CSE 421
Introduction to Algorithms

Overview

Larry Ruzzo

University of Washington
Computer Science & Engineering

CSE 421, Wi '15: Introduction to Algorithms

Administrative
FAQ
Schedule & Reading

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

Lecture Notes
1: **Overview & Example**

Lecture Recordings
See '14 Summer web.

Lecture: MGH 241 (schematic)   MWF 1:30- 2:20

		Office Hours		Location	Phone
Instructor:	Larry Ruzzo, ruzzo@cs	XX	2:30- 3:30?	CSE 554	206-543-6298
TA:	Katie Doroschak, kdorosch@cs	W	2:30- 3:30?	CSE 2xx	
	Elaine Levey, elevey@cs	Th	3:00- 4:00?	CSE 2xx	
	Mert Saglam, saglam@cs	XX	99:00-99:00?	CSE 2xx	

Course Email: cse421a_wi15@uw.edu. Staff announcements and general interest student/staff ..... es, etc. The instructor and TA are subscribed to this list. Enrolled students are as well, but ..... subscription options. Messages are automatically archived.

Catalog Description: Techniques for design of effici..... ..... ng lower bounds on computational complexity. Particular algorithms for sorting, searchi..... .....aph problems, pattern matching.

Prerequisites: either CSE 31..... ..... or CSE 332.

Credits: 3

C..... ....mal. Homework will be a mix of paper & pencil exercises and programing. Overall weights 55%, 15%, 30%,

.....licy: Unless otherwise announced, homework is due at the start of class on the due date. 20% off per day thereafter (business day, e.g., Monday for Friday due dates).

Extra Credit: Assignments may include "extra credit" sections. These will enrich your understanding of the material, but at a low points per hour ratio. Do them for the 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 from U Book Store, Amazon, etc.)

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

What you have to do

Weekly Homework   (~55% of grade)

Programming?

perhaps some small projects

Written homework assignments

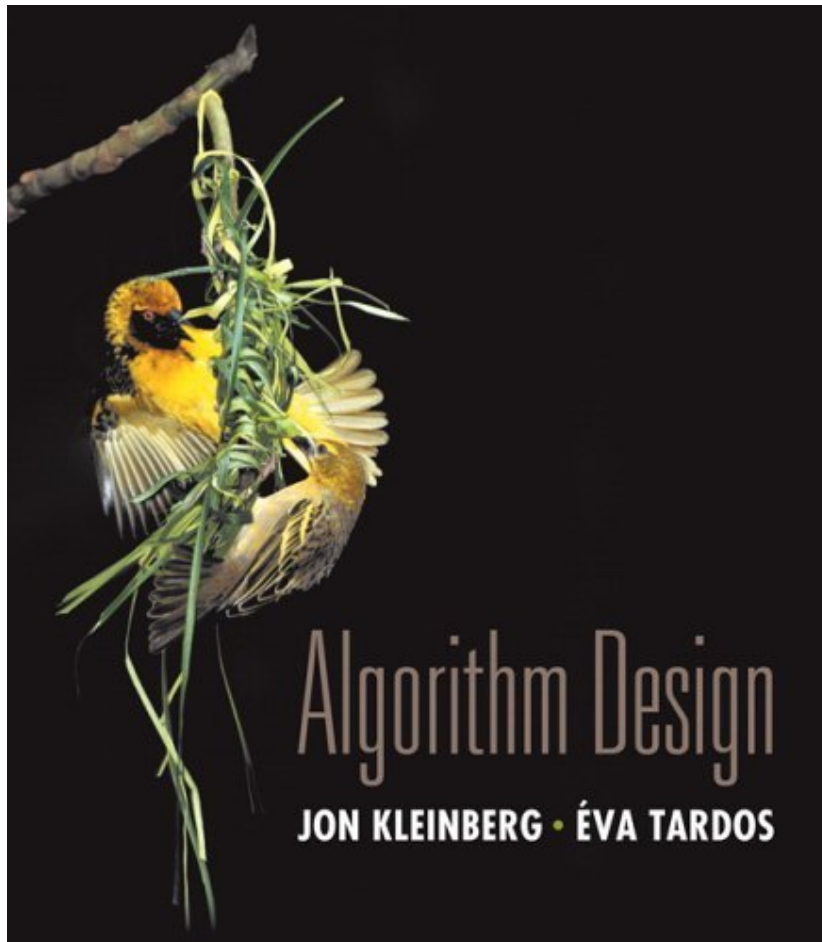English exposition and pseudo-code

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; 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:

