CSE 373: Data Structures and Algorithms

Asymptotic Analysis

Autumn 2018

Shrirang (Shri) Mare
shri@cs.washington.edu

Thanks to Kasey Champion, Ben Jones, Adam Blank, Michael Lee, Evan McCarty, Robbie Weber, Whitaker Brand, Zora Fung, Stuart Reges, Justin Hsia, Ruth Anderson, and many others for sample slides and materials ...

Administrivia

Due dates:

- HW2 Part 1 due on Friday 11:59pm
- If you are going to use late days, submit the HW2 late day form (link will be available on the website)

Notes

- Changes in Office Hours
  - Sherdil's OH ~~Monday 4:30 – 6:30pm~~ Friday 12:30 – 1:30 and Friday 3:30 – 4:30
  - Shri's OH ~~Wednesday 3:30 – 6:30pm~~ Wednesday 4–6pm (except today it's 4:30–6:30) and Friday 10:30 – 12:30
  - Changes updated on website. Course calendar is the best place to get office hour info

- Interest in an extra lecture on Git and Eclipse
  - Happening Thursday (10/11) 5:30-6:30 in KNE 210

Review: Modeling functions

Construct a *mathematical function* modeling the worst-case runtime

```
public boolean foo(int n) {
    int[] array = new int[n];
    for (int i = 0; i < array.length - 1; i++) {
            if (i != j && array[i] == array[i + 1]) {
                array[i] = 0;
            } else {
                System.out.println(array[i]);
                array[i] = '?';
            }
    }
    return false;
}
```

$T(n) = 3(n-1) + 2$
$T(n) = 8(n-1) + 2$
$T(n) = 10(n-1) + 2$
$T(n) = 10(n-1) + 5$

All of these answers are okay (will get full credit)

Review: Modeling functions

Construct a *mathematical function* modeling the worst-case runtime

```
public void foobar(int k) {
    int j = 0;
    while (j < k) {
        for (int i = 0; i < k; ++i) {
            System.out.println("Hello word!!");
        }
        j = j + 5;
    }
}
```

k/5

(k+4)/5

Both answers are okay!

Why we don’t care about exact constants?

1. Not enough information to compute the precise constants
   - Depends on lot of factors (e.g., underlying hardware, background processes, temperature, etc)

2. We really care about the growth of our function.
   - If you know certain code runs in X time, can you say something about what it’s run time would be when it’s input increases by 2 times, 3 times, …
   - In other words, we really care about big-O

Asymptotic analysis: Two step process

1. **Model** what we care about as a mathematical function
2. **Analyze** that function using asymptotic analysis

We want to formalize a way to compare two functions.
Specifically, compare growth and for large input.

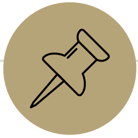
We want to derive an upper bound, i.e., we want to be able to say something like

$f(n)$ is "less then or equal to" $g(n)$

In other words: $f(n)$ is "dominated by" $g(n)$

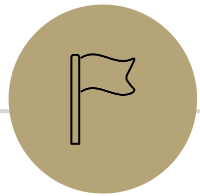
Our current intuition

Function	O(?)
$n + 2$	
$4n + 3$	
$2n^2 + n + 100$	
n^2	
$100n^3 + n^2 + 20$	



Our current intuition

Function	O(?)
$n + 2$	$O(n)$
$4n + 3$	$O(n)$
$2n^2 + n + 100$	$O(n^2)$
n^2	$O(n^2)$
$100n^3 + n^2 + 20$	$O(n^3)$



Formally defining Big-O

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.

Formally 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.

The formal, mathematical definition is Big-O.

