CSE 417: Algorithms and Computational Complexity

1: Organization & Overview

Winter 2006

Larry Ruzzo

University of Washington
Computer Science & Engineering

CSE 417, Wi '06: Algorithms & Computational Complexity

CSE Home | About Us | Search | Contact Info

Administrative
- FAQ
- Draft Schedule

Email
- Mail archive

Assignments

Solutions

Lecture Notes

Time: MWF 2:30-3:20
Place: Low 101 (schematic)

		Office Hours		Phone
Instructor:	Larry Ruzzo, ruzzo@cs,	MF 12:00- 1:00,	CSE 554,	543-6298
TA:	Paul Pham, ppham@cs,		CSE ???,	

Catalog Description: Design and analysis of algorithms and data structures. Efficient algorithms for manipulating graphs and strings. Fast Fourier Transform. Models of computation, including Turing machines. Time and space complexity. NP-complete problems and undecidable problems.
Prerequisite: CSE 373
Credits: 3

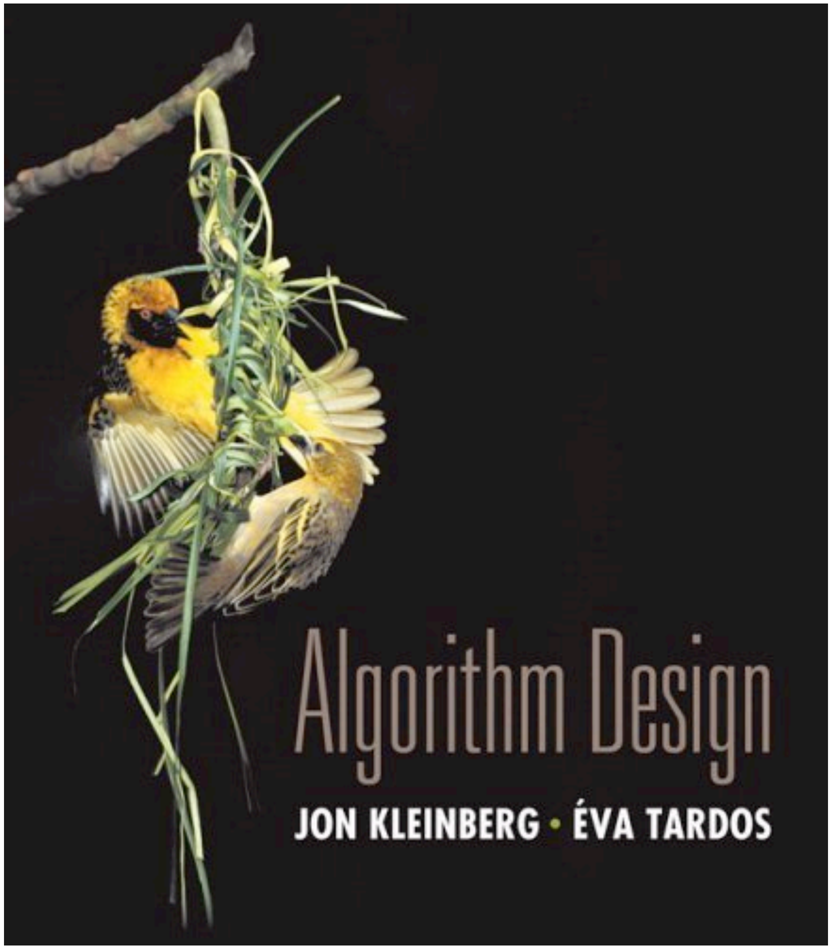
Class email lists: cse417a_wi06@u.washington.edu. Use this list to ask and/or answer questions about homework, lectures, etc. The instructor and TA are subscribed to this list, and will answer questions, but I almost always find that the questions and answers are of general interest, and that your fellow students often will answer more quickly (and more clearly) than the staff can. Students should be automatically subscribed within 24 hours of registration. You can modify your subscription options. All messages are automatically archived. General information about the email system is here. Questions not of general interest should be directed to the instructor and/or TA.

http://www.cs.washington.edu/417

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



- *Algorithm Design* by Jon Kleinberg and Eva Tardos. Addison Wesley, 2006.