        graph search,greedy, divide&conquer, dynamic programming, reduction

    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

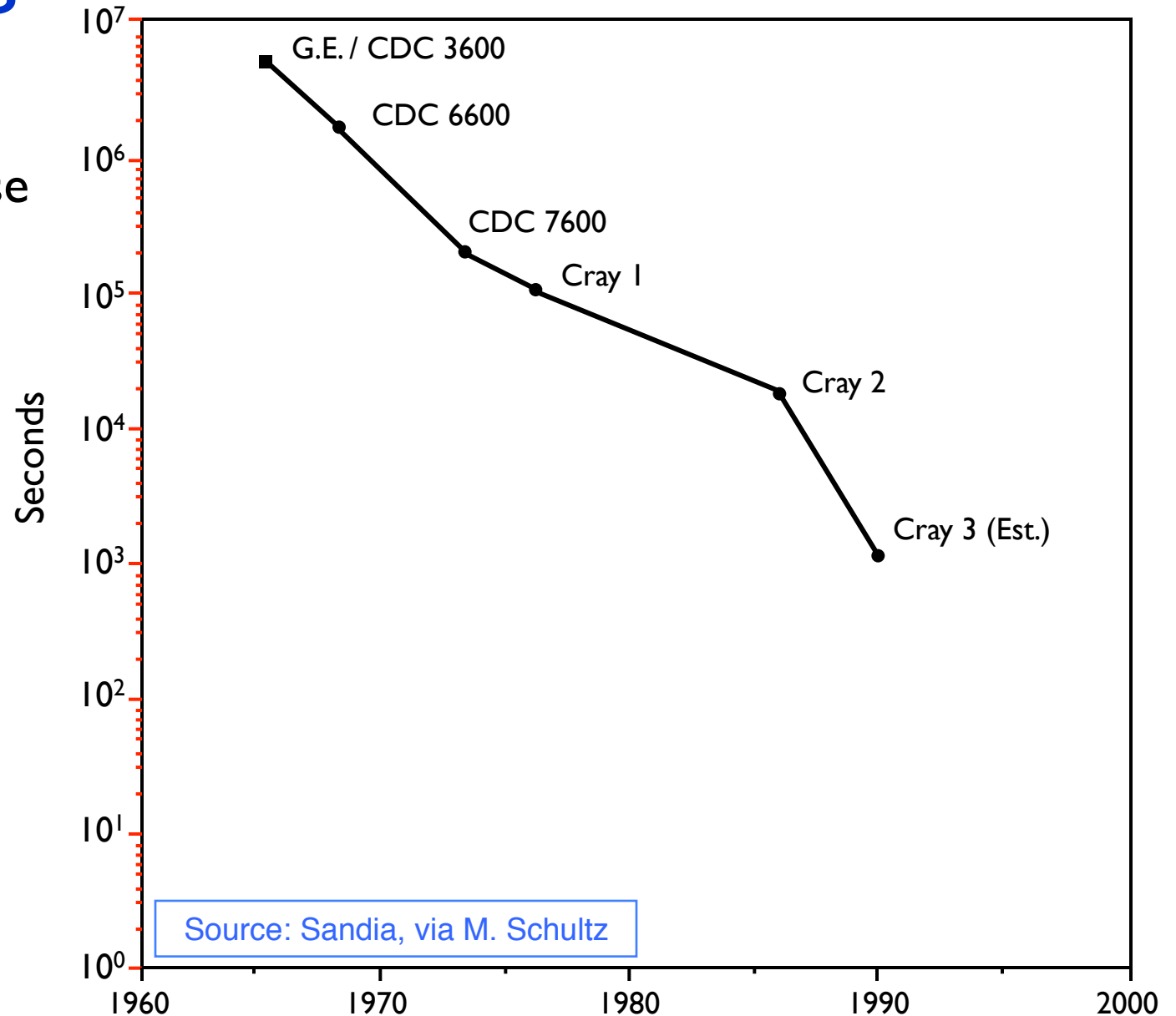We all know about Moore's Law and the exponential improvements in hardware...

