CSE 421
Introduction to Algorithms

Overview

Larry Ruzzo

University of Washington

Computer Science & Engineering

CSE 421, Su '12: Introduction to Algorithms

▷ CSE Home

Administrative
- FAQ
- Schedule & Reading

Course Email/BBoard
- Subscription Options
- Class List Archive
- E-mail Course Staff
- GoPost BBoard

Lecture Notes
- 1: Overview & Example
- 2: Analysis

Lecture: LOW 206 (schematic) MW 10:50-12:20

		Office Hours	Location	
Instructor:	Larry Ruzzo, ruzzo@cs	W 1:00- 2:0…		6298
TA:	Tamara Bonaci, tbonaci@uw.edu	M…		

http://www.cs.washington.edu/421

Course Em… … and general interest student/staff Q&A … …, but probably should change their default subscri… … use Catalyst GoPost to discuss homework, etc.

… Techniques for design of efficient algorithms. Methods for showing lower bou… …g, set manipulation, arithmetic, graph problems, pattern matching.

Prerequisites: either CSE 312 or CSE 322; either CSE 326 or CSE 332.

Credits: 3

Grading: Homework, Final. Homework will be a mix of paper & pencil exercises and programing. O…

Late Policy: Unless otherwise announced, weekly homeworks will be due by 4:00PM on Thursdays; … electronically, including scanned versions of handwritten papers, via the Catalyst drop box link at lef…

Extra Credit: Assignments may include "extra credit" sections. These will enrich your understanding … glory, not the points, and don't start extra credit until the basics are complete.

Textbook: *Algorithm Design* by Jon Kleinberg and Eva Tardos. Addison Wesley, 2006. (Available f…

What you have to do

Weekly Homework (~60% of grade)

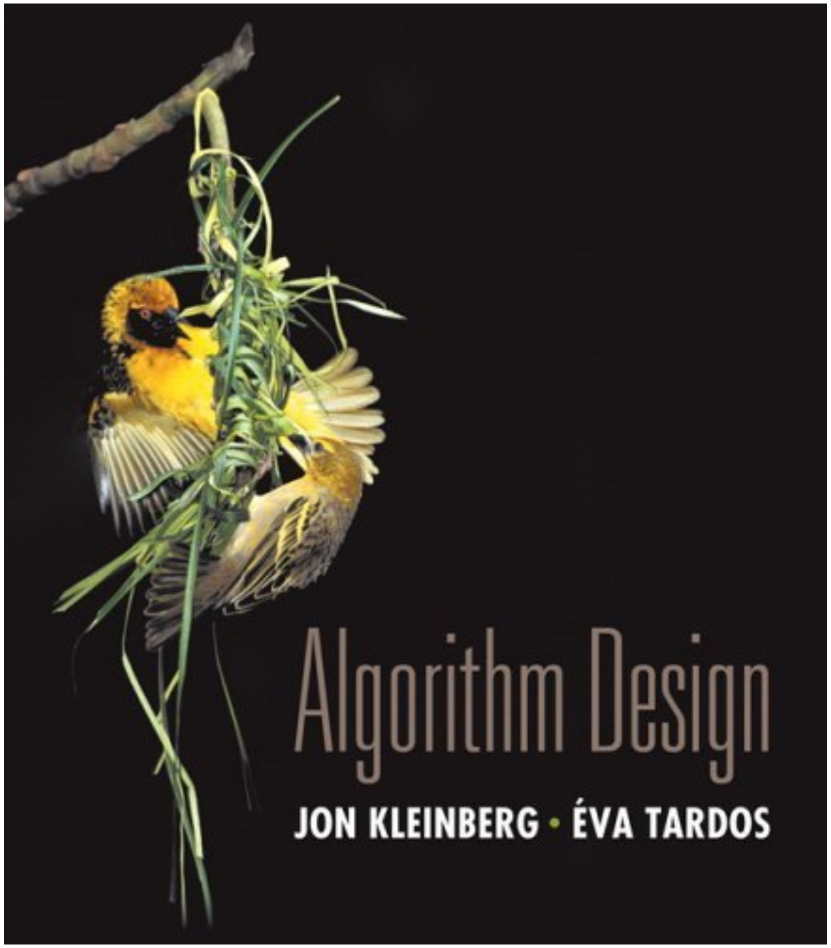
- Programming?
  - perhaps some small projects
- Written homework assignments
  - English exposition and pseudo-code
  - Analysis and argument as well as design

Final Exam (~40%)

Late Policy:

- Papers and/or electronic turnins generally due Thursdays by 4:00 pm; minus 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
- 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: greedy, divide&conquer, dynamic programming,
- Toolkit: e.g., flows & matchings
- Applications

Complexity & NP-completeness (2-3 weeks)

Check online schedule page for (evolving) details

University of Washington
Computer Science & Engineering

CSE 421, 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	

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...