What the course is about

- Design of Algorithms
  - design methods
  - common or important types of problems
  - how to analyze algorithms
  - correctness proofs

What the course is about

- Complexity and NP-completeness
  - solving problems in principle is not enough
    - algorithms must be **efficient**
  - NP
    - class of useful problems whose solutions can be easily checked but not necessarily found efficiently
  - NP-completeness
    - understanding when problems are hard to solve

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

University of Washington
Computer Science & Engineering

CSE 417, Wi '06: *Approximate* Schedule

CSE Home   About Us

		Due	Lecture Topic	Reading
Week 1 1/2-1/6	M		Holiday	
	W		Intro, Examples & Complexity	Ch. 1; Ch. 2
	F		Intro, Examples & Complexity	
Week 2 1/9-1/13	M		Intro, Examples & Complexity	
	W		Graph Algorithms	Ch. 3
	F		Graph Algorithms	

Complexity Example

- Cryptography (e.g. RSA, SSL in browsers)
  - Secret: p,q prime, say 512 bits each
  - Public: n which equals pxq, 1024 bits
- In principle
  - there is an algorithm that given n will find p and q by trying all 2^{512} possible p's.
- In practice
  - security of RSA depends on the fact that no **efficient** algorithm is known for this

Algorithms versus Machines

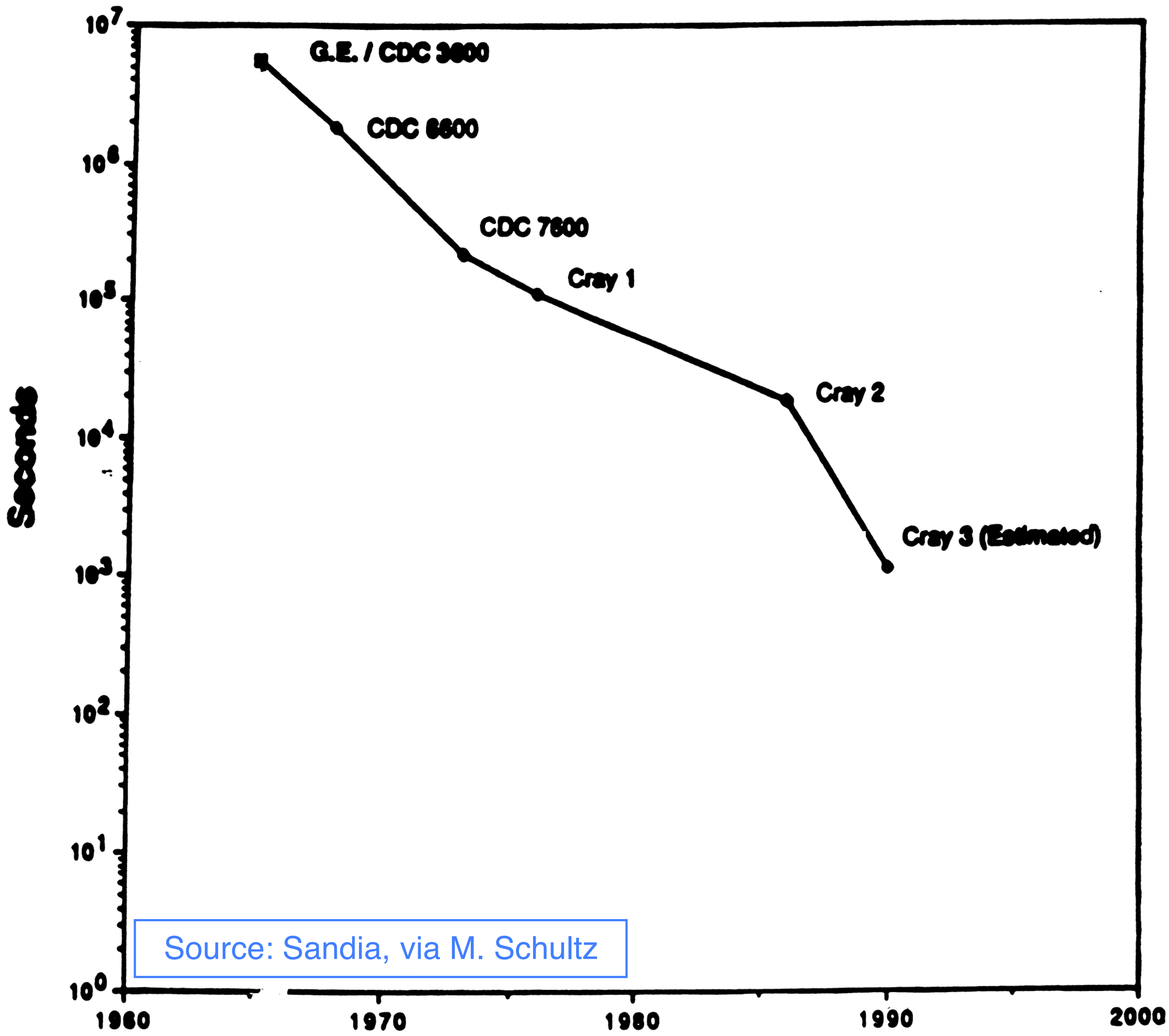
- We all know about Moore's Law and the exponential improvements in hardware but...

