CSE 417: Algorithms and Computational Complexity

Intro to Algorithms & Complexity

Autumn 2002
Paul Beame

Algorithms & Complexity

- Now
  - we know a bunch of problems are undecidable
  - lets try to avoid those and concentrate on getting good solutions to problems that we have a hope of solving
    - Ex: sorting names
    - Ex: checking for primality
  - Simply solving them isn't enough, efficiency is important too

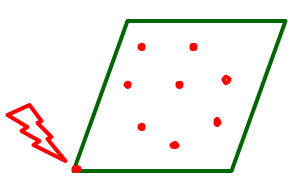
Reading assignment

- Read Chapter 1 of *The ALGORITHM Design Manual*

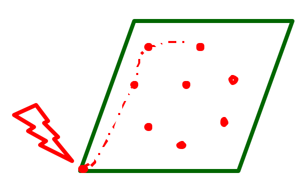
Algorithms: an example problem

- 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

Printed Circuit Board



Printed Circuit Board



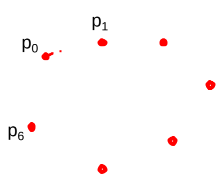
A well-defined Problem

- Input: Given a set **S** of **n** points in the plane
- Output: The shortest cycle tour that visits each point in the set **S**.
- How might you solve it?

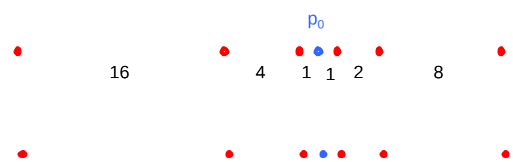
Nearest Neighbor Heuristic

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

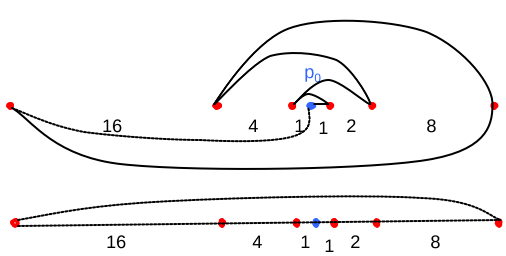
Nearest Neighbor Heuristic



An input where it works badly



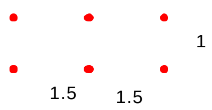
An input where it works badly



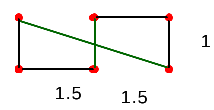
Revised idea - Closest Pairs first

- 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



$1+\sqrt{10}$ 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

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

Complexity 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**

19

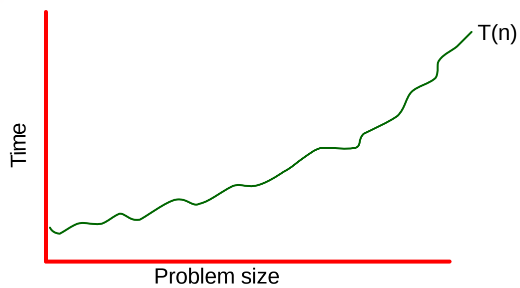
Complexity

- The complexity of an algorithm associates a number **T(n)**, the best/worst/average-case time the algorithm takes, with each problem size **n**.

- Mathematically,
  - $T: N^+ \rightarrow R^+$
  - that is **T** is a function that maps positive integers giving problem size to positive real numbers giving number of steps.

20

Complexity



21

Why Worst-Case Analysis?

- Appropriate for time-critical applications, e.g. avionics
- Unlike Average-Case, no debate about what the right definition is
- Analysis often easier
- Result is often representative of "typical" problem instances
- Of course there are exceptions…

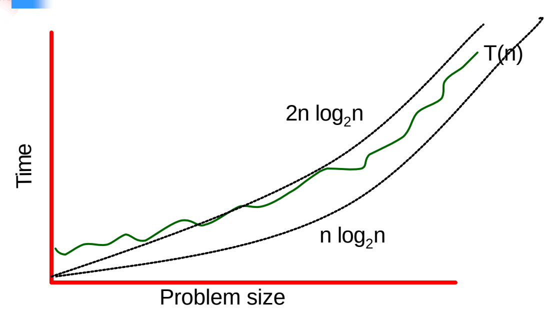
22

O-notation etc

- Given two functions **f** and $g: N \rightarrow R$
  - **f(n)** is **O(g(n))** iff there is a constant **c**>0 so that **f(n)** is eventually always $\leq$ **c g(n)**
  - **f(n)** is Ω**(g(n))** iff there is a constant **c**>0 so that **f(n)** is eventually always $\geq$ **c g(n)**
  - **f(n)** is Θ**(g(n))** iff there is are constants c_1 and c_2>0 so that eventually always $c_1 g(n) \leq f(n) \leq c_2 g(n)$

23

Complexity



24

Examples

- **$10n^2$-16n+100** is **$O(n^2)$** also **$O(n^3)$**
  - $10n^2-16n+100 \le 11n^2$ for all $n \ge 10$
- **$10n^2$-16n+100** is **W(n^2)** also **W(n)**
  - $10n^2-16n+100 \ge 9n^2$ for all $n \ge 16$
  - Therefore also **$10n^2$-16n+100** is **Q(n^2)**
- **$10n^2$-16n+100** is **not $O(n)$** also **not W(n^3)**
- **Note:** I don't use notation **$f(n)=O(g(n))$**

Working with O-W-Q notation

- Claim: For any **a**, **b**>**1** **$\log_a n$** is **Q$(\log_b n)$**
  - **$\log_a n = \log_a b \times \log_b n$** so letting **$c=\log_a b$** we get that **$c\log_b n \le \log_a n \le c\log_b n$**
- Claim: For any **a** and **b**>**0**, **$(n+a)^b$** is **$\Theta(n^b)$**
  - $(n+a)^b \le (2n)^b$ for $n \ge |a|$
    $= 2^b n^b = cn^b$ for $c=2^b$ so $(n+a)^b$ is $O(n^b)$
  - $(n+a)^b \ge (n/2)^b$ for $n \ge 2|a|$
    $=2^{-b} n^b = c'n$ for $c'=2^{-b}$ so $(n+a)^b$ is W(n^b)