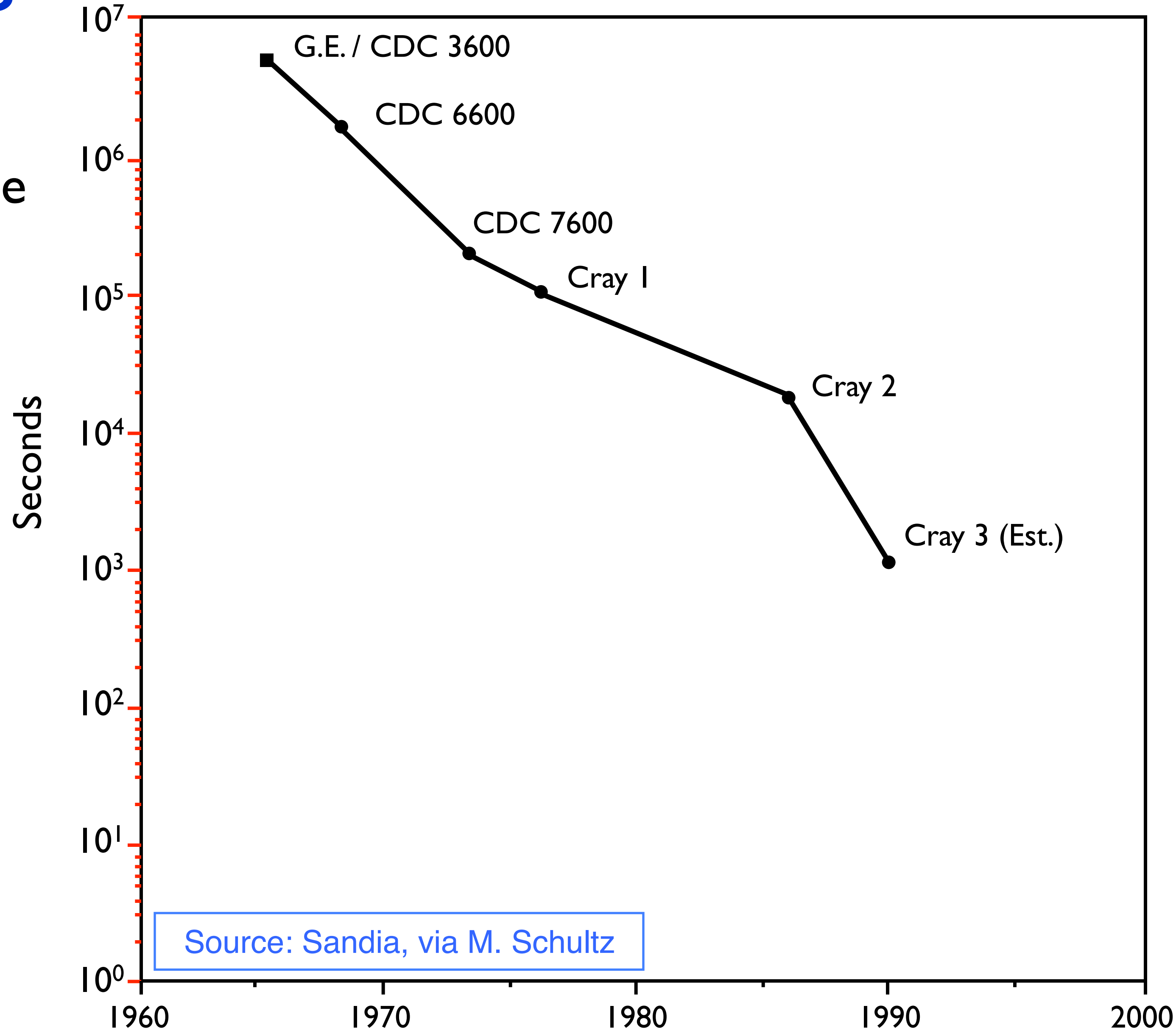
Ex: sparse linear equations over 25 years

10 orders of magnitude improvement!

