CSE 417: Algorithms and Computational Complexity

1: Algorithms and Efficiency

Winter 2002

Instructor: Larry Ruzzo

TA: Justin Campbell

TA: Bill Pentney

1

Algorithms: 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
  - Ex: sorting names
  - Ex: checking for primality

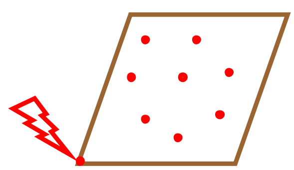
2

Algorithms: an example 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

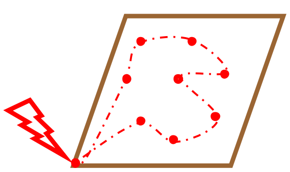
3

Printed Circuit Board



4

Printed Circuit Board



5

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**.

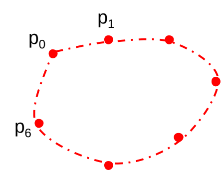
- How might you solve it?

6

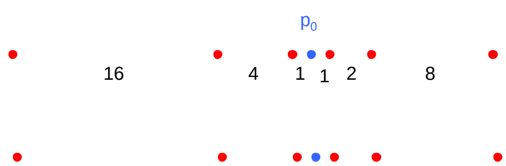
Nearest Neighbor Heuristic

- 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



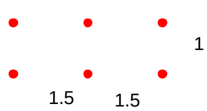
An input where it works badly



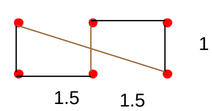
Revised idea - Closest Pairs first

- 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



1+√10 vs 3

Something that works

- For each of the n! orderings of the points
  check the length of the cycle you get
- Keep the best one

Efficiency

- The two incorrect algorithms were **greedy**
  - they made choices and never reconsidered their choices
  - often it does not work
    - when it does the algorithms are typically efficient
- Our correct algorithm 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!)

Measuring efficiency: The RAM model

- RAM = Random Access Machine
- Time ≈ # of instructions executed in an ideal assembly language
  - each simple operation (+,*,-,=,if,call) takes one time step
  - each memory access takes one time step
- No bound on the memory

We left out things but...

- Things we've dropped
  - memory hierarchy
    - disk, caches, registers have many orders of magnitude differences in access time
  - not all instructions take the same time in practice
- However,
  - the RAM model is useful for designing algorithms and measuring their efficiency
  - one can usually tune implementations so that the hierarchy etc. is not a huge factor

Efficiency: What kind of analysis?

- Problem size n
  - **Worst-case complexity**: max # steps algorithm takes on any input of size n
  - **Best-case complexity:** min # steps algorithm takes on any input of size n
  - **Average-case complexity**: avg # steps algorithm takes on inputs of size n

Pros and cons:

- Best-case
  - unrealistic overselling
  - can tune an algorithm so it works on one easy input
- Worst-case
  - a fast algorithm has a comforting guarantee
  - no way to cheat by hard-coding special cases
  - maybe too pessimistic
- Average-case
  - over what distribution?
  - different people may have different average problems