CSE 421
Introduction to Algorithms

Overview

Larry Ruzzo

University of Washington
Computer Science & Engineering

CSE 421, Su '14: 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 Recordings
1: Help

Lecture: EEB 026 (schematic)   MW 10:50- 12:20

	Office Hours	Location	Phone
Instructor: Larry Ruzzo, ruzzo@cs M	2:00- 3:00	CSE 554	206-543-6298

Course Email: cse421a_su14@uw.edu. Staff announcements and general interest student/staff Q&A about ho~ subscribed to this list. Enrolled students are as well, but probably should change their default subs~

Catalog Description: Techniques for design of efficient algorithms. Methods for ~h sorting, searching, set manipulation, arithmetic, graph problems, pattern~

Prerequisites: either CSE 312 or CSE 322; either CSE 32~

Credits: 3

Grading: Homework, Final ~ programing. Overall weights 60%, 40%,

Late Policy: ~ ~ne due date. (Outside of class, you may turn in work~ e' ~email them.)

**Ext~ ~a credit" sections. These will enrich your understanding of the material, but at a l~ for ~ don't start extra credit until the basics are complete.

**Textb~ ~an Design_ by Jon Kleinberg and Eva Tardos. Addison Wesley, 2006. (Available from U Book Store, Amazo~

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

Portions of the CSE 421 Web may be reprinted or adapted for academic nonprofit purposes, providing the source is accurately quoted and duly credited. The C~
and the Department of Computer Science and Engineering, University of Washington.

Computer Science & Engineering
University of Washington
Box 352350
Seattle, WA 98195-2350
(206) 543-1695 voice, (206) 543-2969 FAX

2

What you have to do

Weekly Homework   (~60% of grade)

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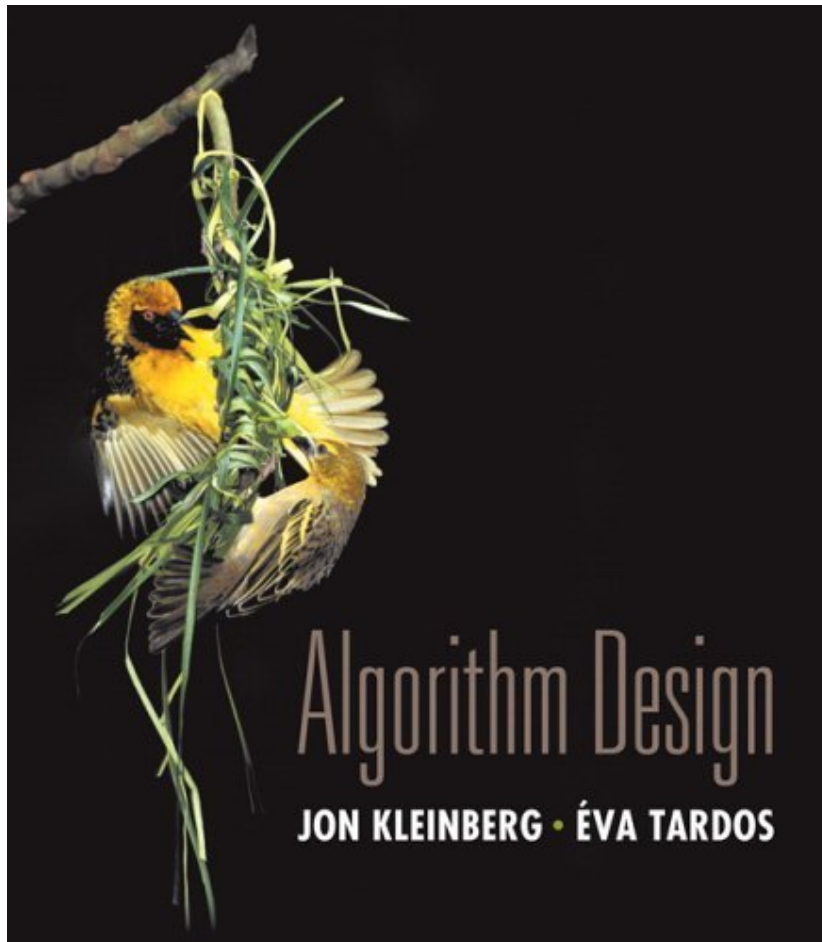   - Analysis and argument as well as design

Final Exam   (~40%)