Algorithms or Hardware?

25 years progress solving sparse linear systems

hardware: 4 orders of magnitude

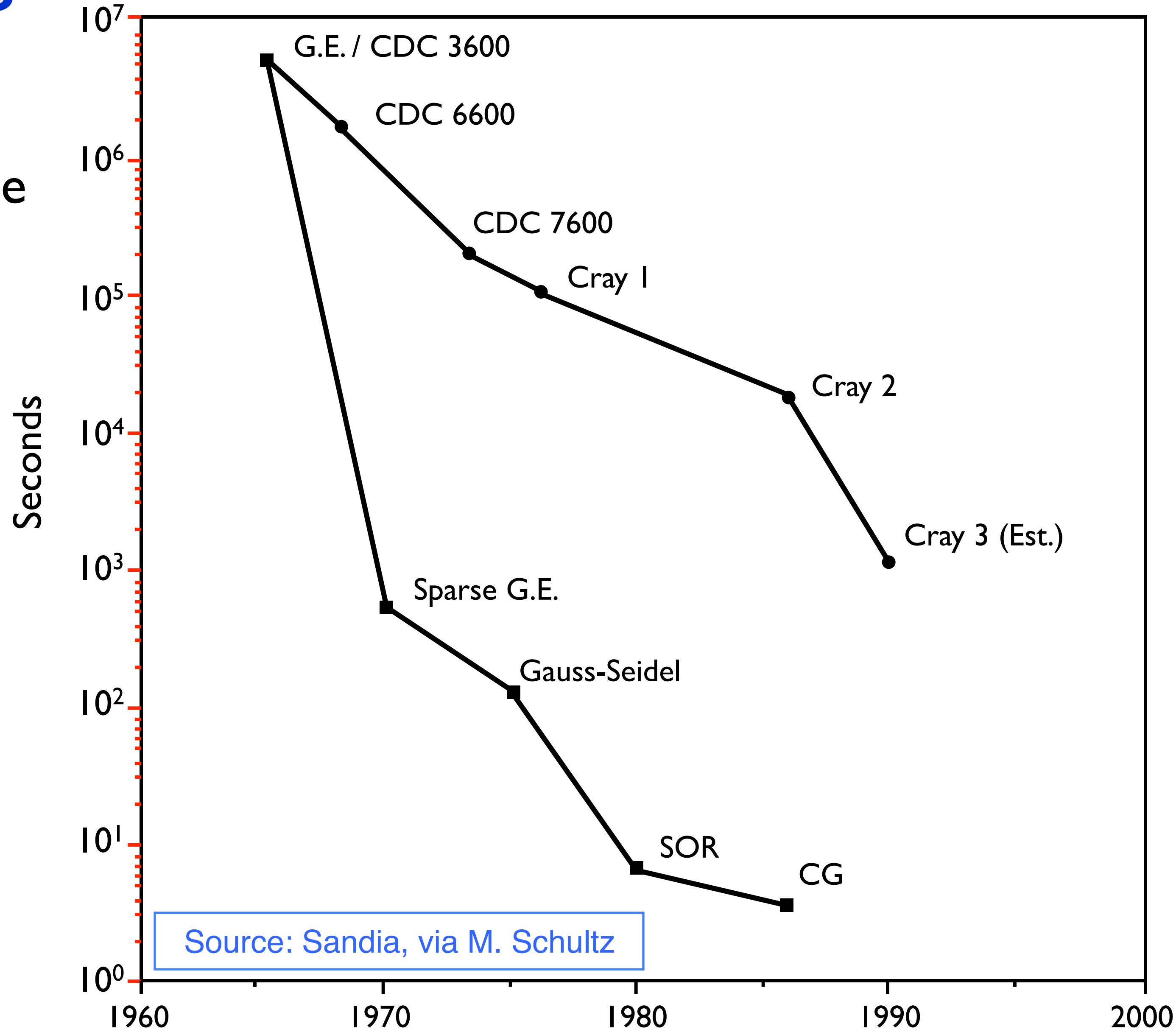


Algorithms or Hardware?

