \; when \; c = 15 \; for \; n \geq 1$

$10n^2 + 15n \leq 10n^3 + 15n^3 \leq 25n^3 \; for \; n \geq 1$

$10n^2 + 15n$ is $O(n^3)$ because $10n^2 + 15n \leq 25n^3 \; for \; n \geq 1$

Big-O is just an upper bound. It doesn't have to be a good upper bound

If we want the best upper bound, we'll ask you for a **tight** big-O bound.
$O(n^2)$ is the tight bound for this example.
It is (almost always) technically correct to say your code runs in time $O(n!)$.
DO NOT TRY TO PULL THIS TRICK ON AN EXAM. Or in an interview.

Why Are We Doing This?

You already intuitively understand what big-O means.

Who needs a formal definition anyway?

- We will.

Your intuitive definition and my intuitive definition might be different.

We’re going to be making more subtle big-O statements in this class.

- We need a mathematical definition to be sure we’re on the same page.

Once we have a mathematical definition, we can go back to intuitive thinking.

- But when a weird edge case, or subtle statement appears, we can figure out what’s correct.

3 Minutes

Function comparison: exercise

$f(n) = n \leq g(n) = 5n + 3$? **True** – all linear functions are treated as equivalent

$f(n) = 5n + 3 \leq g(n) = n$? **True**

$f(n) = 5n + 3 \leq g(n) = 1$? **False**

$f(n) = 5n + 3 \leq g(n) = n^2$? **True** – quadratic will always dominate linear

$f(n) = n^2 + 3n + 2 \leq g(n) = n^3$? **True**

$f(n) = n^3 \leq g(n) = n^2 + 3n + 2$? **False**

O, Omega, Theta [oh my?]

Big-O is an **upper bound**
- My code takes at most this long to run

Big-Omega is a **lower bound**

Big-Omega

$f(n)$ is $\Omega(g(n))$ if there exist positive constants c, n_0 such that for all $n \geq n_0$,

$$f(n) \geq c \cdot g(n)$$

Big Theta is **“equal to”**

Big-Theta

$f(n)$ is $\Theta(g(n))$ if
$f(n)$ is $O(g(n))$ and $f(n)$ is $\Omega(g(n))$.

$$\Omega(f(n)) \leq f(n) == \theta(f(n)) \leq O(f(n))$$

	Is dominated by $f(n) \in O(g(n))$	Dominates $f(n) \in \Omega(g(n))$
f(n)		
O(1)		
O(log n)		
$O(n)$		
$O(n^2)$		
$O(n^3)$		

Viewing O as a class

Sometimes you'll see big-O defined as a family or set of functions.

Big-O (alternative definition)

$O(g(n))$ is the set of all functions $f(n)$ such that there exist positive constants c, n_0 such that for all $n \geq n_0$, $f(n) \leq c \cdot g(n)$

For that reason, some people write $f(n) \in O\big(g(n)\big)$ where we wrote "$f(n)$ is $O(g(n))$".
Other people write "$f(n) = O\big(g(n)\big)$" to mean the same thing.
The set of all functions that run in linear time (i.e. $O(n)$) is a "complexity class."
We never write $O(5n)$ instead of $O(n)$ – they're the same thing!

It's like writing $\frac{6}{2}$ instead of 3. It just looks weird.

Examples

$4n^2 \in \Omega(1)$

true

$4n^2 \in \Omega(n)$

true

$4n^2 \in \Omega(n^2)$

true

$4n^2 \in \Omega(n^3)$

false

$4n^2 \in \Omega(n^4)$

false

$4n^2 \in O(1)$

false

$4n^2 \in O(n)$

false

$4n^2 \in O(n^2)$

true

$4n^2 \in O(n^3)$

true

$4n^2 \in O(n^4)$

true

Big-O

$f(n) \in O(g(n))$ if there exist positive constants c, n_0 such that for all $n \geq n_0$,

$$f(n) \leq c \cdot g(n)$$

Big-Omega

$f(n) \in \Omega(g(n))$ if there exist positive constants c, n_0 such that for all $n \geq n_0$,

$$f(n) \geq c \cdot g(n)$$

Big-Theta

$f(n) \in \Theta(g(n))$ if $f(n)$ is $O(g(n))$ and $f(n)$ is $\Omega(g(n))$.

Practice

3 Minutes

$5n + 3 \in O(n)$ True

$n \in O(5n + 3)$ True

$5n + 3 = O(n)$ True

$O(5n + 3) = O(n)$ True

$O(n^2) = O(n)$ False

$n^2 \in O(1)$ False

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

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

$n^2 \in O(n^3)$ True

$n^2 \in O(n^{100})$ True

Big-O

$f(n) \in O(g(n))$ if there exist positive constants c, n_0 such that for all $n \geq n_0$,

$$f(n) \leq c \cdot g(n)$$

Big-Omega

$f(n) \in \Omega(g(n))$ if there exist positive constants c, n_0 such that for all $n \geq n_0$,

$$f(n) \geq c \cdot g(n)$$

Big-Theta

$f(n) \in \Theta(g(n))$ if
$f(n)$ is $O(g(n))$ and $f(n)$ is $\Omega(g(n))$.