Late Policy:

Papers and/or electronic turnins generally due Fridays by 5:00 pm; minus 20% per calendar 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 ⊳ S

		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

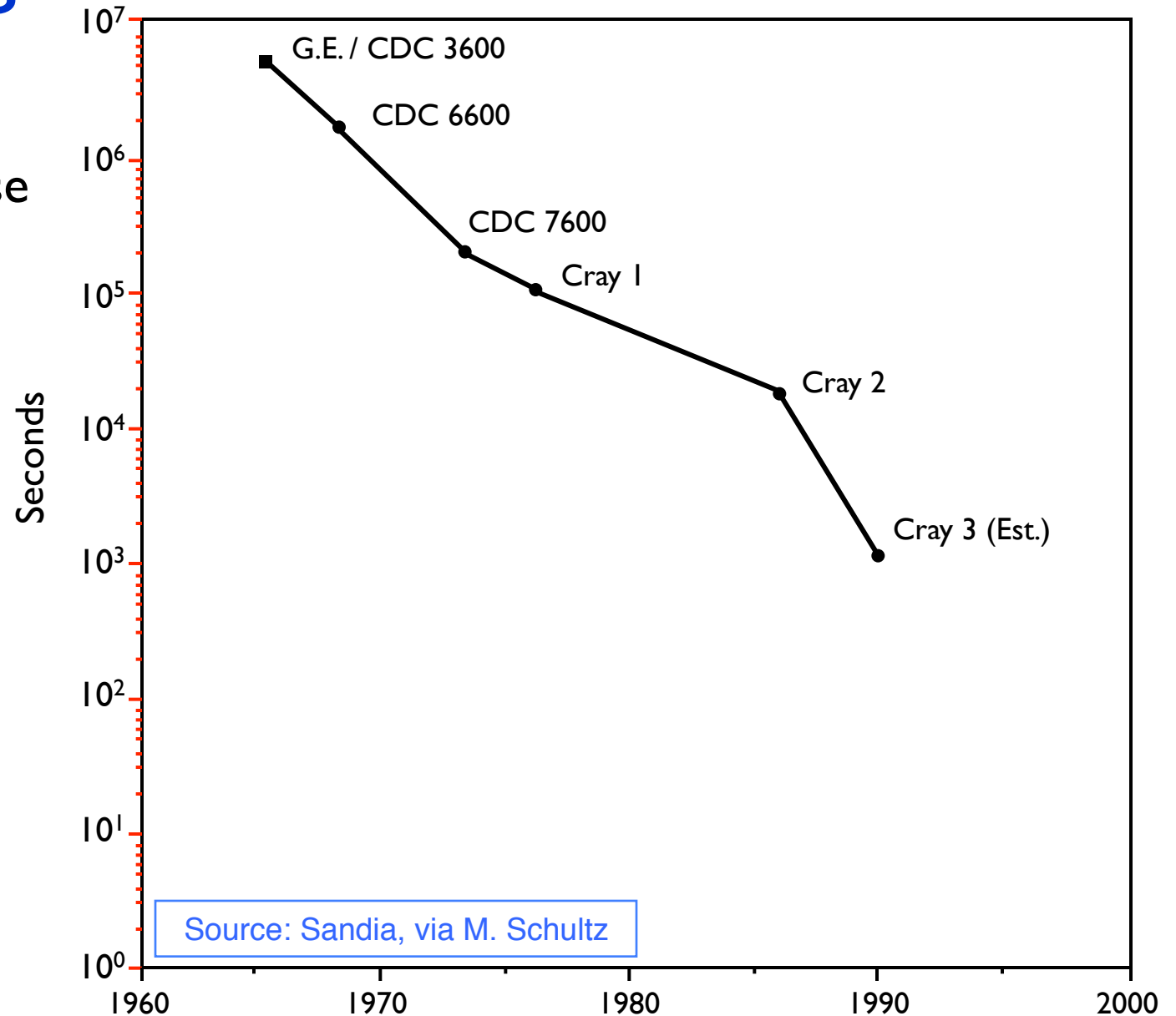We all know about Moore's Law and the exponential improvements in hardware...

Ex: sparse linear equations over 25 years

10 orders of magnitude improvement!

Algorithms or Hardware?