- Ex: sparse linear equations over past few decades

- 10 orders of magnitude improvement in speed
  - 4 orders of magnitude improvement in hardware
  - 6 orders of magnitude improvement in algorithms

Algorithms or Hard-ware?

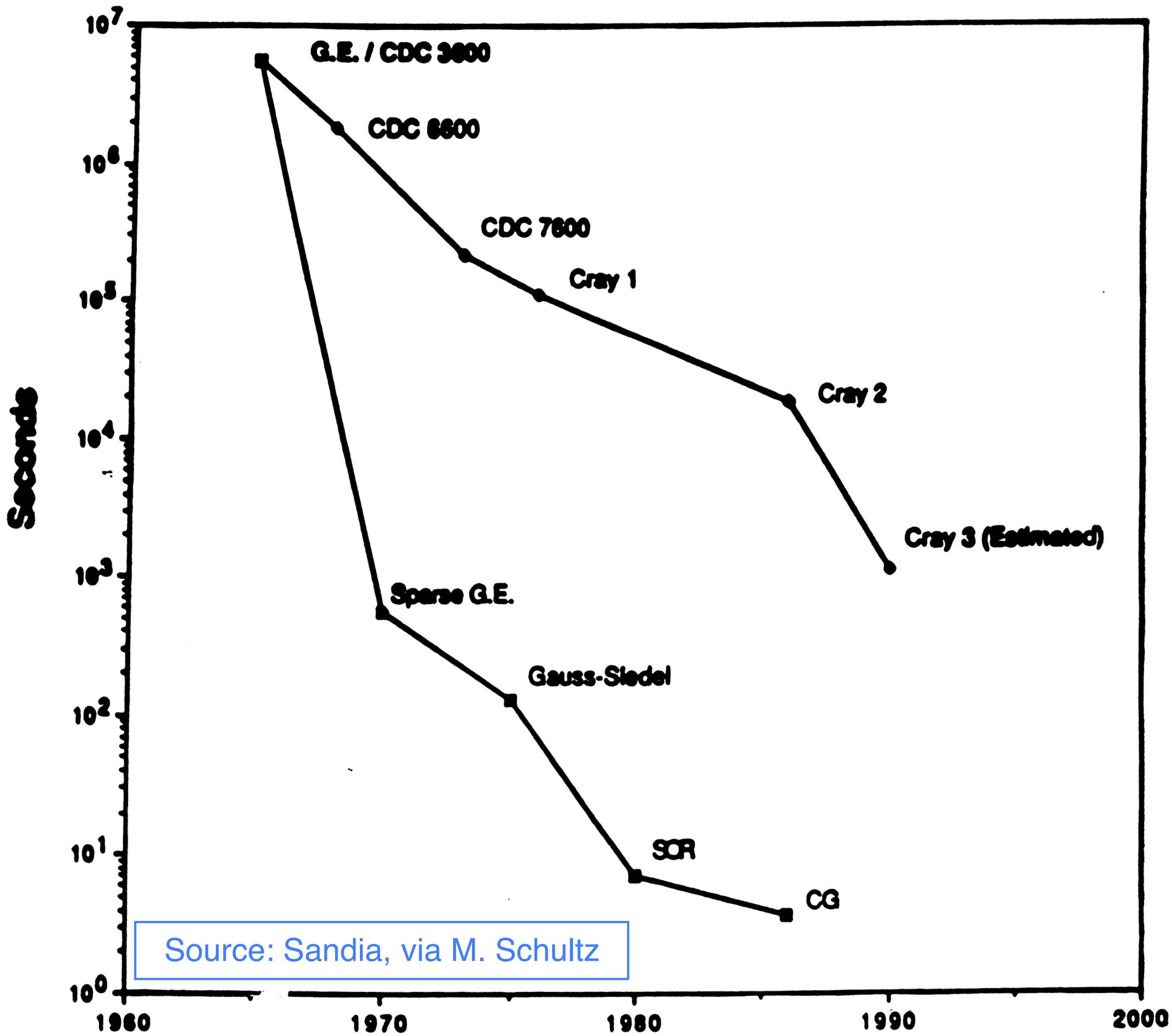
Solving sparse linear systems



Source: Sandia, via M. Schultz

Algorithms or Hard-ware?

