Computer Science 2300: Data Structures and Algorithms

RPI, Spring 2009
Instructor: Sanmay Das

Course Structure

- 2 lectures (MTh 12-1:30) and 1 lab (W -- must attend your assigned lab)
- Small lab projects (~6), plus ~8 homeworks (20% + 35% of grade)
- One midterm (20%), plus a final exam (25%)
- http://www.cs.rpi.edu/~sanmay/teaching/cs2300/

What is this class about?

- Thinking like a computer scientist
  - Continuing the transition from programmer
- Learning how to approach problems
- NOT about code. We assume you are competent in C++

Textbooks

- Primary: *Algorithms* by Dasgupta, Papadimitriou, and Vazirani (more readable) [DPV]
- Secondary: *Introduction to Algorithms* by Cormen, Leiserson, Rivest and Stein (more encyclopedic) [CLRS]
- Any programming reference you need, but none assigned (suggestion, Stroustrup)

Prerequisites

- CS2 and Discrete Structures
- Taking Discrete Structures right now?
  - Motivated? Confident about picking up discrete math quickly?
  - Check the appendices in CLRS
- Programming: you won't need any new tricks, but there will be little handholding!

Staff

- My office hours: Mondays after class. Plus by appointment
- TAs: Eyuphan Bulut and Ashok Sukumaran
  - Primary point of contact: your lab TA
  - Available during lab (OH in non-lab weeks, but not this week) and by appointment
- UTAs will be available during labs to help you out

Syllabus and Course Policies

- Role of labs
- You are responsible for all announcements made in lecture, posted on the website, or sent via e-mail
- Late-day policy
- Collaboration policy
- Grading policies

Lectures

- I strongly encourage attendance. You are responsible for everything discussed.
- I'm a big fan of questions. Both receiving them and posing them.
- If no one answers my questions I will wait as long as it takes until someone does

Data Structures & Algorithms

- MCDXLVIII + DCCCXII = ?
- Answer: MMCCLX
- How did you do it? 1448 + 812 = 2260?

Addition, contd.

- Note that representation is key (sort of like a data structure)
- Algorithms operate on data
- Decimal addition is easy, roman numeral addition is not!

A Brief Tour of the Class

- Introduction. Correctness and running time analysis
- Divide-and-conquer algorithms. How to reduce problems.
  - Faster integer multiplication (!)
  - Sorting
  - Median-finding

Graph Algorithms

- Why graphs?
  - Encapsulate many problems
  - The web!!
- Graph representations
  - Lists or matrices?
- Exploring graphs and finding short paths

Data Structures

- More familiar, but more advanced material, which will be coupled with interesting new algorithms:
  - Heaps
  - Trees
  - Hash tables

More advanced algorithms

- Dynamic programming
- Introduction to NP-completeness -- when can we (probably) not solve a (large) problem exactly?

Analysis of Algorithms

- Two things we care deeply about:
  - Proving correctness
  - Analyzing running time

Fibonacci Numbers

- 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, ...
- From rabbit reproduction to Vedic metre...
- Rule:
  - $F_n = F_{n-1} + F_{n-2}, n > 1$
  - $F_n = 1, n = 1$
  - $F_n = 0, n = 0$

Some Properties

- Grow almost as fast as powers of 2!
- $F_n \approx 2^{0.694n}$
- F_{100} is 21 digits long!

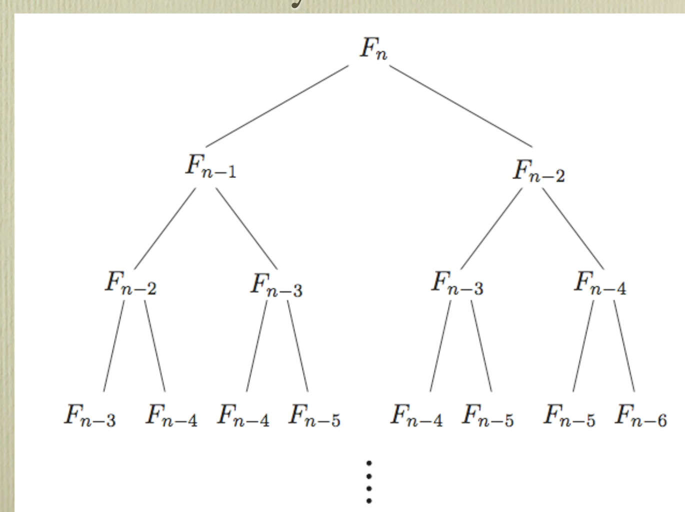
How to compute F_n?