25 years
progress
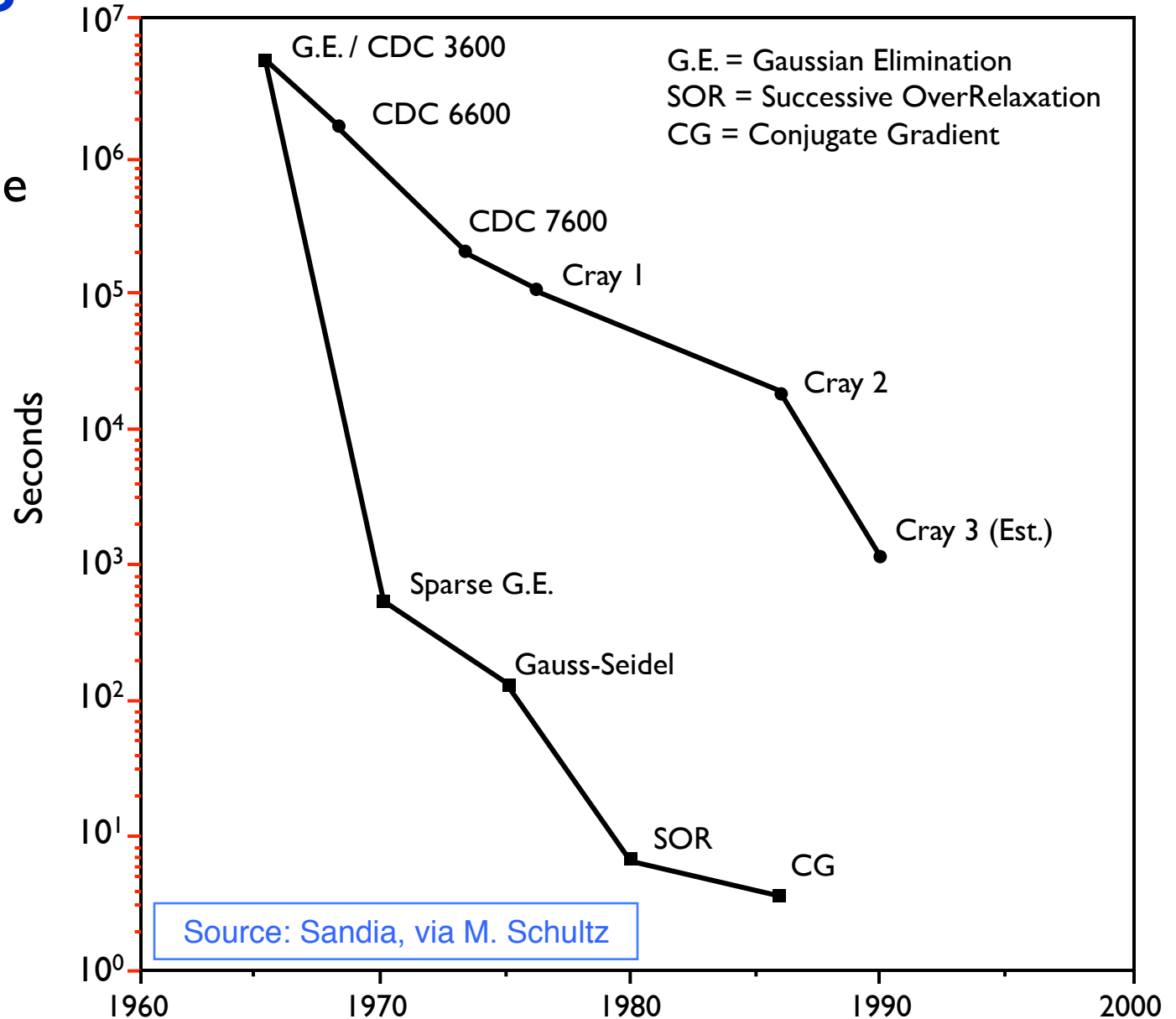