25 years progress solving sparse linear systems

hardware: 4 orders of magnitude

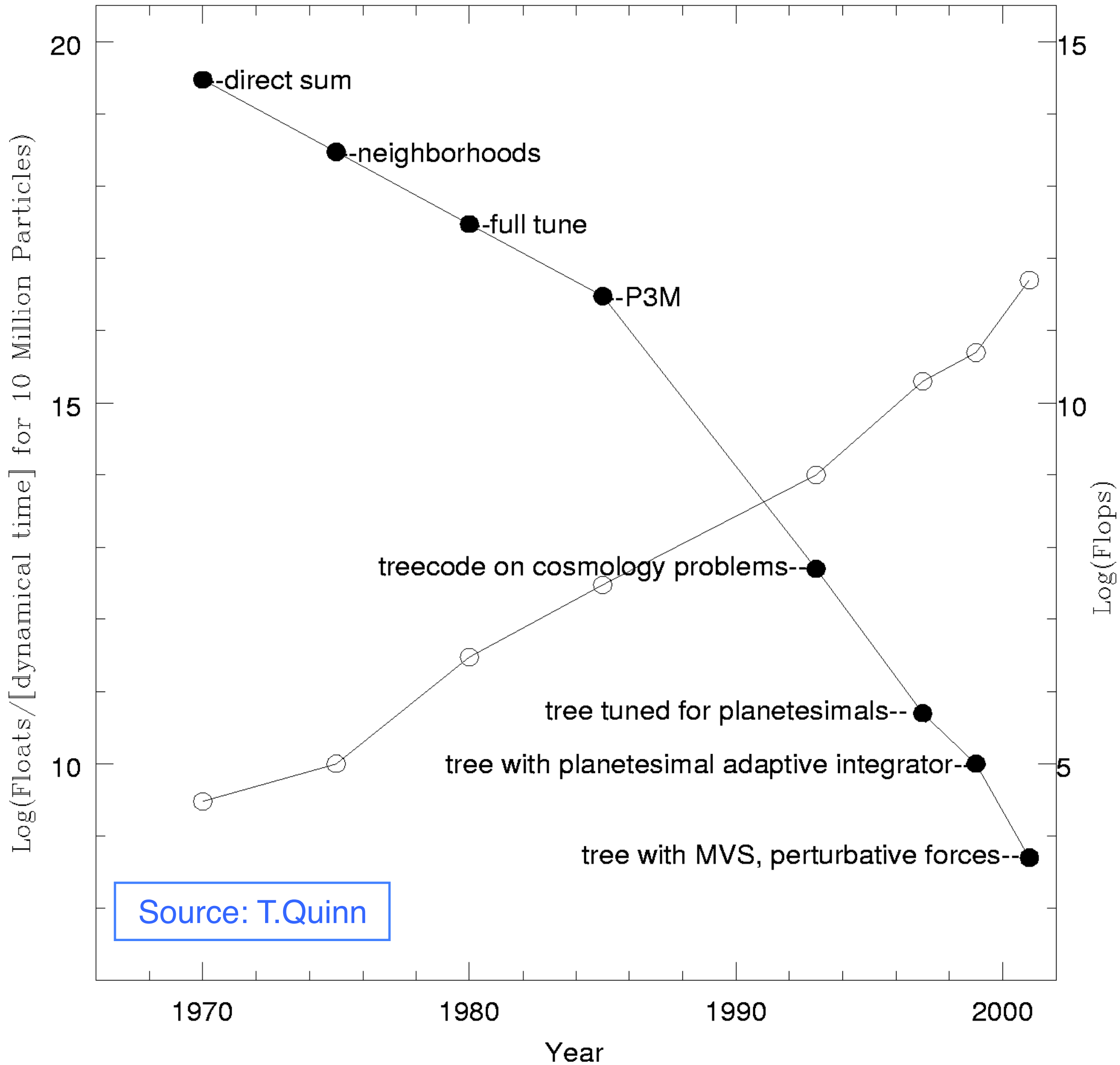
software: 6 orders of magnitude



Algorithms or Hardware?

