CSE 417: Algorithms and Computational Complexity

Lecture I: Overview

Winter 2019

Larry Ruzzo
CSE 417, Wi ‘19: Algorithms & Computational Complexity

Lecture: JHN 102 (room info) MWF 1:30-2:20

Instructor: Larry Ruzzo, ruzzo@cs
TAs: Yuqing Ai, yuqingai@cs
      Daniel Jones, dcjones@cs
      Saidutt Nimmagadda, nimm@cs
      Alex Okeson, aokeson@cs

Office Hours Location Phone
F 2:30-3:30 CSE 554 (206) 543-6298 <--11/11 expected
Tu 1:00-2:00 4th floor breakout TBA
Tu 2:30-3:30 CSE 021

Course Email: cse417a_wi19@uw.edu. Staff announcements and course related issues should go to this email, e.g., questions about homework, lectures, etc. The instructor and TAs are subscribed to this list. Messages about personal matters should change their default subscription options. Messages are:

Discussion Board: Also for questions about homework, etc.

Catalog Description: Succinct algorithms for manipulating graphs and trees. Efficient algorithms for manipulating graphs and trees. Turing machines. Time and space complexity. NP-

Reading: Homework, Midterm, Final. Homework will be a mix of paper & pencil exercises and programing. Overall weights 55%, 15%, 30%, roughly.

Late Policy: Papers and/or electronic turins are due at the start of class on the due date. 10% off for up to one day late; additional 15% per day thereafter. (Day = calendar day, i.e., Sunday is later than Saturday.)

Textbooks: Algorithm Design by Jon Kleinberg and Eva Tardos. Addison Wesley, 2006. (Available from U Book Store, Amazon, etc.)
What you’ll have to do

Homework (~55% of grade)
  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:
  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 weeks)
- Analysis of Algorithms
- Basic Algorithmic Design Techniques
- Applications

Complexity & NP-completeness (3 weeks)

Check online schedule page for (evolving) details

<table>
<thead>
<tr>
<th>Week 1</th>
<th>Due</th>
<th>Lecture Topic</th>
<th>Reading</th>
</tr>
</thead>
<tbody>
<tr>
<td>1/2-1/6</td>
<td>M</td>
<td>Holiday</td>
<td></td>
</tr>
<tr>
<td></td>
<td>W</td>
<td>Intro, Examples &amp; Complexity</td>
<td>Ch. 1, Ch. 2</td>
</tr>
<tr>
<td></td>
<td>F</td>
<td>Intro, Examples &amp; Complexity</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Week 2</th>
<th>Due</th>
<th>Lecture Topic</th>
<th>Reading</th>
</tr>
</thead>
<tbody>
<tr>
<td>1/9-1/13</td>
<td>M</td>
<td>Intro, Examples &amp; Complexity</td>
<td></td>
</tr>
<tr>
<td></td>
<td>W</td>
<td>Graph Algorithms</td>
<td>Ch. 3</td>
</tr>
<tr>
<td></td>
<td>F</td>
<td>Graph Algorithms</td>
<td></td>
</tr>
</tbody>
</table>
Complexity Example

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

Secret: p,q prime, say 512 bits each
Public: n which equals p x 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!
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

G.E. = Gaussian Elimination
SOR = Successive OverRelaxation
CG = Conjugate Gradient
Algorithms or Hardware?

The N-Body Problem:

in 30 years
$10^7$ hardware
$10^{10}$ software
Algorithms or Hardware?

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

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 +, -, *, /, \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 $\sim 84$

$p_0$
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?
Another bad example

1

1.5  1.5
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
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/