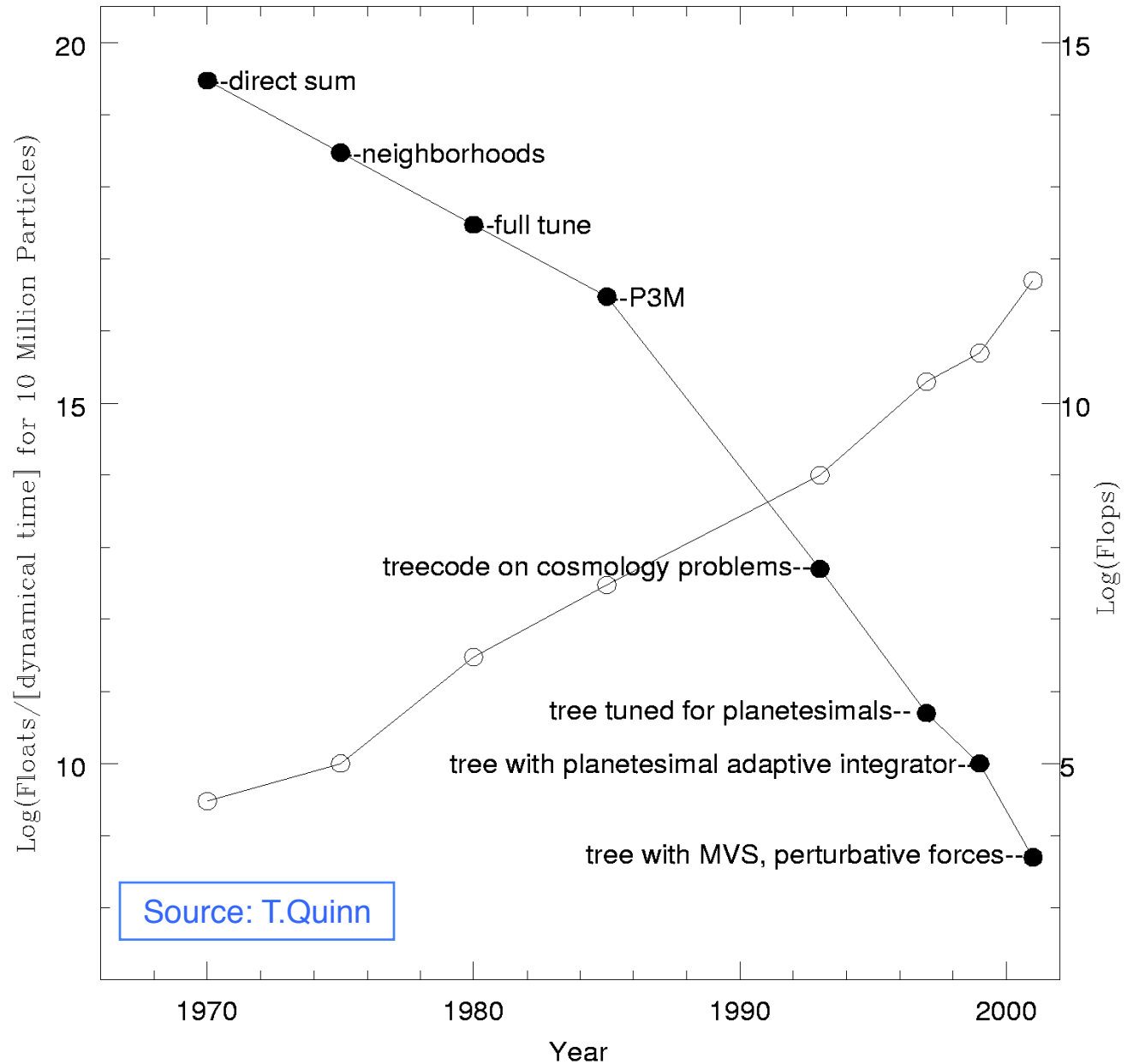solving sparse
linear
systems