Ex: sparse linear equations over 25 years

10 orders of magnitude improvement!

Algorithms or Hardware?

25 years
progress
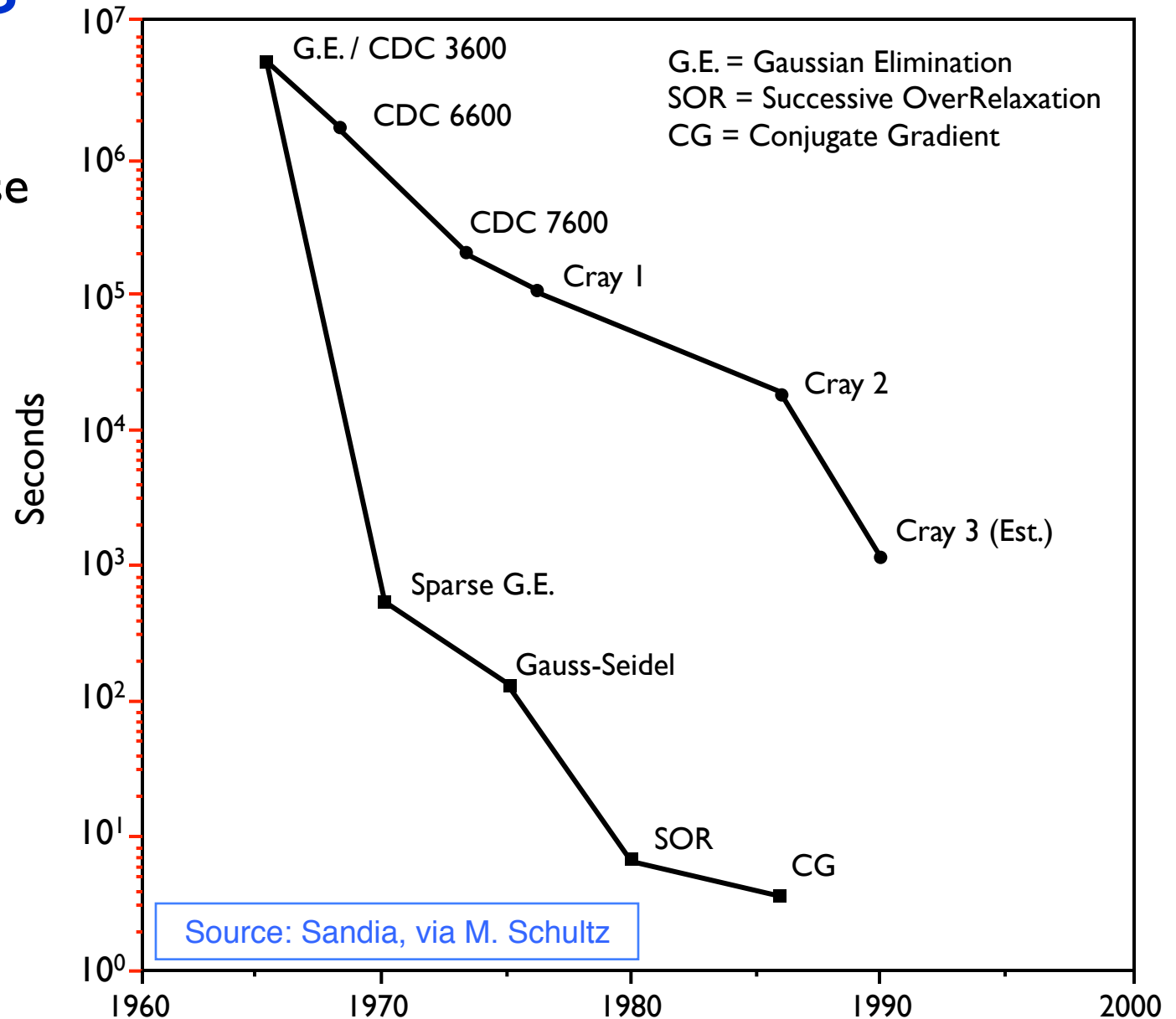