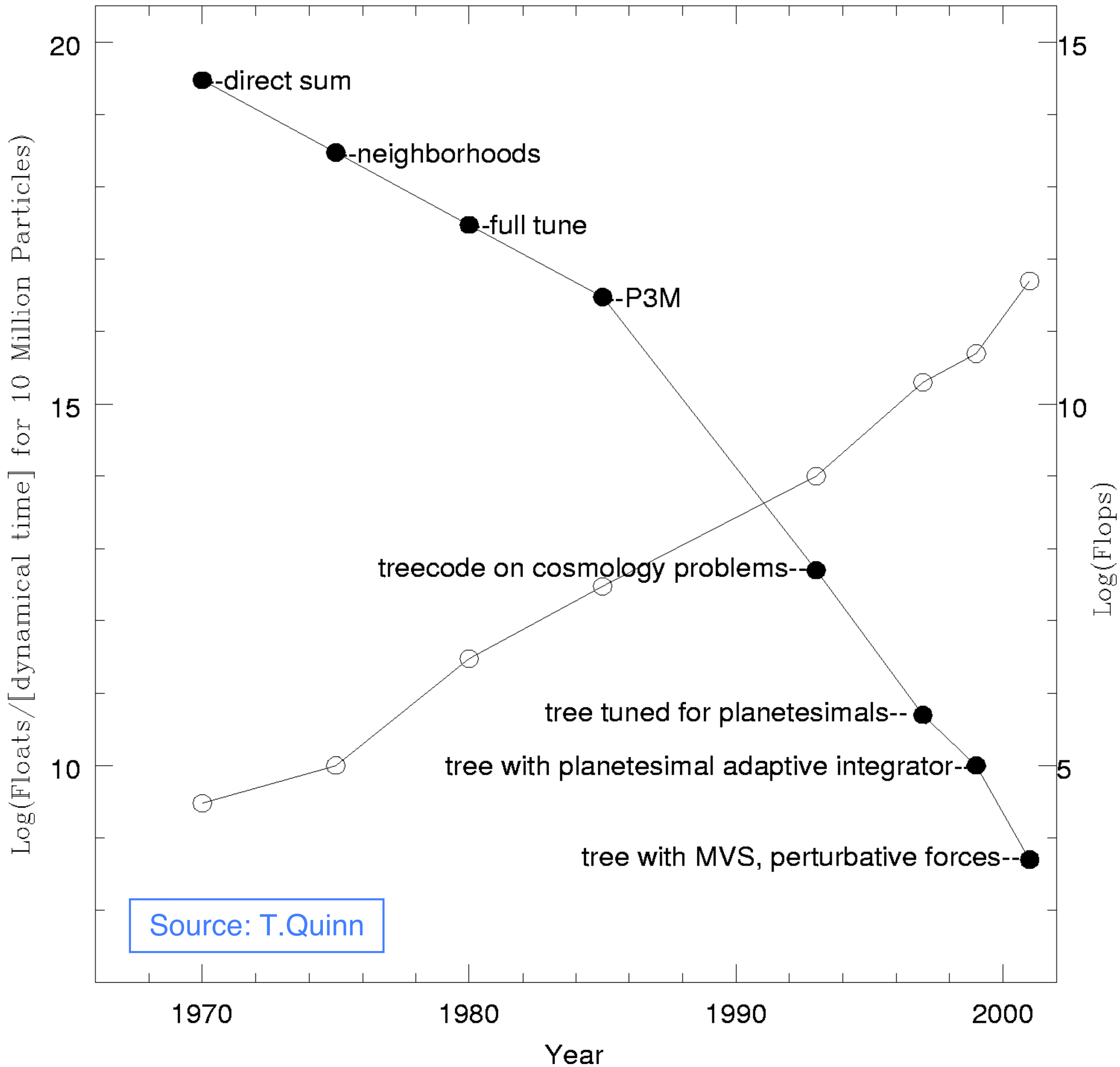
Solving sparse linear systems



Source: Sandia, via M. Schultz

Algo-rithms or Hard-ware?