hardware:
4 orders of
magnitude

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

G.E. / CDC 3600

CDC 6600

CDC 7600

Cray 1

Cray 2

Cray 3 (Est.)

G.E. = Gaussian Elimination
SOR = Successive OverRelaxation
CG = Conjugate Gradient

Sparse G.E.

Gauss-Seidel

SOR

CG

Seconds

10^7
10^6
10^5
10^4
10^3
10^2
10^1
10^0

1960   1970   1980   1990   2000

Source: Sandia, via M. Schultz

Algorithms or Hardware?

The
N-Body
Problem:

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

- direct sum
- neighborhoods
- full tune
- P3M
- treecode on cosmology problems--
- tree tuned for planetesimals--
- tree with planetesimal adaptive integrator--
- tree with MVS, perturbative forces--

Log(Floats/[dynamical time] for 10 Million Particles)

Log(Flops)

Year

Source: T.Quinn

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
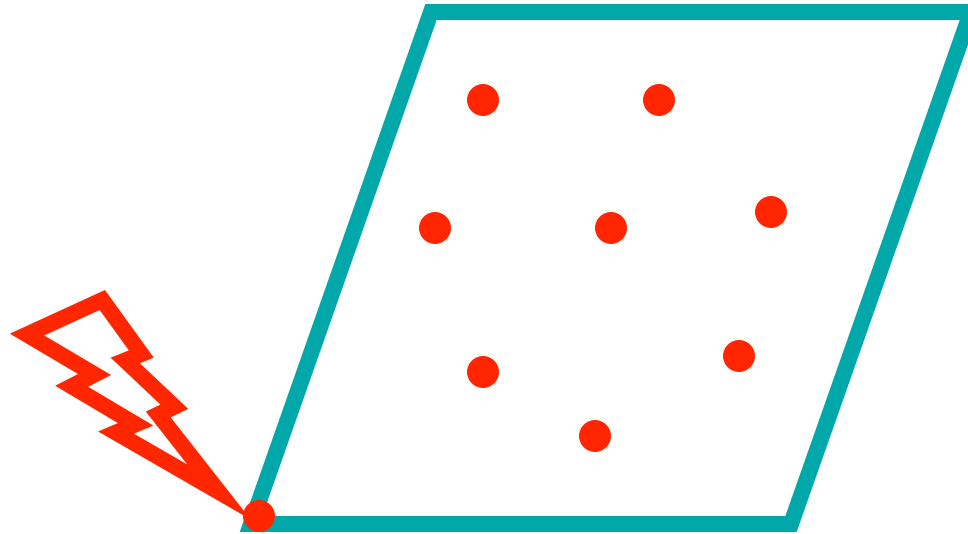
Printed Circuit Board
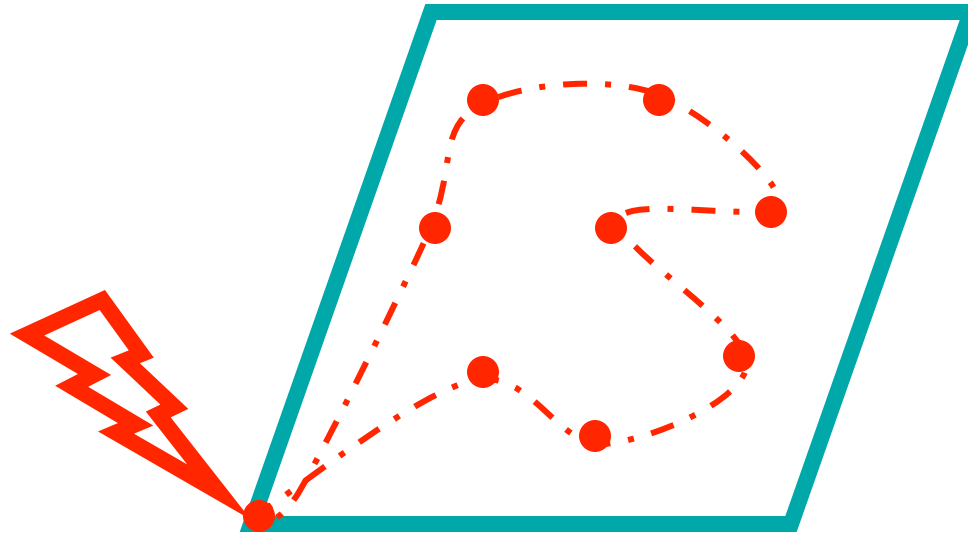
A Well-defined Problem

