CSE 421
Introduction to Algorithms

Overview

Larry Ruzzo
CSE 421, Wi '15: Introduction to Algorithms

Lecture: MGH 241 (schedules) MWF 1:30 - 2:20

Instructor: Larry Ruzzo, ruzzo@cs
TA: Katie Doroschak, kдоросча@cs W 2:30 - 3:30? CSE 554 206-543-6298
Haine Levey, slevey@cs Th 3:00 - 4:00? CSE 2xx
Mert Saglam, mert@cs XX 99:00 - 99:00? CSE 2xx

Course Email: cse421w15@uw.edu. Staff announcements and general interest student email by subscription only. Details, etc. The instructor and TA are subscribed to this list. Enrolled students are as well, but may need to request subscription options. Messages are automatically archived.

Catalog Description: Techniques for design of efficient algorithms and proving lower bounds on computational complexity. Particular algorithms for sorting, searching, graph problems, string problems, pattern matching.

Prerequisites: either CSE 374 or CSE 332.

Credits: 3

Course Details: Final. Homework will be a mix of paper & pencil exercises and programming. Overall weights 55%, 15%, 30%.

Policy: Unless otherwise announced, homework is due at the start of class on the due date. 20% off per day thereafter (business day, e.g., Monday for Friday due dates).

Extra Credit: Assignments may include "extra credit" sections. These will enrich your understanding of the material, but at a low points per hour ratio. Do them for the glory, not the points, and don't start extra credit until the basics are complete.


Portions of the CSE 421 Web may be reprinted or adapted for academic nonprofit purposes, providing the source is accurately quoted and duly credited. The CSE 421 Web © 1993-2015, the Authors and the Department of Computer Science and Engineering, University of Washington.
What you have to do

Weekly Homework (~55% of grade)
  Programming?
    perhaps some small projects
  Written homework assignments
    English exposition and pseudo-code
    Analysis and argument as well as design

Midterm / Final Exam (~15% / 30%)

Late Policy:
  Papers and/or electronic turnins are due at the start of class on the due date; minus 20% per day thereafter
Textbook

What the course is about

Design of Algorithms

- design methods
- common or important types of problems
- analysis of algorithms - efficiency
- correctness proofs
What the course is about

Complexity, NP-completeness and intractability

solving problems in principle is not enough

algorithms must be efficient

some problems have no efficient solution

NP-complete problems

important & useful class of problems whose solutions
(seemingly) cannot be found efficiently, but can be
checked easily
Very Rough Division of Time

Algorithms (6-7 weeks)

Analysis

Techniques:
- graph search
- greedy
- divide & conquer
- dynamic programming
- reduction

Applications

Complexity & NP-completeness (2-3 weeks)

Check online schedule page for (evolving) details
Complexity Example

Cryptography (e.g., RSA, SSL in browsers)

Secret: \( p, q \) prime, say 512 bits each

Public: \( n \) which equals \( p \times q \), 1024 bits

In principle

*there is an algorithm* that given \( n \) will find \( p \) and \( q \):

try all \( 2^{512} > 1.3 \times 10^{154} \) possible \( p \)'s: kinda slow…

In practice

no fast algorithm known for this problem (on non-quantum computers)

security of RSA depends on this fact

(“quantum computing”: strongly driven by possibility of changing this)
Algorithms versus Machines

We all know about Moore’s Law and the exponential improvements in hardware...

Ex: sparse linear equations over 25 years

10 orders of magnitude improvement!
Algorithms or Hardware?

25 years progress solving sparse linear systems

hardware: 4 orders of magnitude

Source: Sandia, via M. Schultz
Algorithms or Hardware?

25 years progress solving sparse linear systems

hardware: 4 orders of magnitude

software: 6 orders of magnitude

Source: Sandia, via M. Schultz
Algorithms or Hardware?

The N-Body Problem:

in 30 years
- $10^7$ hardware
- $10^{10}$ software

Source: T. Quinn
Algorithm: definition

Procedure to accomplish a task or solve a well-specified problem

Well-specified: know what all possible inputs look like and what output looks like given them

“accomplish” via simple, well-defined steps

Ex: sorting names (via comparison)

Ex: checking for primality (via +, -, *, /, \leq)
Goals

Correctness
  often subtle

Analysis
  often subtle

Generality, Simplicity, ‘Elegance’

Efficiency
  time, memory, network bandwidth, …
Algorithms: a sample problem

Printed circuit-board company has a robot arm that solders components to the board.

Time: proportional to total distance the arm must move from initial rest position around the board and back to the initial position.

For each board design, find best order to do the soldering.
Printed Circuit Board
Printed Circuit Board
A Well-defined Problem

Input: Given a set $S$ of $n$ points in the plane

Output: The shortest cycle tour that visits each point in the set $S$ once.

Better known as “TSP”

How might you solve it?
Nearest Neighbor Heuristic

Start at some point $p_0$
Walk first to its nearest neighbor $p_1$
Repeatedly walk to the nearest unvisited neighbor $p_2$, then $p_3$, ... until all points have been visited
Then walk back to $p_0$

**heuristic:**
A rule of thumb, simplification, or educated guess that reduces or limits the search for solutions in domains that are difficult and poorly understood. May be good, but usually *not* guaranteed to give the best or fastest solution.
Nearest Neighbor Heuristic

\[ p_0, p_1, p_6 \]
An input where NN works badly

length $\sim 84$
An input where NN works badly

optimal soln for this example
length = 63.8
Revised idea - Closest pairs first

Repeatedly join the closest pair of points
(s.t. result can still be part of a single loop in the end. I.e., join endpoints, but not points in middle, of path segments already created.)

How does this work on our bad example?
Another bad example
Another bad example

\[ 6 + \sqrt{10} = 9.16 \]

vs

8
Something that works

“Brute Force Search”: For each of the $n! = n(n-1)(n-2)\ldots 1$ orderings of the points, check the length of the cycle you get
Keep the best one
Two Notes

The two incorrect algorithms were “greedy”
  Often very natural & tempting ideas
  They make choices that look great “locally” (and never reconsider them)
  When greed works, the algorithms are typically efficient
  BUT: often does not work - you get boxed in

Our correct alg avoids this, but is incredibly slow
  20! is so large that checking one billion orderings per second would take 2.4 billion seconds (around 70 years!)
  And growing: $n! \sim \sqrt{2 \pi n} \cdot (n/e)^n \sim 2^{O(n \log n)}$
The Morals of the Story

Algorithms are important
  Many performance gains outstrip Moore’s law
Simple problems can be hard
  Factoring, TSP
Simple ideas don’t always work
  Nearest neighbor, closest pair heuristics
Simple algorithms can be very slow
  Brute-force factoring, TSP
For some problems, even the best algorithms are slow

Course Goals:
  formalize these ideas, and
  develop more sophisticated approaches