The N-Body Problem



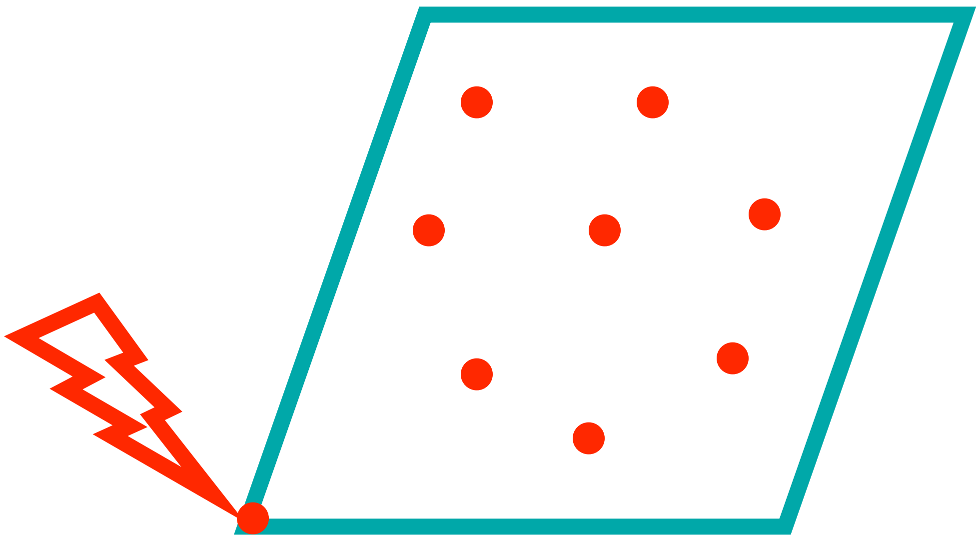
Algorithm: definition

- 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 +, -, *, /, ≤)

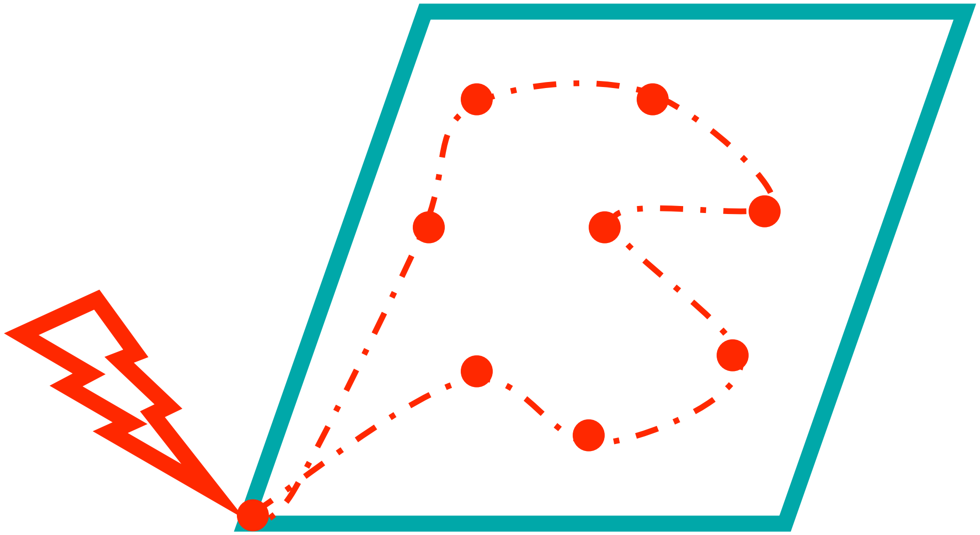
Algorithms: a sample problem

- Printed circuit-board company has a robot arm that solders components to the board
- Time to do it depends on
  - total distance the arm must move from initial rest position around the board and back to the initial positions
- For each board design, must figure out good 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**.

- Better known as "TSP"

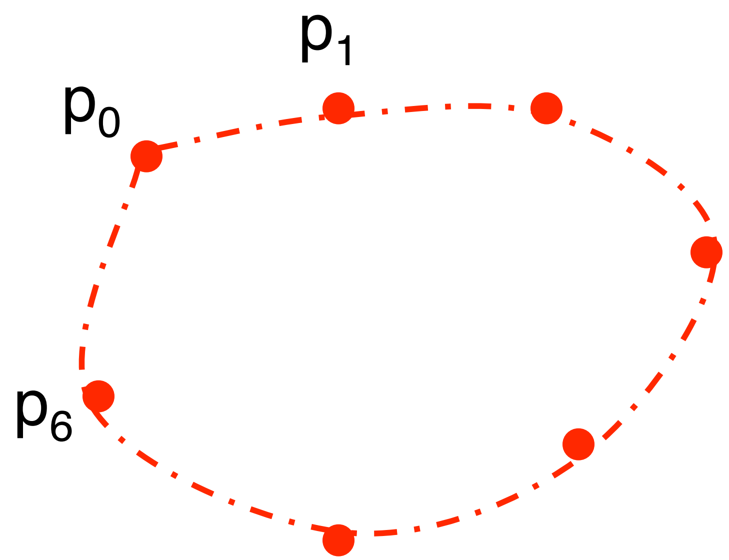
- How might you solve it?

