ALGORITHMS

INTRODUCTION

PROOF BY

ASYMPTOTIC NOTATION

The Course

- Purpose: a rigorous introduction to the design and analysis of algorithms
  - Not a lab or programming course
  - Not a math course, either
- Textbook: *Introduction to Algorithms*, Cormen, Leiserson, Rivest, Stein
  - The “Big White Book”
  - Second edition: now “Smaller Green Book”
  - An excellent reference you should own

The Course

- Instructor: David Luebke
  - *luebke@cs.virginia.edu*
  - Office: Olsson 219
  - Office hours: 2-3 Monday, 10-11 Thursday
- TA: Pavel Sorokin
  - Office hours and location TBA

The Course

- Grading policy:
  - Homework: 30%
  - Exam 1: 15%
  - Exam 2: 15%
  - Final: 35%
  - Participation: 5%

The Course

- Prerequisites:
  - CS 202 w/ grade of C- or better
  - CS 216 w/ grade of C- or better
  - CS 302 recommended but not required

The Course

- Format
  - Three lectures/week
  - Homework most weeks
    - Problem sets
    - Maybe occasional programming assignments
  - Two tests + final exam

Review: Induction

- Suppose
  - S(k) is true for fixed constant k
    - Often k = 0
  - S(n) → S(n+1) for all n >= k
- Then S(n) is true for all n >= k

Proof By Induction

- Claim:S(n) is true for all n >= k
- Basis:
  - Show formula is true when n = k
- Inductive hypothesis:
  - Assume formula is true for an arbitrary n
- Step:
  - Show that formula is then true for n+1

Induction Example: Gaussian Closed Form

- Prove $1 + 2 + 3 + \ldots + n = n(n+1) / 2$
  - Basis:
    - If $n = 0$, then $0 = 0(0+1) / 2$
  - Inductive hypothesis:
    - Assume $1 + 2 + 3 + \ldots + n = n(n+1) / 2$
  - Step (show true for n+1):

$$1 + 2 + \ldots + n + n+1 = (1 + 2 + \ldots + n) + (n+1)$$

$$= n(n+1)/2 + n+1 = [n(n+1) + 2(n+1)]/2$$

$$= (n+1)(n+2)/2 = (n+1)(n+1 + 1) / 2$$

Induction Example: Geometric Closed Form

- Prove $a^0 + a^1 + \ldots + a^n = (a^{n+1} - 1)/(a - 1)$ for all $a \neq 1$
  - Basis: show that $a^0 = (a^{0+1} - 1)/(a - 1)$

    $a^0 = 1 = (a^1 - 1)/(a - 1)$
  - Inductive hypothesis:
    - Assume $a^0 + a^1 + \ldots + a^n = (a^{n+1} - 1)/(a - 1)$
  - Step (show true for n+1):

    $a^0 + a^1 + \ldots + a^{n+1} = a^0 + a^1 + \ldots + a^n + a^{n+1}$

    $= (a^{n+1} - 1)/(a - 1) + a^{n+1} = (a^{n+1+1} - 1)/(a - 1)$

Induction

- We've been using *weak induction*
- *Strong induction* also holds
  - Basis: show S(0)
  - Hypothesis: assume S(k) holds for arbitrary k <= n
  - Step: Show S(n+1) follows
- Another variation:
  - Basis: show S(0), S(1)
  - Hypothesis: assume S(n) and S(n+1) are true
  - Step: show S(n+2) follows

Asymptotic Performance

- In this course, we care most about *asymptotic performance*
  - How does the algorithm behave as the problem size gets very large?
    - Running time
    - Memory/storage requirements
    - Bandwidth/power requirements/logic gates/etc.

Asymptotic Notation

- By now you should have an intuitive feel for asymptotic (big-O) notation:
  - *What does $O(n)$ running time mean? $O(n^2)$? $O(n \lg n)$?*
  - *How does asymptotic running time relate to asymptotic memory usage?*
- Our first task is to define this notation more formally and completely

Analysis of Algorithms

- Analysis is performed with respect to a computational model
- We will usually use a generic uniprocessor random-access machine (RAM)
  - All memory equally expensive to access
  - No concurrent operations
  - All reasonable instructions take unit time
    - Except, of course, function calls
  - Constant word size
    - Unless we are explicitly manipulating bits

Input Size

- Time and space complexity
  - This is generally a function of the input size
    - E.g., sorting, multiplication
  - How we characterize input size depends:
    - Sorting: number of input items
    - Multiplication: total number of bits
    - Graph algorithms: number of nodes & edges
    - Etc

Running Time

- Number of primitive steps that are executed
  - Except for time of executing a function call most statements roughly require the same amount of time
    - y = m * x + b
    - c = 5 / 9 * (t - 32)
    - z = f(x) + g(y)
- We can be more exact if need be

Analysis

- Worst case
  - Provides an upper bound on running time
  - An absolute guarantee
- Average case
  - Provides the expected running time
  - Very useful, but treat with care: what is “average”?
    - Random (equally likely) inputs
    - Real-life inputs