- function fib1(n)
  - if n = 0: return 0
  - if n = 1: return 1
  - return fib1(n-1) + fib1(n-2)
- Correctness?
- Time taken?

Time

- T(n) : # computer steps taken to compute fib1(n)
- $T(n) \leq 2$ for $n \leq 1$
- $T(n) = T(n-1) + T(n-2) + 3$ for $n > 1$
- $T(n) \geq F_n$!!
- This is very bad news. Exponential complexity!

Why so slow?



A Better Algorithm

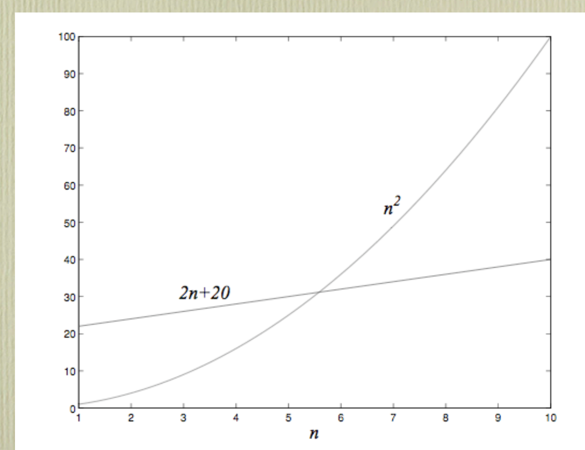
- if n = 0 : return 0
- create an array f[0...n]
- f[0] = 0, f[1] = 1
- for i = 2...n:
  - f[i] = f[i-1] + f[i-2]
- return f[n]

Running Time?

- Linear!
- Caveat:
  - When does it stop making sense to think of each computer operation as 1 time unit?
- F_n is about 0.694 bits long. We'll quickly exceed 32 (or even 128) bits, so can't just assume one operation.

Big-O Notation

- f(n), g(n) : functions from integers to reals
- f is O(g) if ∃ (positive) constants c and n_0 such that $f(n) \le c\, g(n)$ for $n \ge n_0$
- Intuition: f grows no faster than g



Analogs

- $f = \Omega(g)$: $g = O(f)$
- $f = \Theta(g)$: $f = O(g)$ and $g = O(f)$
- Small o: strictly slower growth

Rules of Thumb

- Exponentials dominate polynomials
- Polynomials dominate logs
- n^a dominates n^b if $a > b$
- Omit multiplicative constants

Maximum Subsequence Sum

- Given a sequence of integers $A_1, \ldots A_n$, find the maximum value of $\sum_{k=i}^{j} A_k$
- Example: -2, 11, -4, 13, -5, -2
- Answer: 20

Algorithm 1

- maxSum = 0
- for i in 0:n-1
  - for j in i:n-1
    - thisSum = 0
      - for k in i:j
        - thisSum = thisSum + a[k]
      - if (thisSum > maxSum)
        - maxSum = thisSum

Running Time?

- Simple analysis: 3 loops, one inside the other, each of worst case size n : $O(n^3)$
- More sophisticated: still $O(n^3)$

$$\sum_{i=0}^{n-1}\sum_{j=i}^{n-1}\sum_{k=i}^{j} 1 = \quad ?$$

$$\sum_{k=i}^{j} 1 = \quad j - i + 1$$

$$\sum_{j=i}^{n-1}(j-i+1) = \quad \frac{(n-i+1)(n-i)}{2}$$

$$\sum_{i=0}^{n-1}\frac{(n-i+1)(n-i)}{2} = \quad \frac{n^3+3n^2+2n}{6}$$

An $O(n^2)$ Algorithm

- maxSum = 0
- for i in 0:n-1
  - thisSum = 0
  - for j in i:n-1
    - thisSum = thisSum + a[j]
    - if (thisSum > maxSum)
      - maxSum = thisSum
- return maxSum

An O(n) Algorithm

- maxSum = 0, thisSum = 0
- for j in 0:n-1
  - thisSum = thisSum + a[j]
  - if (thisSum > maxSum)
    - maxSum = thisSum
  - else if (thisSum < 0)
    - thisSum = 0
- return maxSum

Why Does This Work?

- Key observation: Any negative subsequence cannot be a prefix of the optimal subsequence. We compute the maximum subsequence ending at position j
- Whenever a subsequence first becomes negative, we can reset and consider only subsequences that start beyond j (why?)