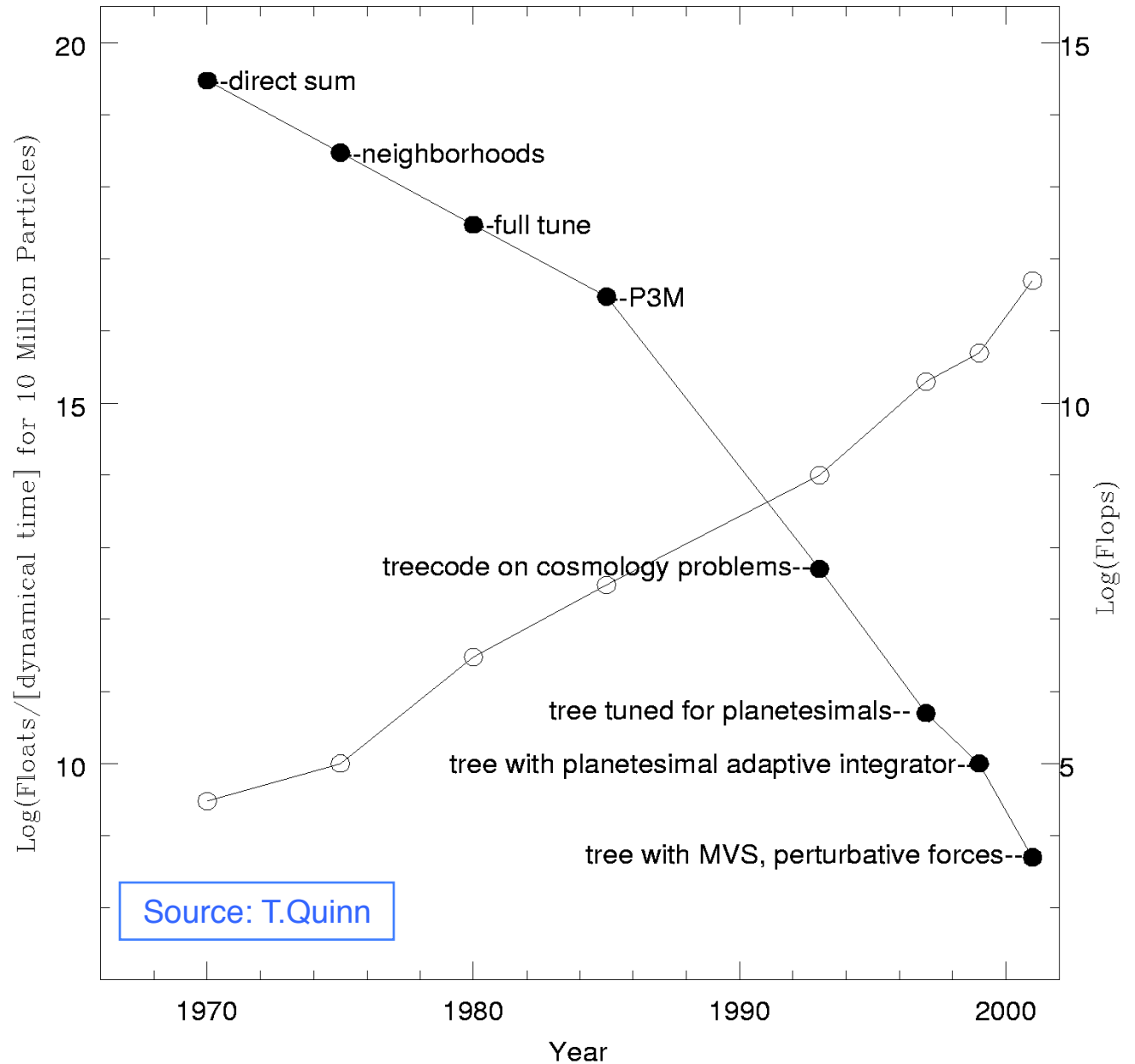solving sparse
linear
systems

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

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

Log(Flops)