Nearest Neighbor Heuristic

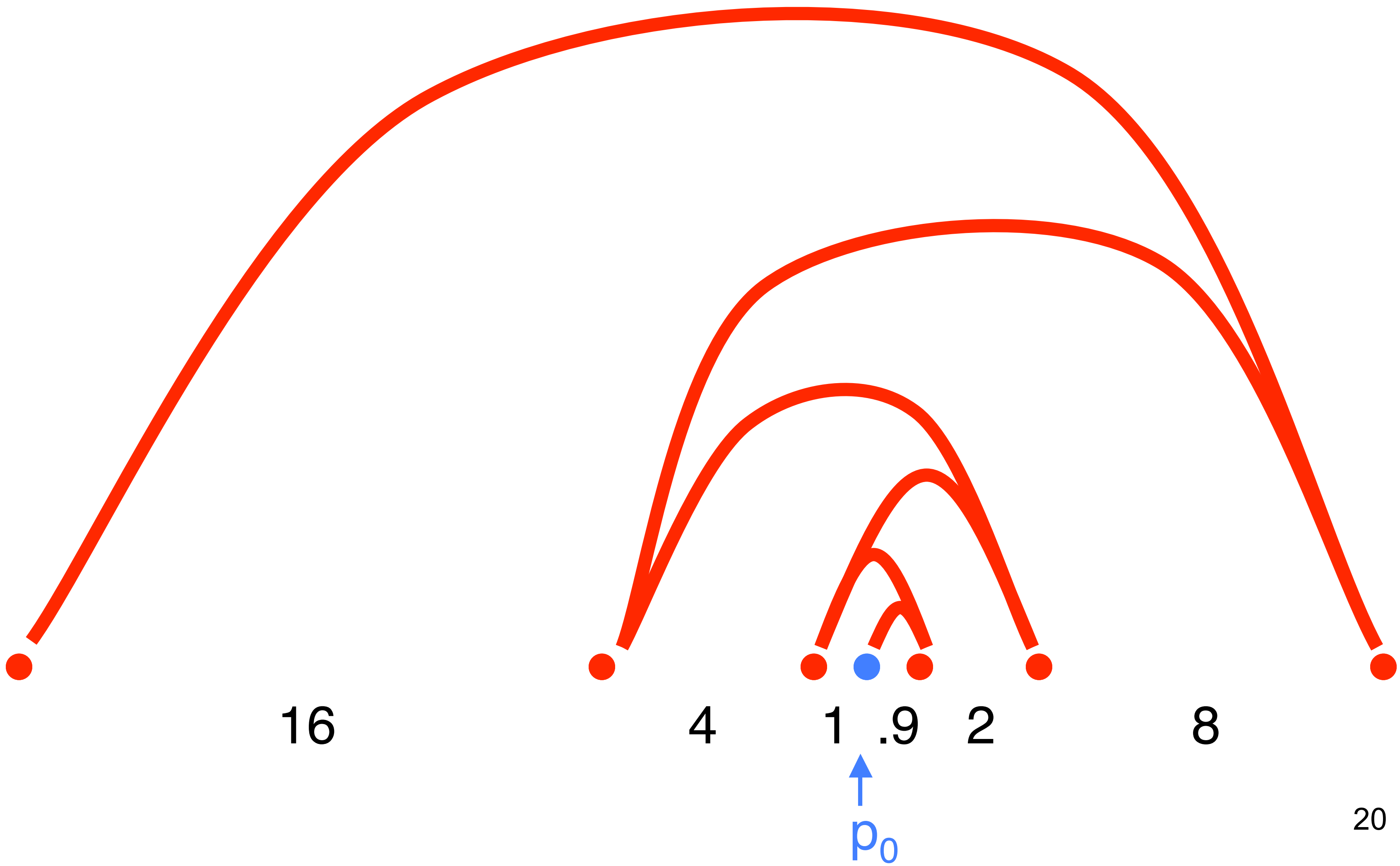
heuristic: A rule of thumb, simplification, or educated guess that reduces or limits the search for solutions in domains that are difficult and poorly understood. Usually ***not*** guaranteed to give the best or fastest solution.

