Conclusions

CptS 223 – Advanced Data Structures

Larry Holder
School of Electrical Engineering and Computer Science
Washington State University
Course Overview

- Advanced data structures
  - Trees, hash tables, heaps, disjoint sets, graphs
- Algorithm development and analysis
  - Insert, delete, search, sort
- Applications
- Object-oriented implementation in C++
Analysis Tools

- Counting primitive operations
  - Exponents, logarithms and summations
- Analyzing recursive solutions
  - Recurrence equations: $T(N) = 2T(N/2) + \Theta(N)$
- Proof by induction and contradiction
- Rate of growth notation: $O, \Omega, \Theta$
  - Summarizes analysis
  - Eases comparison among solutions
Performance

![Graph showing running time vs. input size with different time complexities: Linear, $O(N \log N)$, Quadratic, and Cubic.]
Object-Oriented Design in C++

- Encapsulation: Class = Data + Methods
- Information hiding
  - Hiding implementation details
- Operator overloading
  - Perform familiar operations (<, ==) with complex elements
- Templates
  - Design data structures independent of element type
- Standard Template Library (STL)
Basic Data Structures

- Lists, stacks, queues
- O(1) insert/delete
- O(N) search
- STL: vector, list, stack, queue
Advanced Data Structures

- Trees
  - Binary search tree
    - $O(\log N)$ insert, delete and search
  - Balanced BST: AVL and Splay
  - Massive trees: B-tree
  - STL: set, map
Advanced Data Structures

- Hash tables
  - O(1) insert and search
  - Collision resolution
    - Chaining, Open addressing
  - Good hash functions, probe sequences
  - Rehashing
  - Extendible hashing

- STL+: hash_set, hash_map
Advanced Data Structures

- Heaps (Priority Queues)
  - Keep elements partially ordered
  - Heap = complete binary tree
  - $O(\log N)$ insert, delete (worst-case)
  - $O(1)$ insert (average-case)
  - Mergeable heaps: Binomial heap
  - STL: priority_queue
Advanced Data Structures

- Disjoint sets
  - Implement equivalence class operations
  - Find and Union
  - Tree representation
  - Union by rank
  - Path compression
  - $O(1)$ find, union (average-case)
Advanced Data Structures

- Graphs
  - Adjacency list vs. adjacency matrix

- Algorithms
  - Breadth-first search (BFS): $O(V+E)$
  - Depth-first search (DFS): $O(V+E)$
  - Topological sort (DFS): $O(V+E)$
  - Shortest path: $O(E \log V)$
  - Maximum flow: $O(E^2 \log V)$
  - Minimum spanning tree: $O(E \log V)$
  - Biconnectivity and articulation points (DFS): $O(V+E)$
  - Euler circuits (DFS): $O(V+E)$
  - Strongly-connected components (DFS): $O(V+E)$
I've been asked to reduce headcount.

To be fair about it I created a scientific algorithm to decide who goes.

I thought you were firing the people with the highest salaries.

Okay, maybe "algorithm" is an overstatement.
Algorithm Design and Analysis: Sorting

- Comparison sorts
  - Insertion sort
  - Merge sort
  - Heap sort
  - Quicksort
  - Lower bound: \( \Theta(N \log N) \)

- Linear sorting
  - Counting sort
  - Bucket sort
  - External sorting
## Algorithm Design and Analysis: Sorting

<table>
<thead>
<tr>
<th>Sort</th>
<th>Worst Case</th>
<th>Average Case</th>
<th>Best Case</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>InsertionSort</td>
<td>$\Theta(N^2)$</td>
<td>$\Theta(N^2)$</td>
<td>$\Theta(N)$</td>
<td>Fast for small $N$</td>
</tr>
<tr>
<td>MergeSort</td>
<td>$\Theta(N \log N)$</td>
<td>$\Theta(N \log N)$</td>
<td>$\Theta(N \log N)$</td>
<td>Requires memory</td>
</tr>
<tr>
<td>HeapSort</td>
<td>$\Theta(N \log N)$</td>
<td>$\Theta(N \log N)$</td>
<td>$\Theta(N \log N)$</td>
<td>Large constants</td>
</tr>
<tr>
<td>QuickSort</td>
<td>$\Theta(N^2)$</td>
<td>$\Theta(N \log N)$</td>
<td>$\Theta(N \log N)$</td>
<td>Small constants</td>
</tr>
</tbody>
</table>
## Hard Problems

<table>
<thead>
<tr>
<th>Input Size vs. Complexity</th>
<th>10</th>
<th>20</th>
<th>30</th>
<th>40</th>
<th>50</th>
<th>60</th>
</tr>
</thead>
<tbody>
<tr>
<td>( n )</td>
<td>.00001 s</td>
<td>.00002 s</td>
<td>.00003 s</td>
<td>.00004 s</td>
<td>.00005 s</td>
<td>.00006 s</td>
</tr>
<tr>
<td>( n^2 )</td>
<td>.0001 s</td>
<td>.0004 s</td>
<td>.0009 s</td>
<td>.0016 s</td>
<td>.0025 s</td>
<td>.0036 s</td>
</tr>
<tr>
<td>( n^3 )</td>
<td>.001 s</td>
<td>.008 s</td>
<td>.027 s</td>
<td>.064 s</td>
<td>.125 s</td>
<td>.216 s</td>
</tr>
<tr>
<td>( n^5 )</td>
<td>.1 s</td>
<td>3.2 s</td>
<td>24.3 s</td>
<td>1.7 min</td>
<td>5.2 min</td>
<td>13.0 min</td>
</tr>
<tr>
<td>( 2^n )</td>
<td>.001 s</td>
<td>1.0 s</td>
<td>17.9 min</td>
<td>12.7 days</td>
<td>35.7 years</td>
<td>366 centuries</td>
</tr>
<tr>
<td>( 3^n )</td>
<td>.059 s</td>
<td>58 min</td>
<td>6.5 years</td>
<td>3855 centuries</td>
<td>2x10^8 centuries</td>
<td>1.3 x 10^{13} centuries</td>
</tr>
</tbody>
</table>
Classes of Hard Problems

Does P = NP?
NP-Complete Problems

- The hardest problems in NP
- Prove problem is NP-Complete by “reducing” known NP-Complete problem to it
- Determining the class of a problem helps us know the best performance we can achieve
- Approximation algorithms
Applications

- Operating systems
- Compilers
- Databases
- Route planning
- Dictionary/symbol lookup
- Molecular analysis
- Image processing
- Theory of computation
- Many more …
Problem-solving

Problem

Design Data Structure → Design Algorithms → Evaluate/Analyze Performance

okay → done
not okay

After several tries; look for lower-bound on problem.

Upper bound on solution.
Summary

- Moral
  - Appropriate data structures ease design and improve performance

- Challenge
  - Design appropriate data structure and associated algorithms for a problem
  - Analyze to show improved performance

PLEASE FILL OUT YOUR ON-LINE COURSE EVALUATION!!