- direct sum
- neighborhoods
- full tune
- P3M

treecode on cosmology problems--

tree tuned for planetesimals--

tree with planetesimal adaptive integrator--

tree with MVS, perturbative forces--

Source: T.Quinn

Year

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
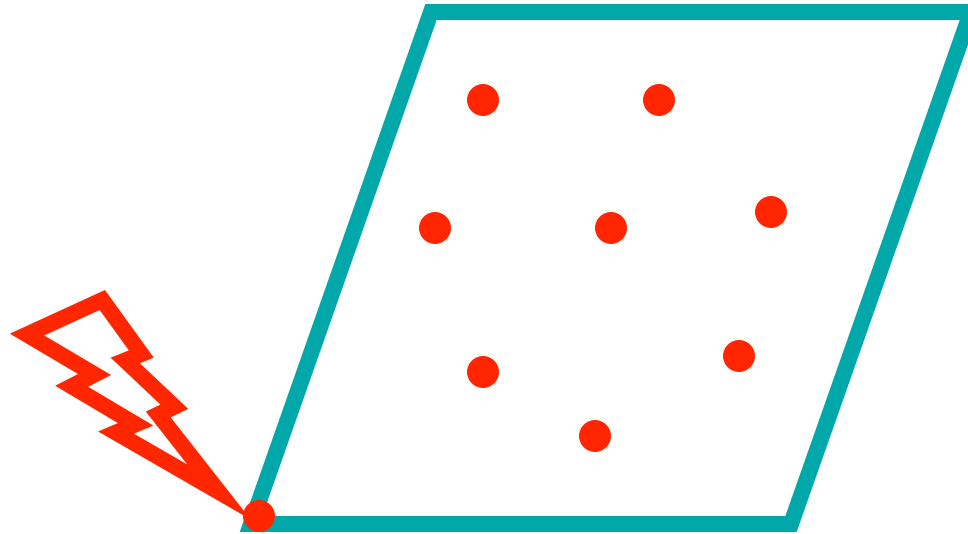
Printed Circuit Board
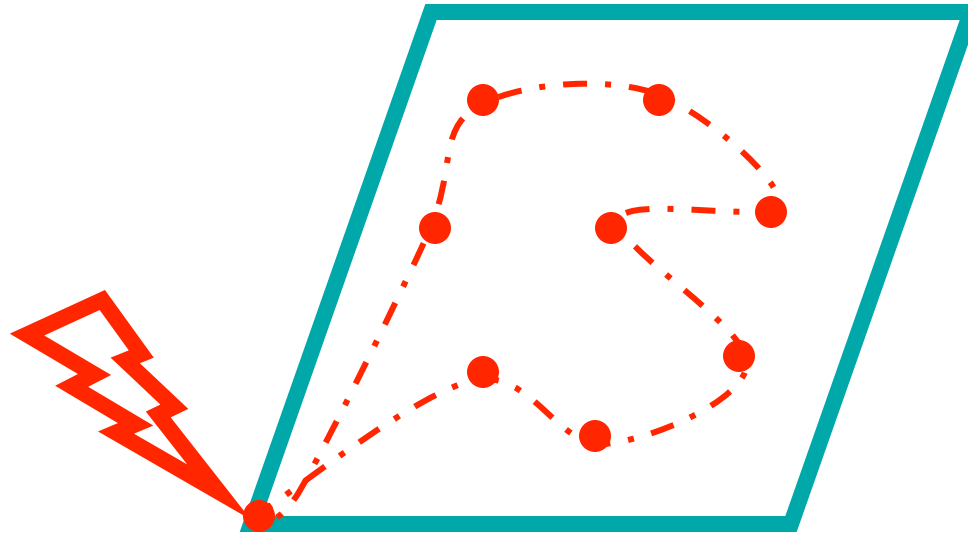
A Well-defined Problem