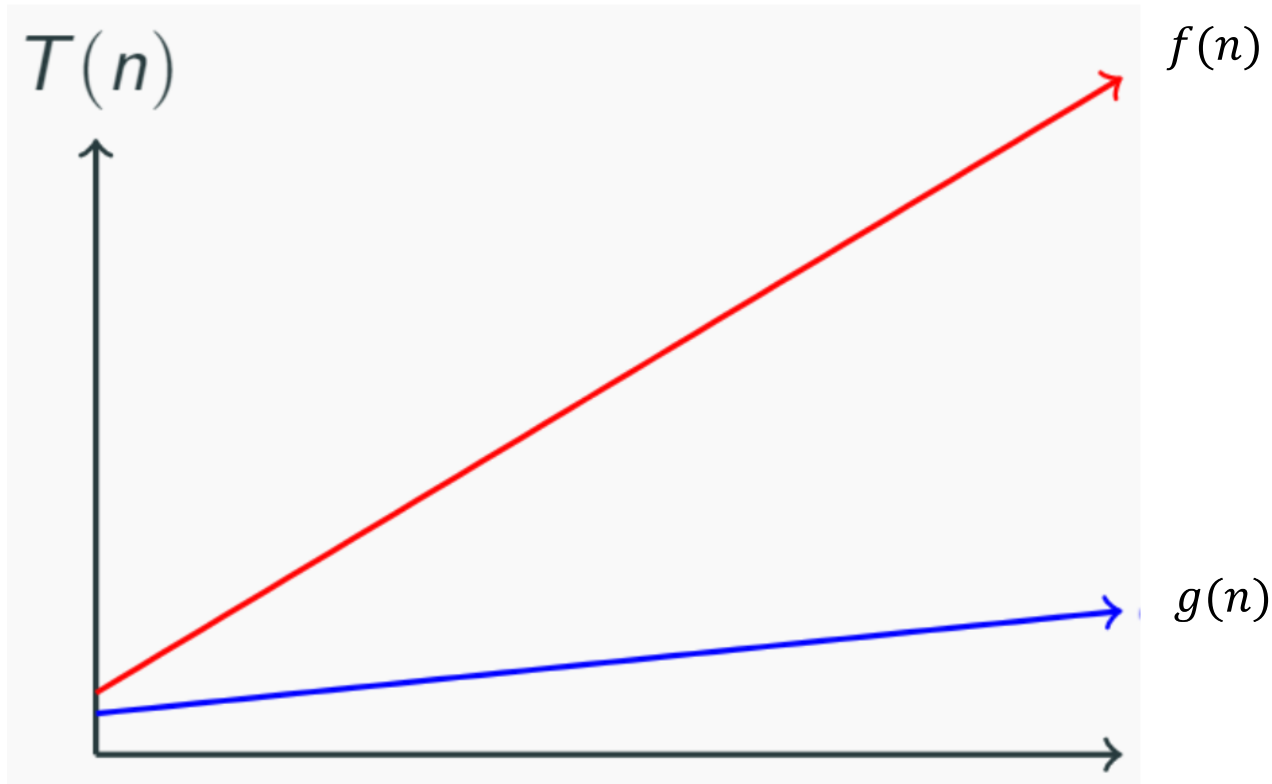
Big-O

$f(n)$ 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 $f(n)$ is "dominated by" $g(n)$

Analysis: Comparing function growth



Big-O

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

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

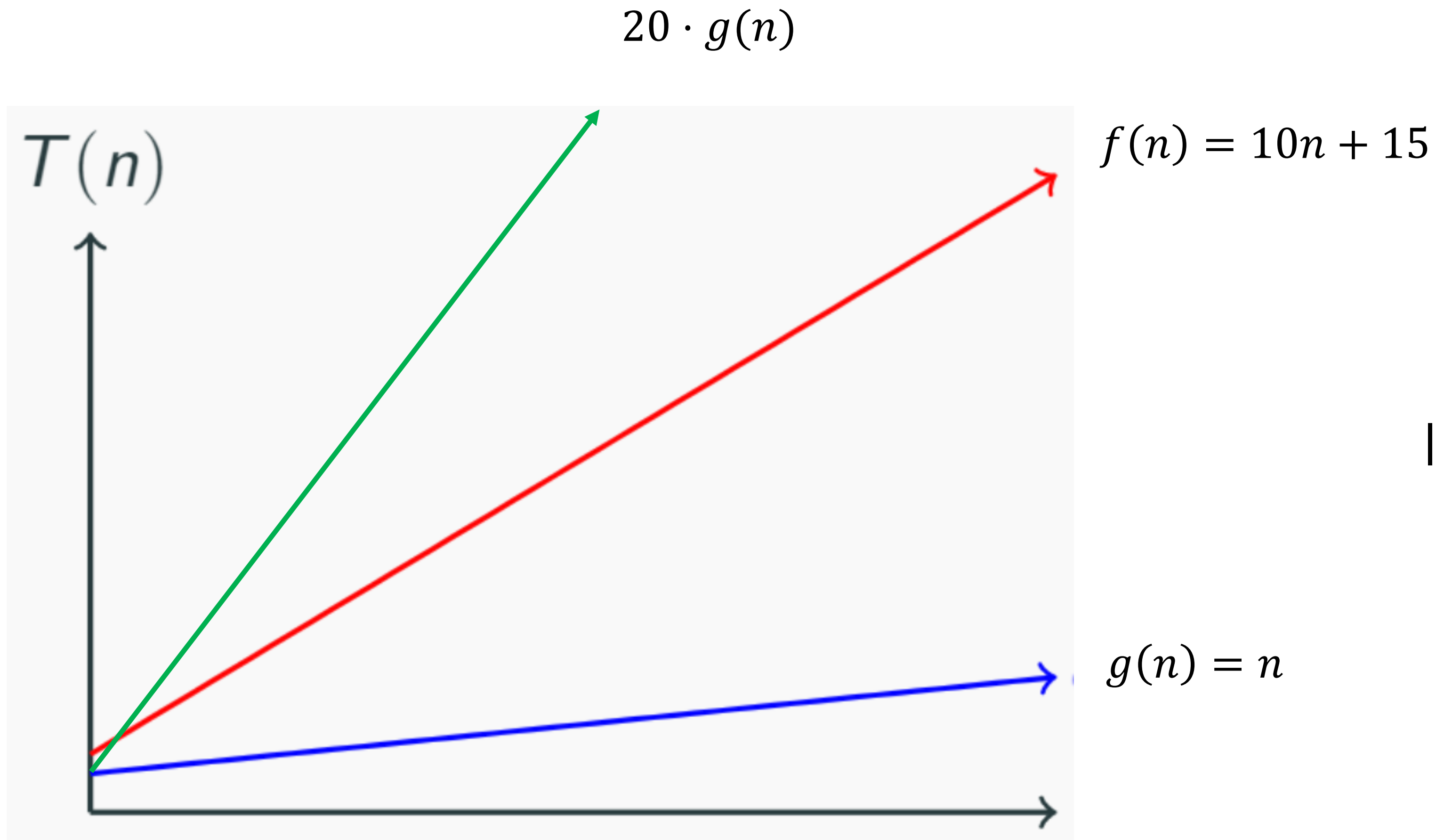
We also say that $f(n)$ is "dominated by" $g(n)$

