CSCI 136: Data Structures and Advanced Programming

Lecture 9

Asymptotic analysis

Instructor: Dan Barowy

Williams

Announcements

- Lab 3: what's the deal with loops?
- Lab 1: feedback sent
- Lab 1: if feedback has mistakes…

Outline

1. Quiz
2. Study tip
3. Asymptotic analysis

Quiz

Code Review

Study tip #3: vocabulary

Every course is a “foreign language”

To learn effectively, study the vocab.

Maintain a “glossary.”


BIC 86 BIE

tion ; *style biblique*, qui offre de l'analogie avec le style de la Bible.
BICAPSULAIRE adj. *Bot.* Se dit des fruits à deux carpelles.
BICARBONATE n. m. Sel à base de soude, qui contient deux fois plus d'acide carbonique que le carbonate neutre pour le même poids de base.
BICARBURE n. m. Carbure qui contient deux portions de carbone.
BICARRÉ, E adj. *Alg.* Qui est élevé au carré du carré, à la quatrième puissance. *Equation bicarrée*, équation dont

amène les eaux jusqu'au moulin ; *bief d'aval*, celui qui s'étend à partir du moulin.
BIELLE n. f. *Méc.* Pièce d'une machine qui sert à communiquer et à transformer le mouvement.
BIEN n. m. Ce qui est bon, honnête, avantageux, convenable ; richesse ; propriété : *bien patrimonial*. Pl. *Biens de la terre*, production du sol ; *biens éternels*, le ciel. *Fig. Biens du corps*, la santé, la force ; *biens de l'esprit*, les talents ; *biens de l'âme*, les vertus. Adv. Beaucoup, fort : *il dort bien* ; formellement : *songez-y bien* ; à peu près : *il y a*

Asymptotic analysis

How do we know if an algorithm
is faster than another?

Why can't we just measure “wall time”?

Why can't we just measure "wall time"?

- Other things are happening at the same time
- Total running time usually varies by input
- Different computers may produce different results!

Let's just count instructions, then

- What do we count?
  - Count all computational steps?
  - What is a "step"?
  - What about steps inside loops?

Stepping back…

- How accurate do we need to be?
  - If one algorithm takes 64 steps and another 128 steps, do we need to know the precise number?

What we do

Instead of precisely counting steps, we usually develop an **approximation** of a program's **time** or **space complexity**.

This approximation **ignores tiny details** and focuses on the big picture: *how do time and space requirements grow as a function of the size of the input?*

Example

```
// pre: array length n > 0
public static int findPosOfMax(int[] arr) {
    int maxPos = 0
    for (int i = 1; i < arr.length; i++)
        if (arr[maxPos] < arr[i]) maxPos = i;
    return maxPos;
}
```

- Can we count steps exactly? Do we even want to?
  - `if` complicates counting
- Idea: **overcount**: assume `if` block always runs
  - in the worst case, it does
- Overcounting gives **upper bound** on run time
- Can also **undercount** for **lower bound**

Overcounting Example

```
// pre: array length n > 0
public static int findPosOfMax(int[] arr) {
    int maxPos = 0 .................................. line 1 cost: c1
    for (int i = 1; i < arr.length; i++) ............ line 2 cost: nc2
        if (arr[maxPos] < arr[i]) ................... line 3 cost: nc3
            maxPos = i; ............................. line 4 cost: nc4
    return maxPos; .................................. line 5 cost: c5
}
```

Total cost: $c_1 + nc_2 + nc_3 + nc_4 + c_5$

$= c_1 + n(c_2 + c_3 + c_4) + c_5$

$= n(c_2 + c_3 + c_4) + c_1 + c_5$

$= O(n)$

(as you shall see)

Focus is on order of magnitude

We can do this analysis for the **best**, **average**, and **worst** cases. We often focus on the worst case.

Big-O notation

Let **f** and **g** be real-valued functions that are defined on the same set of real numbers. Then **f** is of order **g**, written **f(n) is O(g(n))**, if and only if there exists a positive real number **c** and a real number n_0 such that for all **n** in in the common domain of **f** and **g**,

$$|f(n)| \leq c \times |g(n)|, \text{ whenever } n > n_0.$$

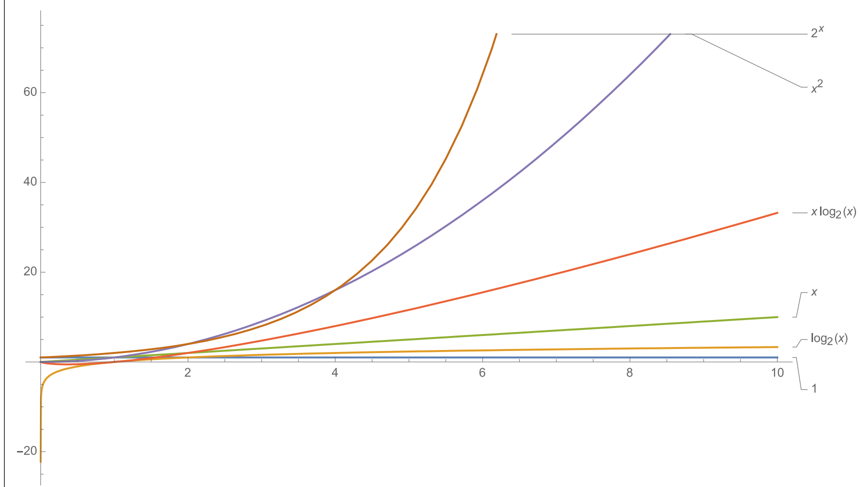
We read this as: "**f(n) is O(g(n))**" as "**f** of **n** is big-oh of **g** of **n**."

Function growth

Consider the following functions, for $x \geq 1$

- $f(x) = 1$
- $g(x) = \log_2(x)$ // Reminder: if x=2^n, $\log_2(x) = n$
- $h(x) = x$
- $m(x) = x \log_2(x)$
- $n(x) = x^2$
- $p(x) = x^3$
- $r(x) = 2^x$

Function growth



Function growth & Big-O

- Rule of thumb: **ignore multiplicative constants**
- Examples:
  - Treat n and n/2 as same order of magnitude
  - $n^2/1000$, $2n^2$, and $1000n^2$ are "pretty much" just n^2
  - $a_0n^k + a_1n^{k-1} + a_2n^{k-2} + \ldots a_k$ is roughly n^k
- The key is to find the **most significant** or **dominant term**
- Ex: $\lim_{x\to\infty} (3x^4 - 10x^3 - 1)/x^4 = 3$ (Why?)
  - So $3x^4 - 10x^3 - 1$ grows "like" x^4

Recap & Next Class

Today we learned:

Intro to asymptotic analysis

Next class:

Big-O notation

Induction