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?
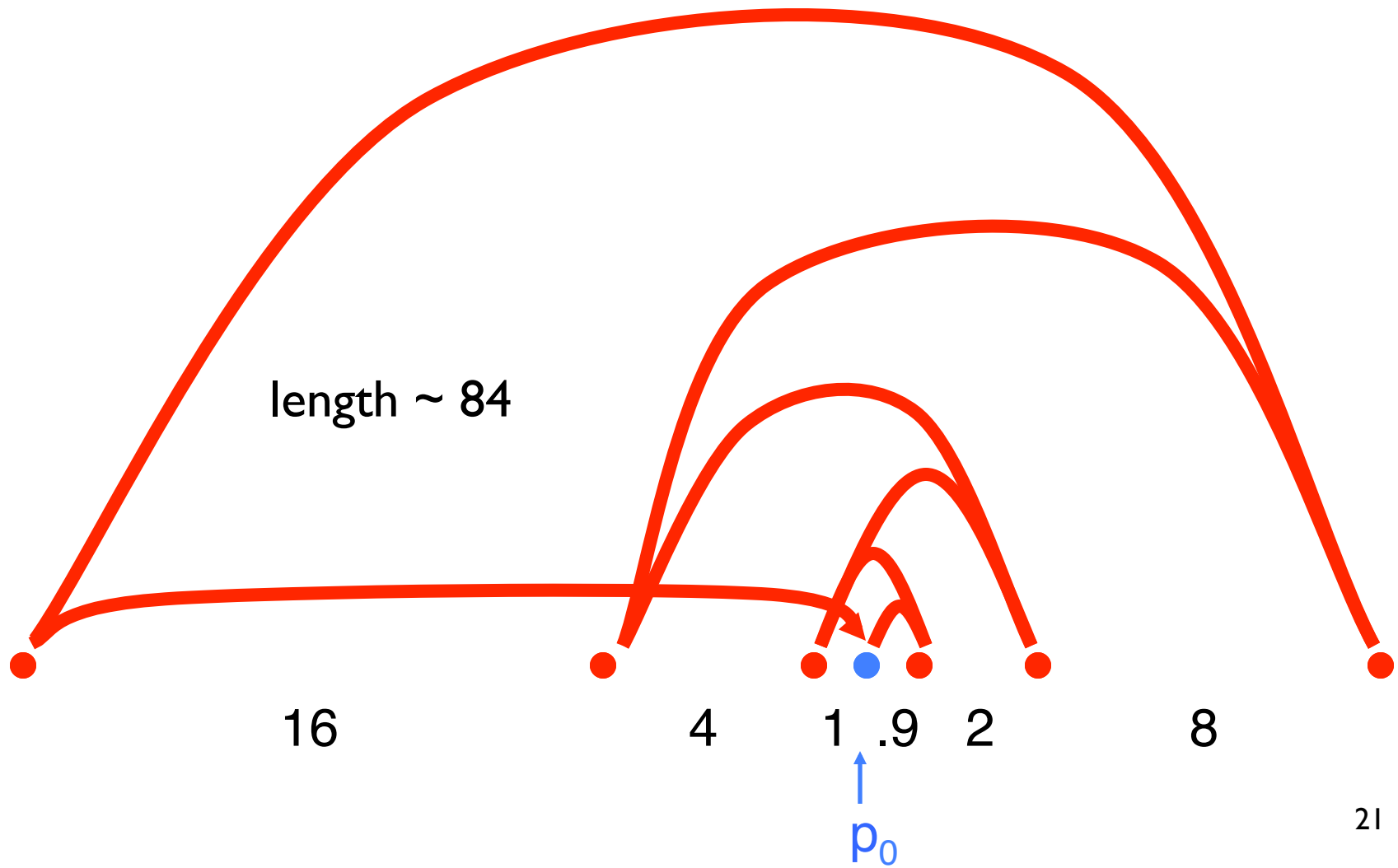
Nearest Neighbor Heuristic