Questions:

1. Is g(n) dominated by f(n) ?

2. Is f(n) dominated by g(n) ?

Question: Is f(n) dominated by g(n) ?



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 $f(n)$ is "dominated by" $g(n)$

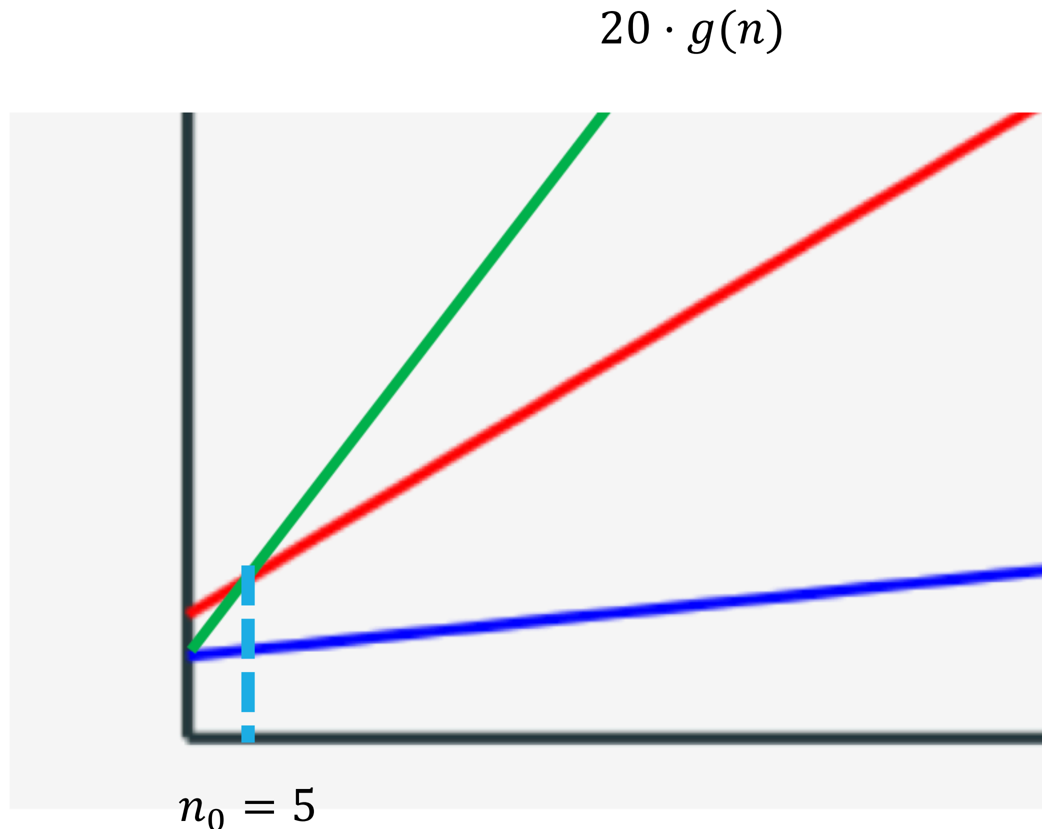
In this example, we can say

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

for c = 20

But is this true for all values of n?