- Start at some point p_0
- Walk first to its nearest neighbor p_1
- Repeatedly walk to the nearest unvisited neighbor until all points have been visited
- Then walk back to p_0

Nearest Neighbor Heuristic



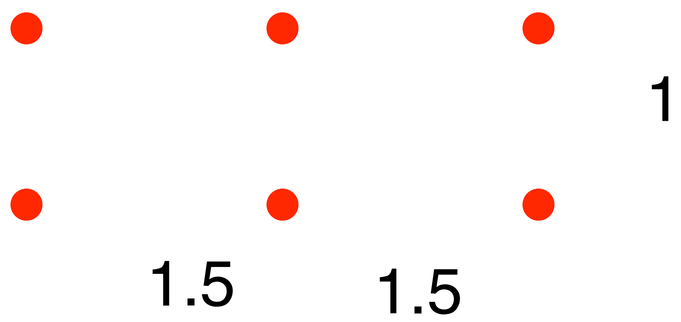
An input where it works badly



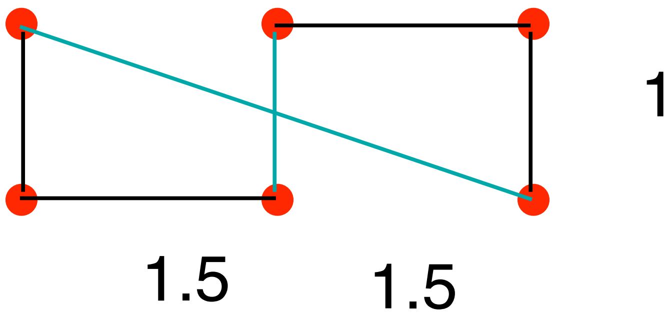
Revised idea - Closest pairs first

- Repeatedly pick the closest pair of points to join so that the result can still be part of a single loop in the end
  - can pick endpoints of line segments already created
- How does this work on our bad example?

Another bad example

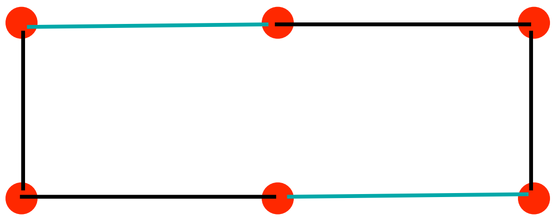


Another bad example



$6+\sqrt{10} = 9.16$

vs



8

Something that works

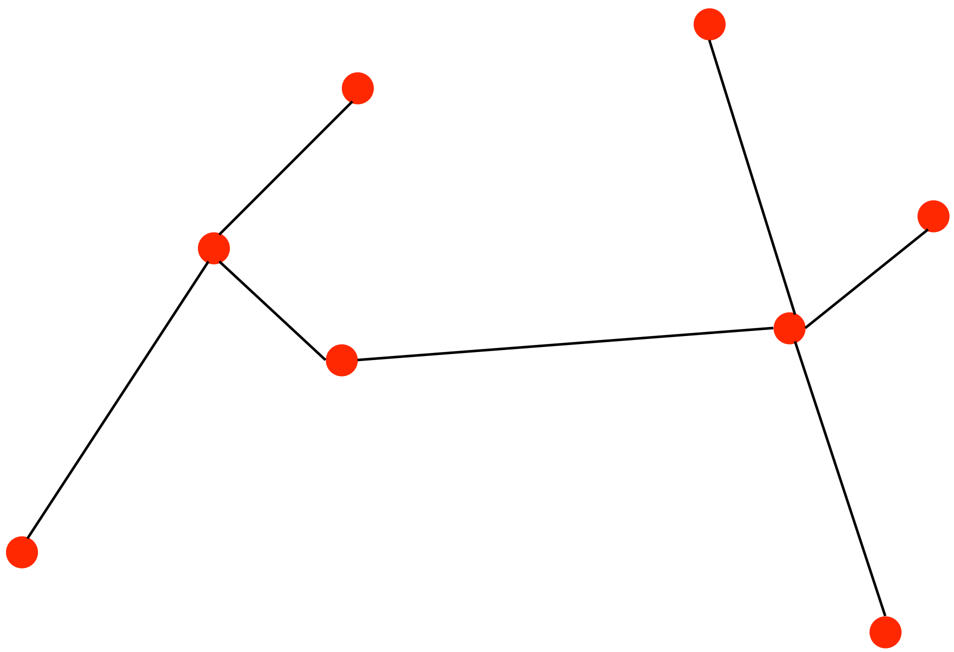
- For each of the $n! = n(n-1)(n-2)\ldots1$ orderings of the points, check the length of the cycle you get
- Keep the best one

Two Notes

- The two incorrect algorithms were **greedy**
  - Often very natural & tempting ideas
  - they make choices that look great “locally” (and never reconsidered them)
  - often does not work - you get boxed in
    - when it works, the algorithms are typically efficient
- Our correct algorithm avoids this, but is incredibly slow
  - 20! is so large that counting to one billion in a second it would still take 2.4 billion seconds
    - (around 70 years!)