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.
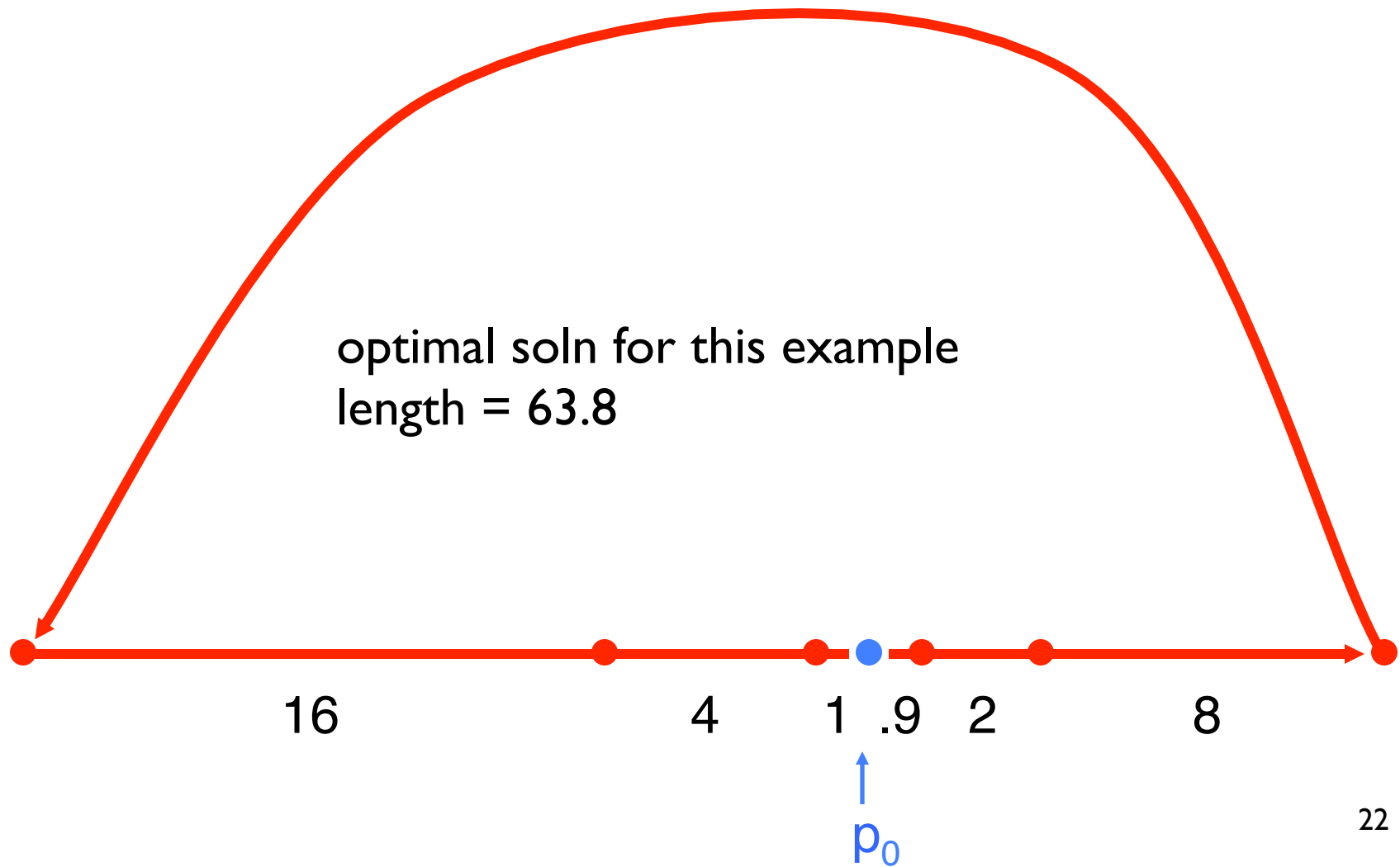
Start at some point p_0