Question: Is f(n) dominated by g(n) ?



$f(n) = 10n + 15$

$g(n) = n$

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 $f(n)$ is “dominated by” $g(n)$

In this example, we can say

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

for c = 20

for $n \geq n_0 = 5$

Formally 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.

The formal, mathematical definition is Big-O.

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 $f(n)$ is "dominated by" $g(n)$

Why c?

We want to be able to scale functions when comparing them.

Why $n \geq n_0$?

We want to be able to ignore initial values – we really care about large n values

Using the definition

Big-O

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

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

Let's show: $10n + 15$ is $O(n)$ $\qquad$ $10n + 15 \leq c\, n$

Scratch work:

$$10n \leq 10n$$
$$15 \leq 15n \text{ for } n \geq 1$$
$$10n + 15 \leq 25\text{n for } n \geq 1$$

Proof:

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

The inequality $10n \leq 10n$ is always true.

The inequality $15 \leq 15n$ is true for $n \geq 1$, as the right hand side is a factor of n more than the right hand side.

As long as both inequalities are true we can add them, thus

$10n + 15 \leq 25n$ holds as long as $n \geq 1$.

This is exactly the inequality we needed to show.

Edge case

Trick Question: $10n + 15$ is $O(n^2)$ Is this true of false?

It's true – it fits the definition.

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

But generally, when we ask for upper bound we mean the best upper bound or also called as the **tight** big-O bound. $O(n)$ is the tight bound for this example.

It is (almost always) technically correct to say your code runs in time $O(n!)$.

- DO NOT PULL THIS TRICK ON AN EXAM. Or in an interview.

O, Omega, Theta

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 $\Omega(g(n))$.

Notations

Big-O

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

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

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 $n \geq n_0$,

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

(1) We wrote "$f(n)$ is $O(g(n))$".

(2) Technically correct notation "$f(n) \in O(g(n))$"

$\in$ means 'element of'; e.g., 'a' $\in$ {'a', 'b', 'c', 3}

(3) Some people also write "$f(n) = O(g(n))$"

Use (1) or (2). But if you use (3) that's okay too, but try not to.

Useful Vocab

The most common running times all have fancy names:

$O(1)$ constant

$O(\log n)$ logarithmic

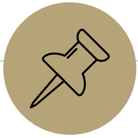
$O(n)$ linear

$O(n^2)$ quadratic

$O(n^3)$ cubic

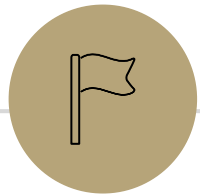
$O(n^c)$ polynomial (where c is a constant)

$O(c^n)$ exponential (where c is a constant)



Our intuition (hopefully hasn't changed)

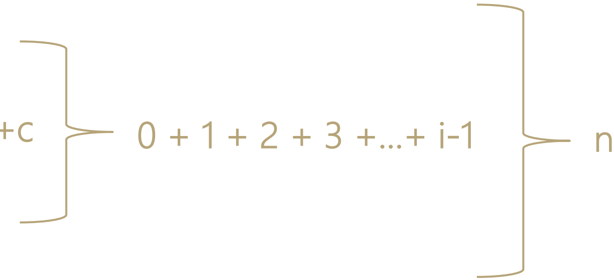
Function	O(?)
$n + 2$	$O(n)$
$4n + 3$	$O(n)$
$2n^2 + n + 100$	$O(n^2)$
n^2	$O(n^2)$
$100n^3 + n^2 + 20$	$O(n^3)$



Modeling complex functions

Modeling complex loops: Simplifying summations

```
for (int i = 0; i < n; i++) {
    for (int j = 0; j < i; j++) {
        System.out.println("Hello!"); +c
    }
}
```



(0c + 1c + 2c + 3c + … + i-1c)
+ (0c + 1c + 2c + 3c + … + i-1c)
+ (0c + 1c + 2c + 3c + … + i-1c)
+ repeat n times

$$T(n) = \sum_{i=0}^{n-1}\sum_{j=0}^{i-1} c = \sum_{i=0}^{n-1} ci$$ Summation of a constant

$$= c\sum_{i=0}^{n-1} i$$ Factoring out a constant

$$= c\frac{n(n-1)}{2}$$ Gauss's Identity

$$= \frac{c}{2}n^2 - \frac{c}{2}n$$ $O(n^2)$