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"
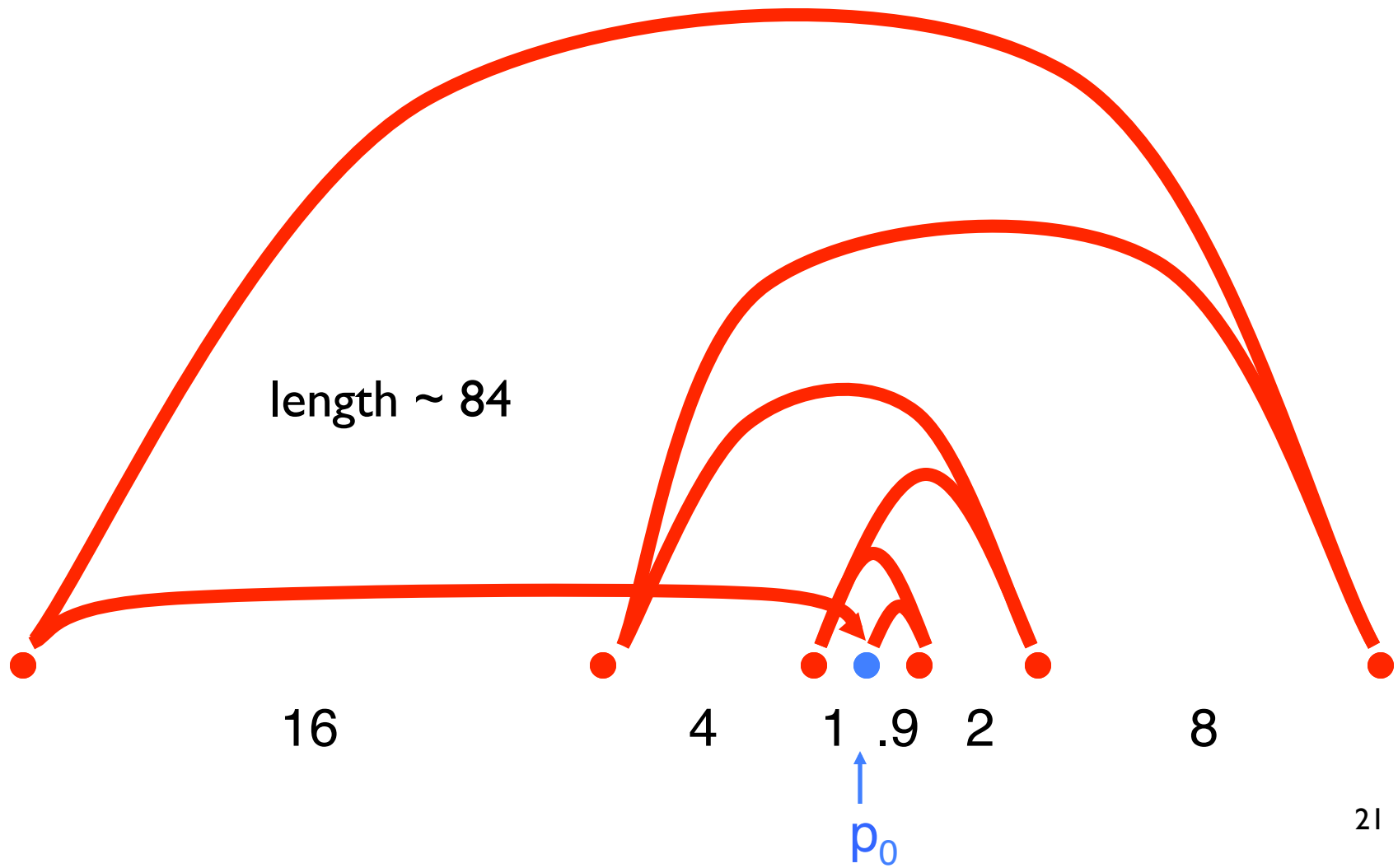
How might you solve it?

