Algorithms & Complexity

Now
- we know a bunch of problems are undecidable
- let's 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
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 $p_0$
- 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 $p_0$

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

Something that works

Efficiency

Measuring efficiency:
The RAM model

We left out things but...
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**: $\text{avg}$ # steps algorithm takes on inputs of size $n$

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,
    - $T : \mathbb{N} \to \mathbb{R}$
    - that is $T$ is a function that maps positive integers giving problem size to positive real numbers giving number of steps.

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…

O-notation etc

- Given two functions $f$ and $g : \mathbb{N} \to \mathbb{R}$
  - $f(n)$ 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 $\Omega(g(n))$ iff there is a constant $c > 0$ so that $f(n)$ is eventually always $\geq c g(n)$
  - $f(n)$ is $o(g(n))$ iff there are constants $c_1$ and $c_2 > 0$ so that eventually always $c_1 g(n) \leq f(n) \leq c_2 g(n)$
  - $f(n)$ is $\omega(g(n))$ iff there is a constant $c > 0$ so that $f(n)$ is eventually always $\geq c g(n)$
Examples

- $10n^2 - 16n + 100$ is $O(n^2)$ also $O(n^3)$
- $10n^2 - 16n + 100 \leq 11n^2$ for all $n \geq 10$
- $10n^2 - 16n + 100$ is $\Omega(n^2)$ also $\Omega(n)$
- $10n^2 - 16n + 100 \geq 9n^2$ for all $n \geq 16$
- Therefore also $10n^2 - 16n + 100$ is $\Theta(n^2)$
- $10n^2 - 16n + 100$ is not $O(n)$ also not $\Omega(n^2)$

Note: I don’t use notation $f(n)=O(g(n))$

Working with $O$-$\Omega$-$\Theta$ notation

- Claim: For any $a, b > 1$, $\log_a n$ is $\Theta(\log_b n)$
  - $\log_a n \cdot \log_b n$ so letting $c=\log_b n$ we get that $\frac{c}{n} \leq \log_a n \leq \frac{c}{n}$

- Claim: For any $a$ and $b > 0$, $(n+a)^b$ is $\Theta(n^b)$
  - $(n+a)^b \leq (2n)^b$ for $n \geq |a|$ 
    - $2^b n^b = cn^b$ for $c=2^b$ so $(n+a)^b$ is $O(n^b)$
  - $(n+a)^b \geq (n/2)^b$ for $n \geq 2|a|$ 
    - $2^b n^b = c'n^b$ for $c' = 2^b$ so $(n+a)^b$ is $\Omega(n^b)$