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 (Monday, for Friday due dates); 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 (7 weeks)
- Analysis of Algorithms
- Basic Algorithmic Design Techniques
- Graph Algorithms

Complexity & NP-completeness (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 x q, 1024 bits

In principle
- there is an algorithm that given n will find p and q:
  - try all $2^{512}$ possible p’s, an astronomical number

In practice
- no efficient algorithm is known for this problem
- security of RSA depends on this fact
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!
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 +, -, *, /, ≤)

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
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 $S$.

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 $p_0$

Nearest Neighbor Heuristic

An input where NN works badly

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.
An input where NN works badly

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

optimal soln for this example
length ~ 64

6 + √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 per second would take 2.4 billion seconds (around 70 years!)

Something that “works” (differently)

1. Find Min Spanning Tree

Something that “works” (differently)

2. Walk around it
Something that “works” (differently)

3. Take shortcuts (instead of revisiting)

Something that “works” (differently): Guaranteed Approximation

Does it seem wacky?
Maybe, but it’s always within a factor of 2 of the best tour!

deleting one edge from best tour gives a spanning tree, so \( \text{Min spanning tree} < \text{best tour} \)

\[ \text{best tour} \leq \text{wacky tour} \leq 2 \times \text{MST} < 2 \times \text{best} \]

The Morals of the Story

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

Changing your objective can be good
- Guaranteed approximation for TSP