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
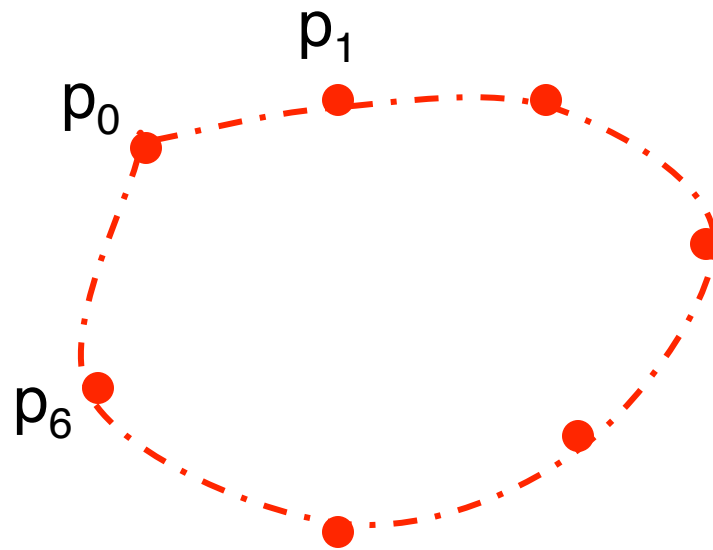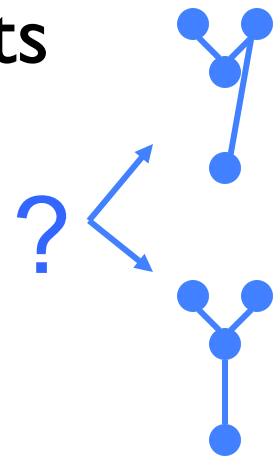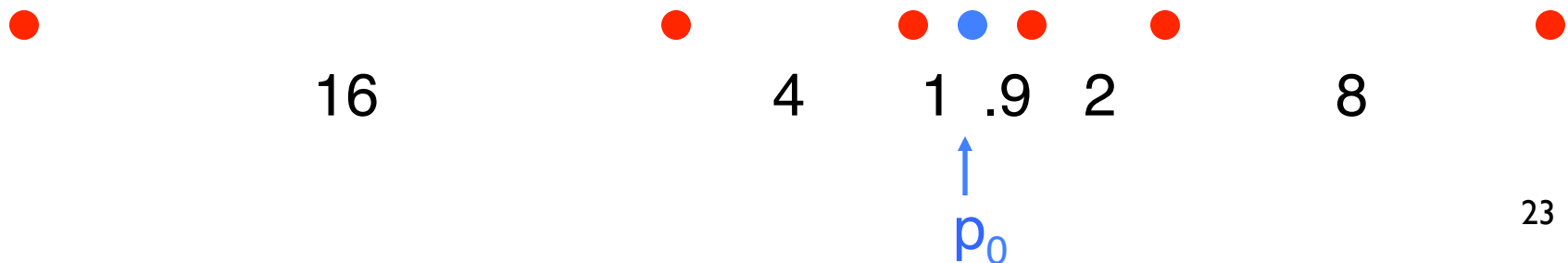
Revised idea - Closest pairs first
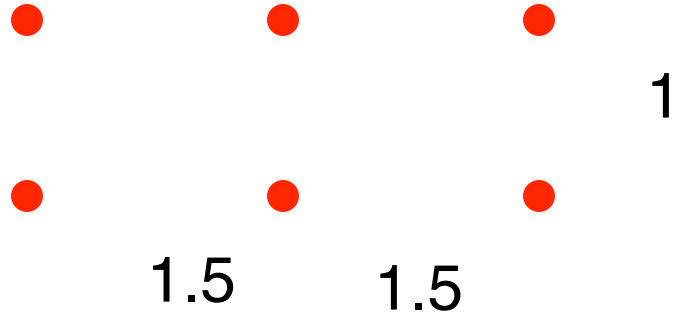
Repeatedly join the closest pair of points

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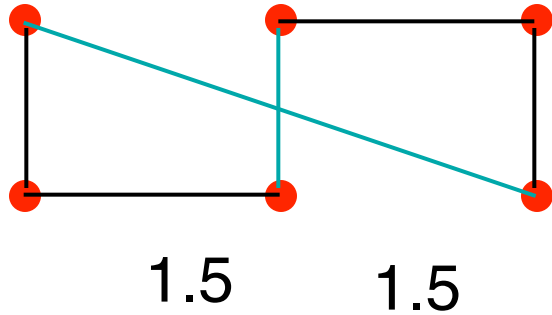endpoints, but not points in middle,
of path segments already created.)
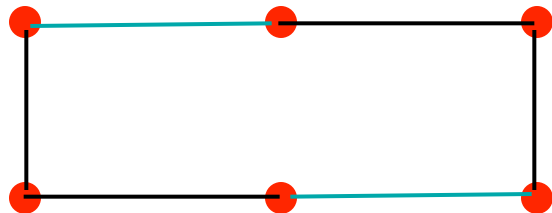
How does this work on our bad example?

16   4   1 .9 2   8

p_0

Another bad example

Another bad example

1

1.5   1.5

$6+\sqrt{10} = 9.16$

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