The
N-Body
Problem:

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



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, …

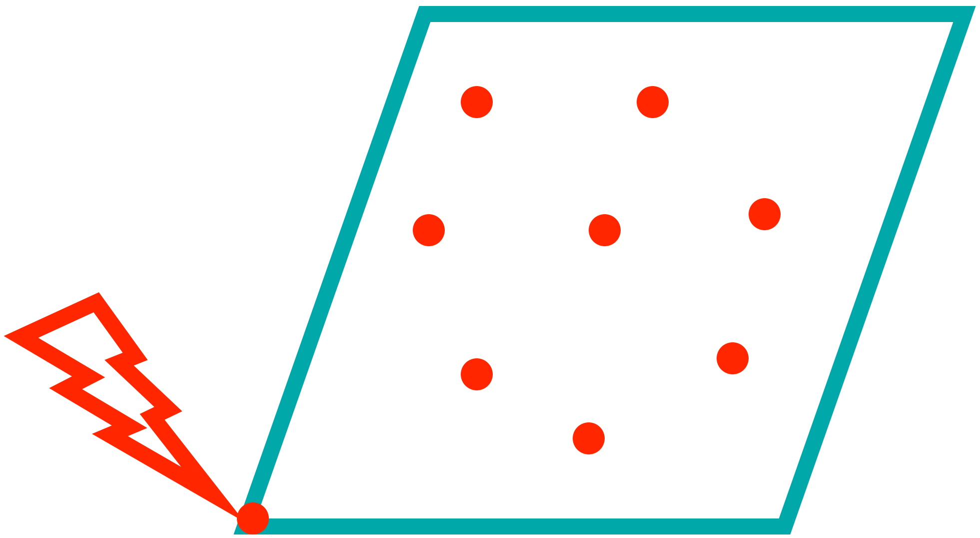
Algorithms: a sample problem

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

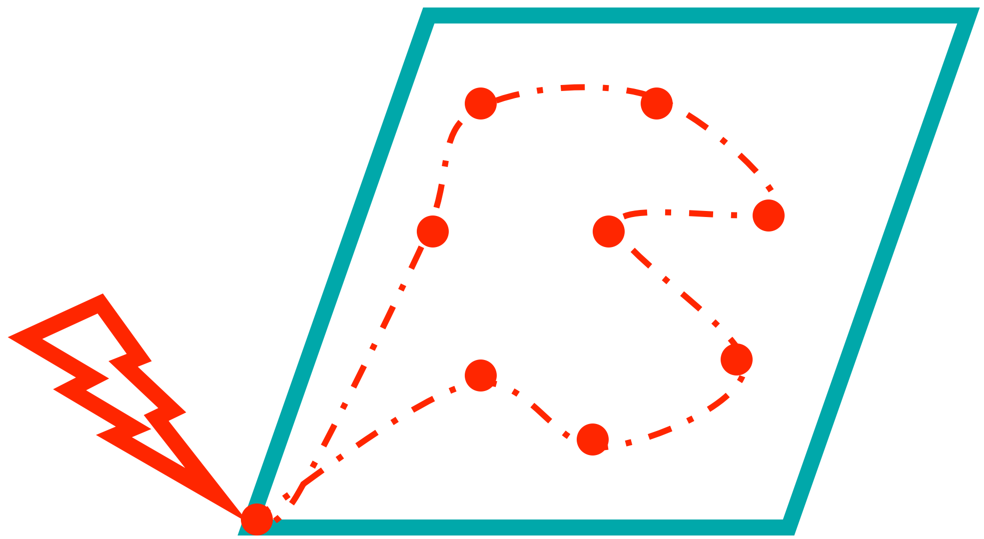
Time: proportional to total distance the arm must move from initial rest position around the board and back to the initial position

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*.

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. May be good, but usually *not* guaranteed to give the best or fastest solution.

