CS 361, Lecture 2

Jared Saia
University of New Mexico

Today's Outline

- Intro to Asymptotic Analysis
- Why do we care?
- Another interview problem
- Some solutions to the problem
- Todo list

Administrative

- Kanglin Xu, Office hours: M 4:30-6:30 and Weds 4:30 to 6:30, both in FEC 301C
- Sections: if you are in the CS dept, you must register for one of the two sections (Th 3:30-4:20 or F 1:00-1:50)
- Book: "Introduction to Algorithms" by Cormen, Leiserson, Rivest, and Stein
- Pretest due on Tuesday

How to analyze an algorithm?

- There are several resource bounds we could be concerned about: time, space, communication bandwidth, logic gates, etc.
- However, we are usually most concerned about time
- Recall that algorithms are independent of programming languages and machine types
- Q: So how do we measure resource bounds of algorithms

Random-access machine model

- We will use RAM model of computation in this class
- All instructions operate in serial
- All basic operations (e.g. add, multiply, compare, read, store, etc.) take unit time
- All "atomic" data (chars, ints, doubles, pointers, etc.) take unit space

Worst Case Analysis

- We'll generally be pessimistic when we evaluate resource bounds
- We'll evaluate the run time of the algorithm on the worst possible input sequence
- Amazingly, in most cases, we'll still be able to get pretty good bounds
- Justification: The "average case" is often about as bad as the worst case.

Example Analysis

- Consider the problem discussed last tuesday about finding a redundant element in an array
- Let's consider the more general problem, where the numbers are 1 to n instead of 1 to $1,000,000$

Algorithm 1

- Create a new "count" array of ints of size n, which we'll use to count the occurences of each number. Initialize all entries to 0
- Go through the input array and each time a number is seen, update its count in the "count" array
- As soon as a number is seen in the input array which has already been counted once, return this number

Algorithm 2

- Iterate through the input array, summing up all the numbers, let S be this sum
- Let $x = S - (n+1)n/2$
- Return x

Example Analysis: Time

- Worst case: Algorithm 1 does $5 * n$ operations (n inits to 0 in "count" array, n reads of input array, n reads of "count" array (to see if value is 1), n increments, and n stores into count array)
- Worst case: Algorithm 2 does $2 * n + 4$ operations (n reads of input array, n additions to value S, 4 computations to determine x given S)

Example Analysis: Space

- Worst Case: Algorithm 1 uses n additional units of space to store the "count" array
- Worst Case: Algorithm 2 uses 2 additional units of space

A Simpler Analysis

- Analysis above can be tedious for more complicated algorithms
- In many cases, we don't care about constants. $5n$ is about the same as $2n + 4$ which is about the same as $an + b$ for any constants a and b
- However we do still care about the difference in space: n is very different from 2
- Asymptotic analysis is the solution to removing the tedium but ensuring good analysis

What is asymptotic analysis?

- A tool for analyzing time and space usage of algorithms
- Assumes input size is a variable, say n, and gives time and space bounds as a function of n
- Ignores multiplicative and additive constants
- Concerned only with the *rate* of growth
- E.g. Treats run times of n, $10,000 * n + 2000$, and $.5n + 2$ all the same (We use the term $O(n)$ to refer to all of them)

What is Asymptotic Analysis?(II)

- Informally, O notation is the leading (i.e. quickest growing) term of a formula with the coefficient stripped off
- O is sort of a relaxed version of "$\leq$"
- E.g. n is $O(n)$ and n is also $O(n^2)$
- By convention, we use the smallest possible O value i.e. we say n is $O(n)$ rather than n is $O(n^2)$

More Examples

- E.g. n, $10,000n - 2000$, and $.5n + 2$ are all $O(n)$
- $n + \log n$, $n - \sqrt{n}$ are $O(n)$
- $n^2 + n + \log n$, $10n^2 + n - \sqrt{n}$ are $O(n^2)$
- $n \log n + 10n$ is $O(n \log n)$
- $10 * \log^2 n$ is $O(\log^2 n)$
- $n\sqrt{n} + n \log n + 10n$ is $O(n\sqrt{n})$
- $10,000$, 2^{50} and 4 are $O(1)$

More Examples

- Algorithm 1 and 2 both take time $O(n)$
- Algorithm 1 uses $O(n)$ extra space
- But, Algorithm 2 uses $O(1)$ extra space

Questions

Express the following in O notation

- $n^3/1000 - 100n^2 - 100n + 3$
- $\log n + 100$
- $10 * \log^2 n + 100$
- $\sum_{i=1}^{n} i$

A digression on logs

It rolls down stairs alone or in pairs,
and over your neighbor's dog,
it's great for a snack or to put on your back,
it's log, log, log!
- "The Log Song" from the Ren and Stimpy Show

- The log function shows up very frequently in algorithm analysis
- As computer scientists, when we use log, we'll mean $\log_2$ (i.e. if no base is given, assume base 2)

Definition

- $\log_x y$ is by definition the value z such that $x^z = y$
- $x^{\log_x y} = y$ by definition

Examples

- $\log 1 = 0$
- $\log 2 = 1$
- $\log 32 = 5$
- $\log 2^k = k$

Note: $\log n$ is way, way smaller than n for large values of n

Examples

- $\log_3 9 = 2$
- $\log_5 125 = 3$
- $\log_4 16 = 2$
- $\log_{24} 24^{100} = 100$

Facts about exponents

Recall that:

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

From these, we can derive some facts about logs

Facts about logs

To prove both equations, raise both sides to the power of 2, and use facts about exponents

- Fact 1: $\log(xy) = \log x + \log y$
- Fact 2: $\log a^c = c \log a$

Memorize these two facts

Incredibly useful fact about logs

- Fact 3: $\log_c a = \log a / \log c$

To prove this, consider the equation $a = c^{\log_c a}$, take $\log_2$ of both sides, and use Fact 2. **Memorize this fact**

Log facts to memorize

- Fact 1: $\log(xy) = \log x + \log y$
- Fact 2: $\log a^c = c \log a$
- Fact 3: $\log_c a = \log a / \log c$

These facts are sufficient for all your logarithm needs. (You just need to figure out how to use them)

Questions

Simplify and give O notation for the following:

- $\log 10 * x^2$
- $\log^2 x$
- $\log \log \sqrt{n}$
- $2^{\log_4 x}$

Logs and O notation

- Note that $\log_8 n = \log n / \log 8$.
- Note that $\log_{600} n^{200} = 200 * \log n / \log 600$.
- Note that $\log_{100000} 30{*}n^2 = 2{*}\log n / \log 100000 {+} \log 30 / \log 100000$.
- Thus, $\log_8 n$, $\log_{600} n^{600}$, and $\log_{100000} 30{*}n^2$ are all $O(\log n)$
- In general, for any constants k_1 and k_2, $\log_{k_1} n^{k_2} = k_2 \log n / \log k_1$, which is just $O(\log n)$

Take Away

- All log functions of form $k_1 \log k_2 n^{k_3}$ for constants k_1, k_2 and k_3 are $O(\log n)$
- For this reason, we don't really "care" about the base of the log function when we do asymptotic notation
- Thus, binary search, ternary search and k-ary search all take $O(\log n)$ time

Todo

- Finish pretest, due next Tuesday!
- Sign up for the class mailing list (cs361)
- Read Chapter 3 (Growth of Functions) in textbook