CSE 417, Wi '22: Algorithms & Computational Complexity

Lecture: (By Zoom, Week 1 at least), MWF 9:30-10:20
Zoom: https://washington.zoom.us/j/94357067917 Requires login to Zoom with a UWNetID.

Instructor: Larry Ruzzo, ruzzo@cs
TAs: Nathan Akkaraphab, akkanath@cs TBD
     Todor Dimitrov, todord@cs TBD
     William Viet Nguyen, williamv@cs TBD
     Lin Qiu, lq9@cs TBD
     Luna Wang, zhengw28@cs TBD
     Yifan Zhang, yifanz47@cs TBD
     Yilin Zhang, yilinz24@cs TBD

Office Hours Location Phone
TBD Zoom

Course Email: cse417a_wi22@uw.edu. Staff and TAs are subscribed to this list. Email to archived.

Discussion: Discuss homework, etc.

Late Policy: TBD
Required Text: Algorithm Design by Jon Kleinberg and Eva Tardos. Addison Wesley, 2006. (Available from U Book Store, Amazon, etc.)
References: See Schedule & Reading.

Portions of the CSE 417 Web may be reprinted or adapted for academic nonprofit purposes, providing the source is accurately quoted and duly credited. The CSE 417 Web: © 1993-2022, the
What you’ll have to do

100%
(≈55% of grade)

Homework

Programming
Several small projects

Written homework assignments
English exposition and pseudo-code
Analysis and argument as well as design

Midterm / Final Exam
(≈15% / 30%)

Late Policy: TBD

“Mastery Grading”:
If you can solve it, you get credit for “mastering” the concept, with a chance to resubmit/regrade some work each week. Details TBD

Papers and/or electronic turnins are due at the start of class on the due date. 10% off for one day late; 15% 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 (7-8 weeks)
- Analysis of Algorithms
- Basic Algorithmic Design Techniques
- 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

You may 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
Algorithms or Hardware?

SAT/SMT Solvers: 1000x improvement in a dozen years

- Solver-based programming languages
- Compiler optimizations using solvers
- Solver-based debuggers
- Solver-based type systems
- Solver-based concurrency bugfinding
- Solver-based synthesis
- Bio & Optimization

Data courtesy of Dr. Vijay Ganesh, U. Waterloo
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 +, -, *, /, ≤)
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 *not* proven to give the best or fastest solution.
Nearest Neighbor Heuristic
An input where NN works badly

length ~ 84
An input where NN works badly

optimal soln for this example
length = 63.8 (vs ~84 above)
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?
A bad example for “close pairs”
A bad example for “close pairs”

\[
1.5 \quad 1.5
\]

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

vs

8
Something that works

“Brute Force Search”: For each of the $n! = n(n-1)(n-2)\ldots1$ orderings of the points, check the length of the cycle you get. Keep the best one.

(Easy to see that it’s correct, but slow!)
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 software gains outstrip hardware gains (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
Our field has been struggling with this problem for years.

Struggle no more! I'm here to solve it with algorithms!

Six months later: Wow, this problem is really hard. You don't say.

https://xkcd.com/