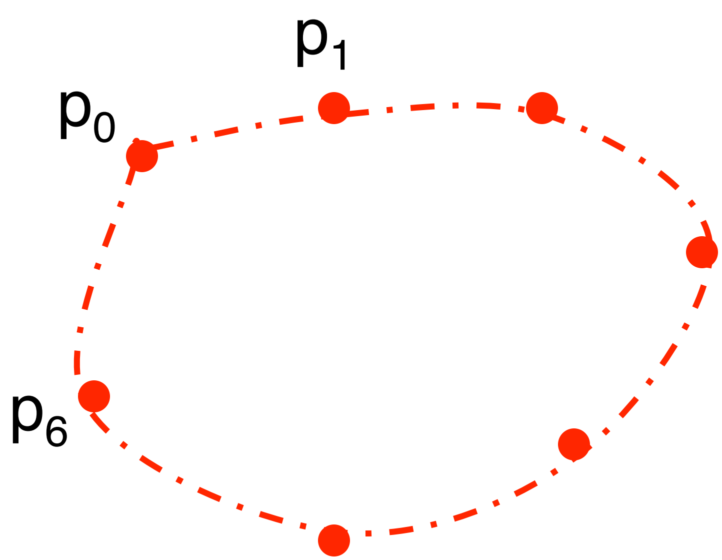
Start at some point p_0

Walk first to its nearest neighbor p_1

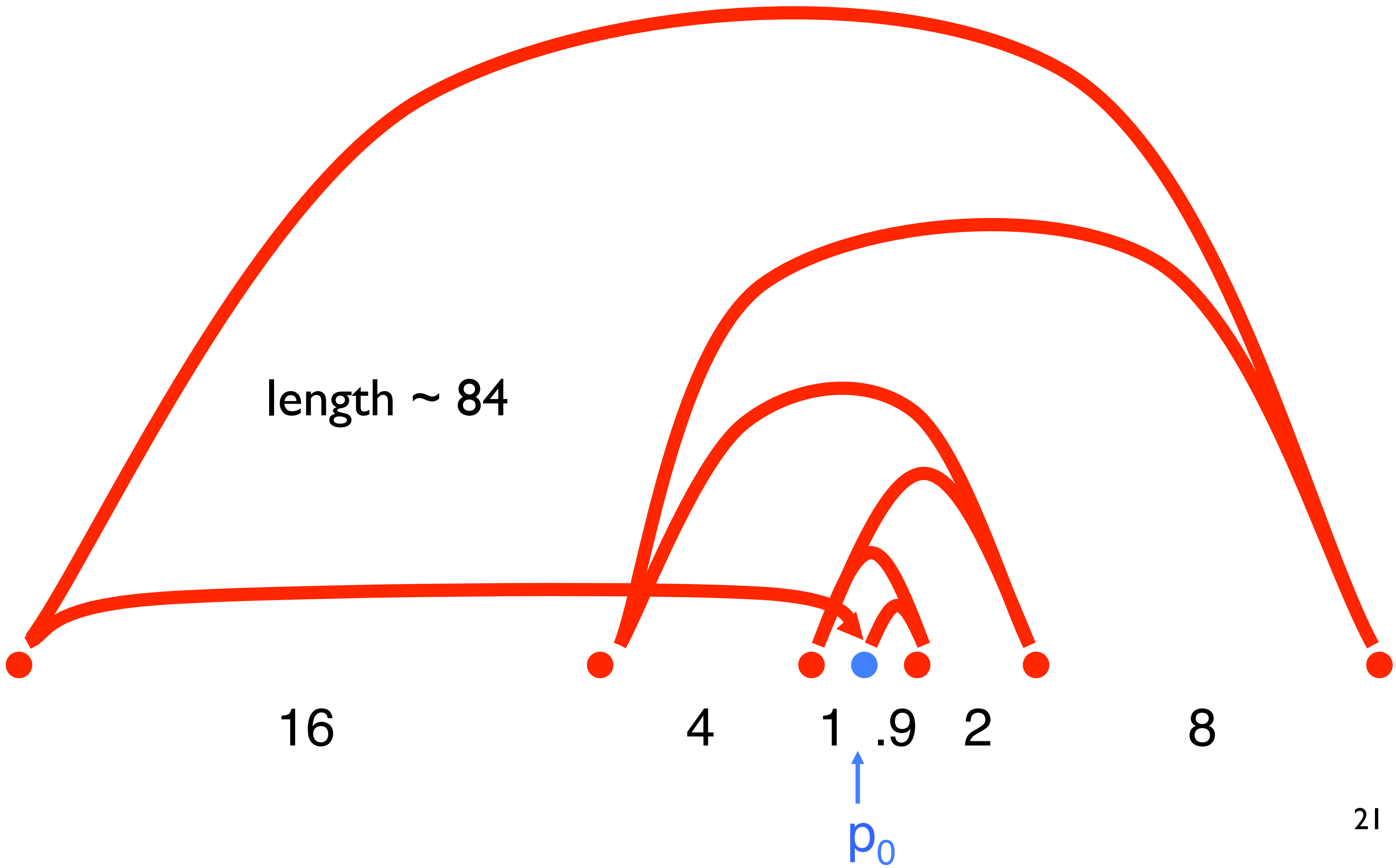
Repeatedly walk to the nearest unvisited neighbor p_2, then p_3,… until all points have been visited