Nearest Neighbor Heuristic

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.
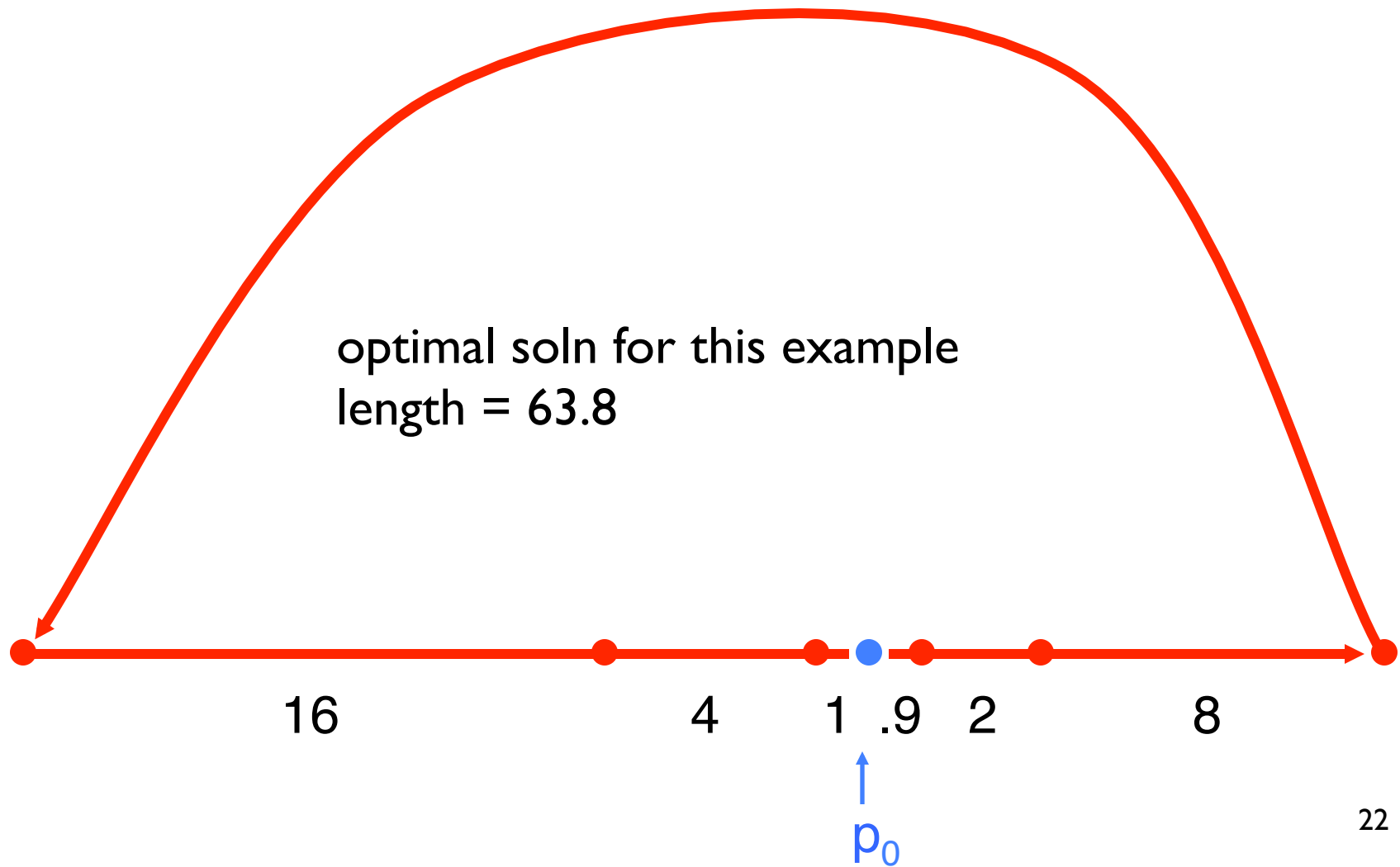
Start at some point p_0

Walk first to its nearest neighbor p_1

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

Then walk back to p_0

Nearest Neighbor Heuristic

An input where NN works badly

length ~ 84

16   4   1 .9 2   8

p_0

An input where NN works badly

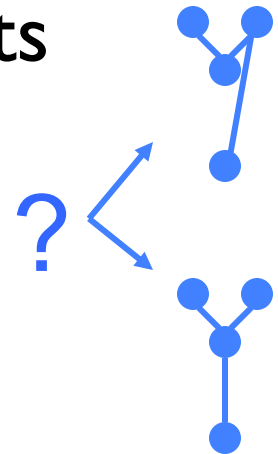optimal soln for this example
length = 63.8