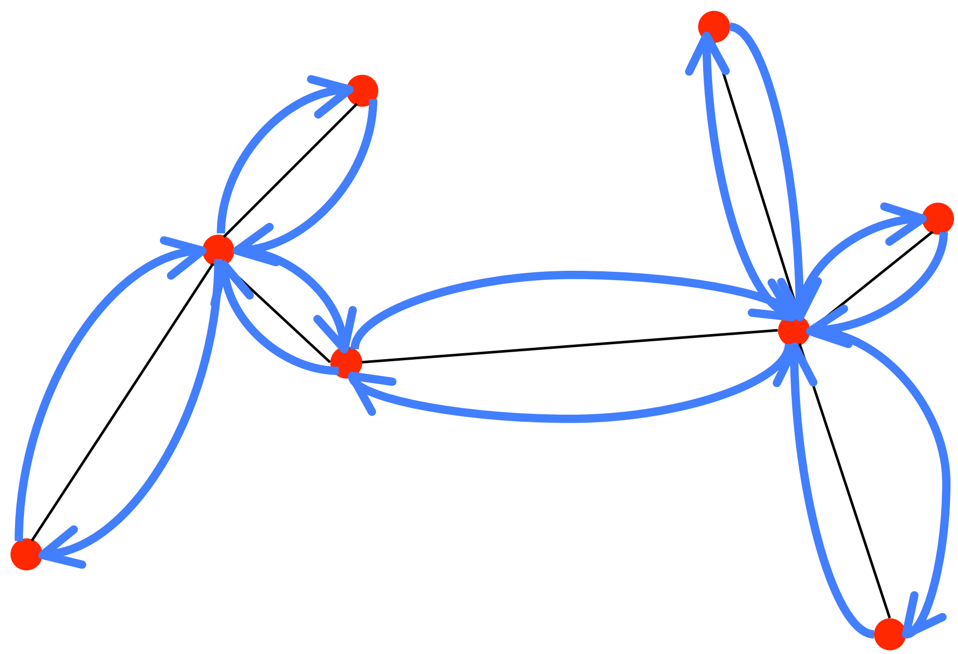
Something that "works" (differently)

1. Find Min Spanning Tree



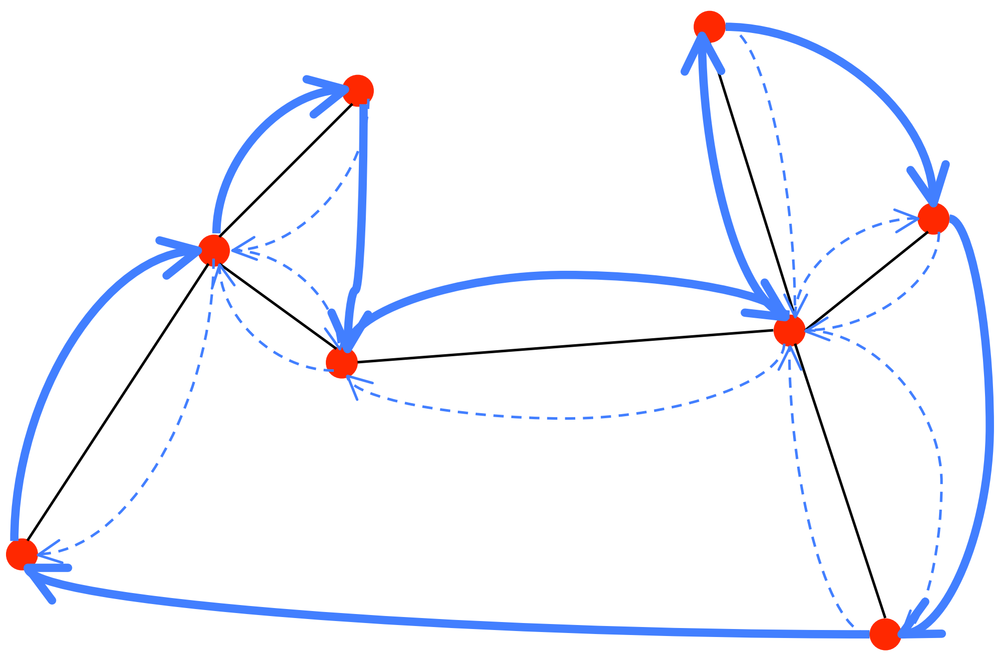
Something that "works" (differently)

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 *Min* spanning tree < best tour
  - best tour ≤ wacky tour ≤ 2 * MST < 2 * 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