Then walk back to p_0

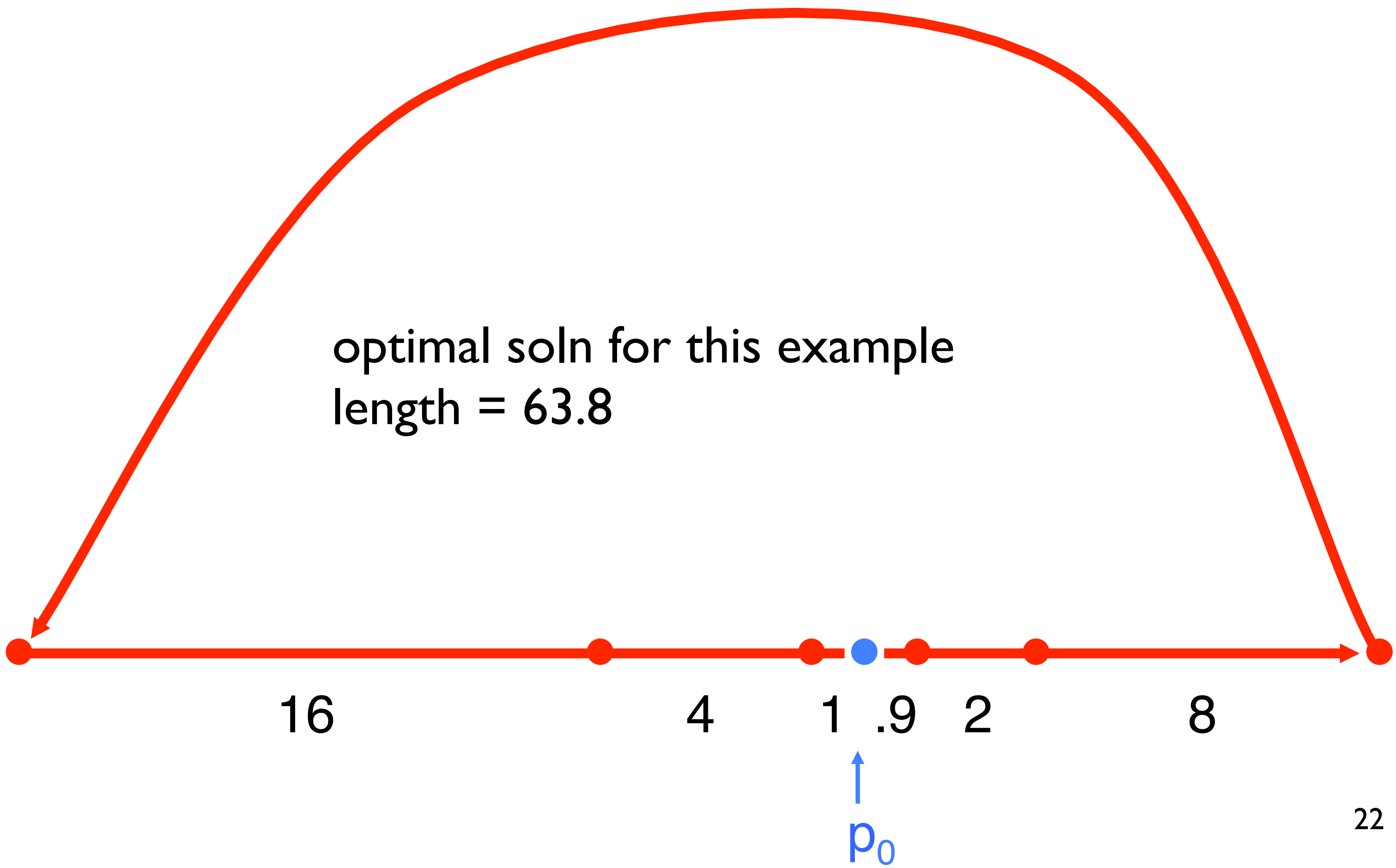
Nearest Neighbor Heuristic



An input where it works poorly



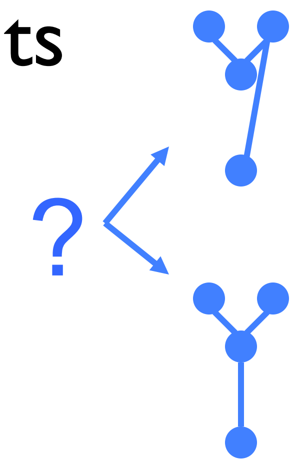
An input where it works poorly



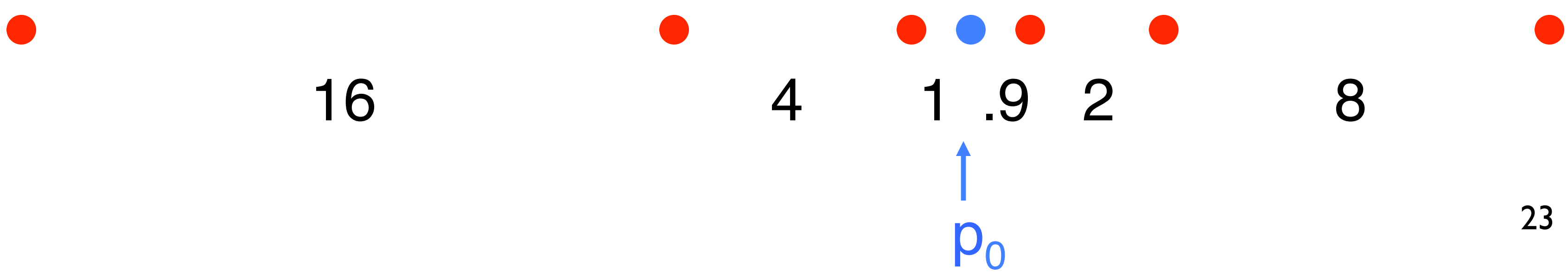
Revised idea - Closest pairs first

Repeatedly join the closest pair of points

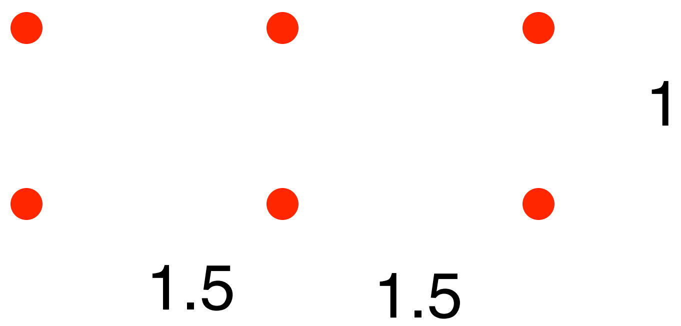
(s.t. result can still be part of a single loop in the end. I.e., join