16   4   1 .9 2   8
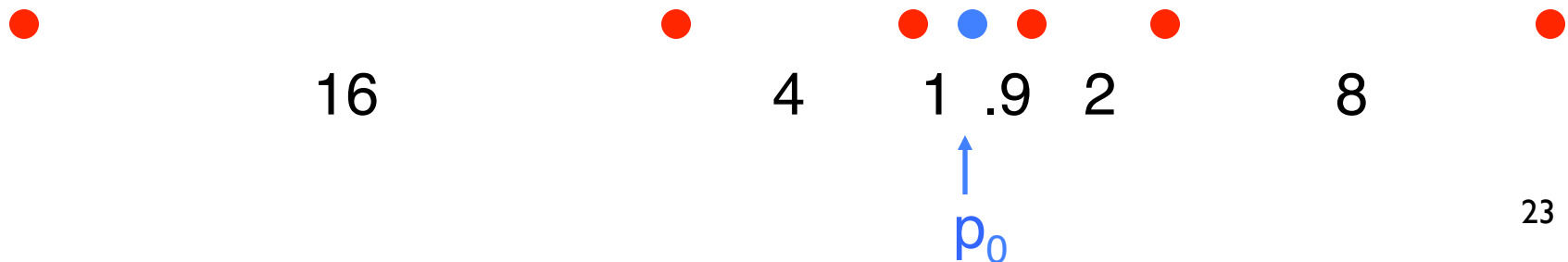
p_0
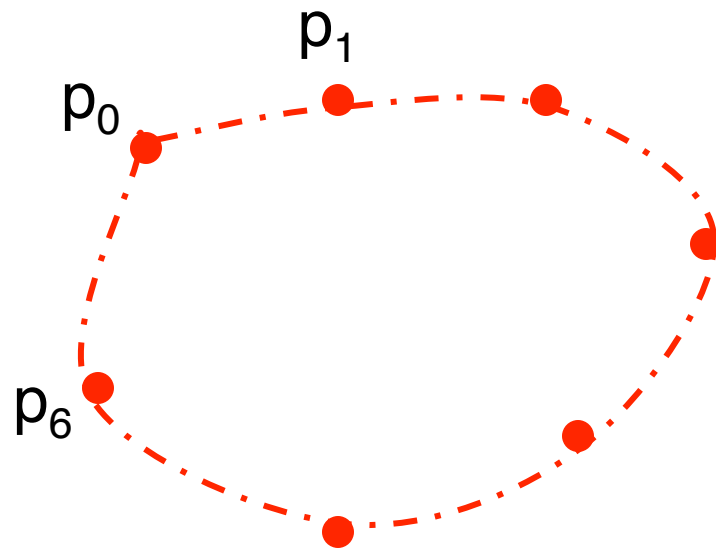
Revised idea - Closest pairs first

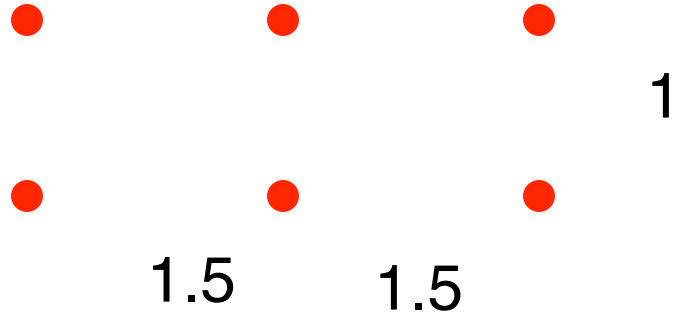Repeatedly join the closest pair of points

(s.t. result can still be part of a
single loop in the end. I.e., join
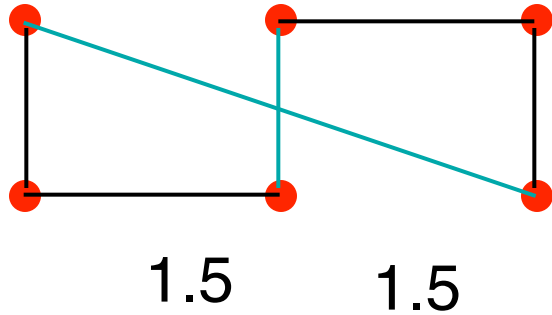endpoints, but not points in middle,
of path segments already created.)
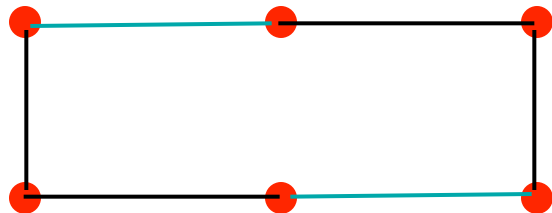
How does this work on our bad example?

16   4   1 .9 2   8

p_0

Another bad example

1

1.5   1.5

Another bad example

1

1.5   1.5

6+√10 = 9.16

vs

8

Something that works

"Brute Force Search":

For each of the n! = n(n-1)(n-2)…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

For some problems, even the *best* algorithms are slow

Course Goals:

   formalize these ideas, and

   develop more sophisticated approaches