endpoints, but not points in middle, of path segments already created.)



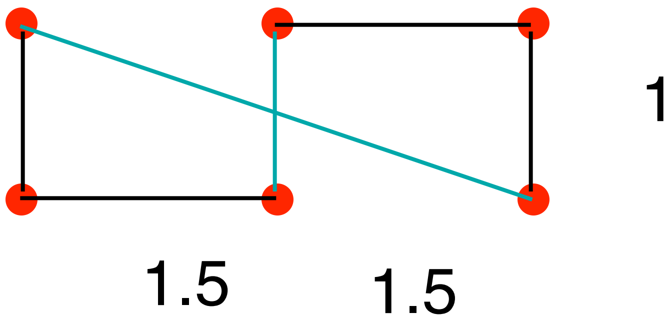
How does this work on our bad example?



Another bad example

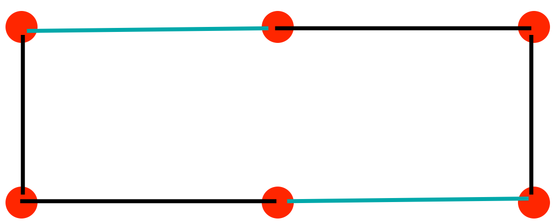


Another bad example



$6+\sqrt{10} = 9.16$

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 *in*correct 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
- And: for some problems, even the *best* algorithms are slow