



# 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++

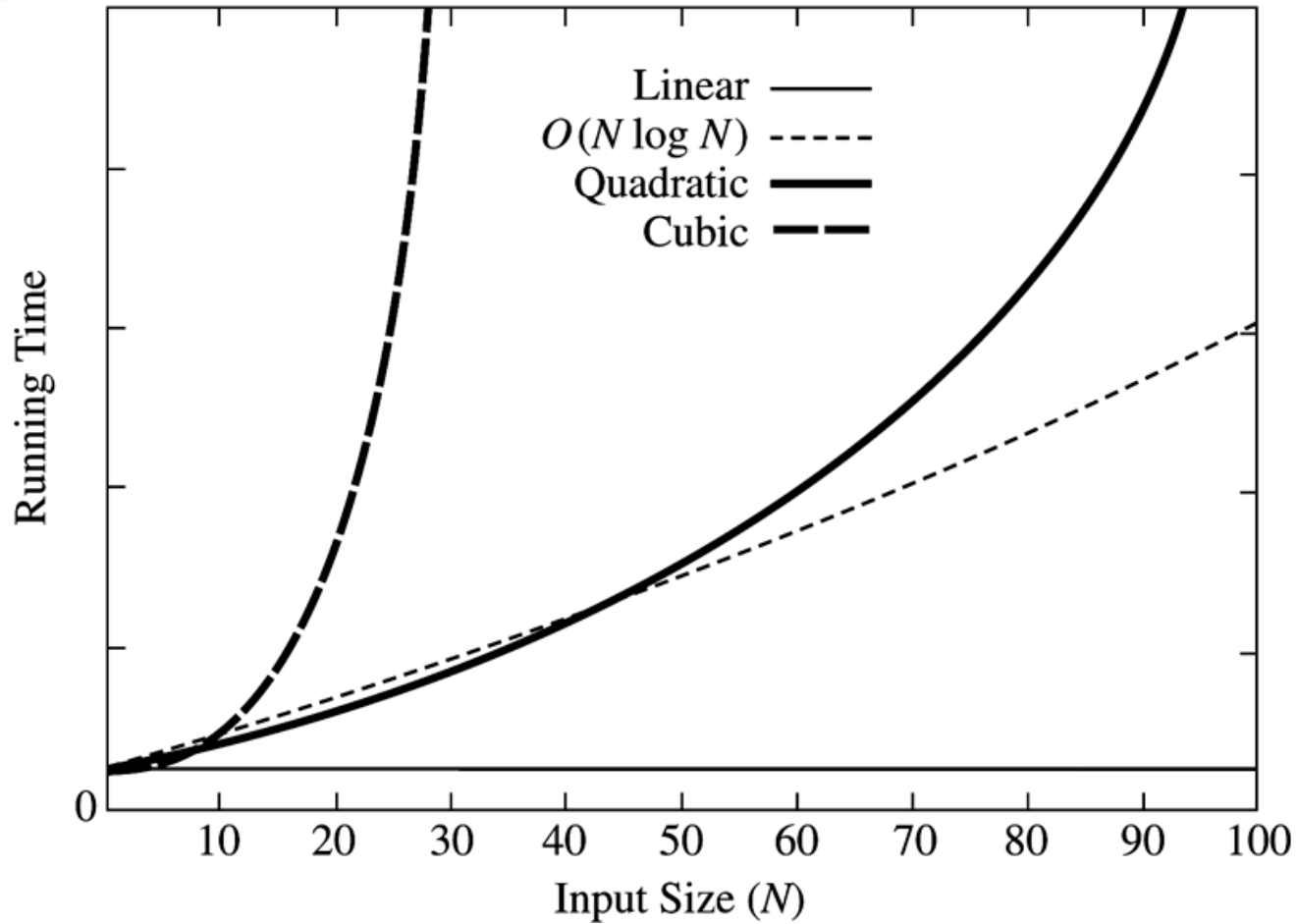


# Analysis Tools

---

- 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





# 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)

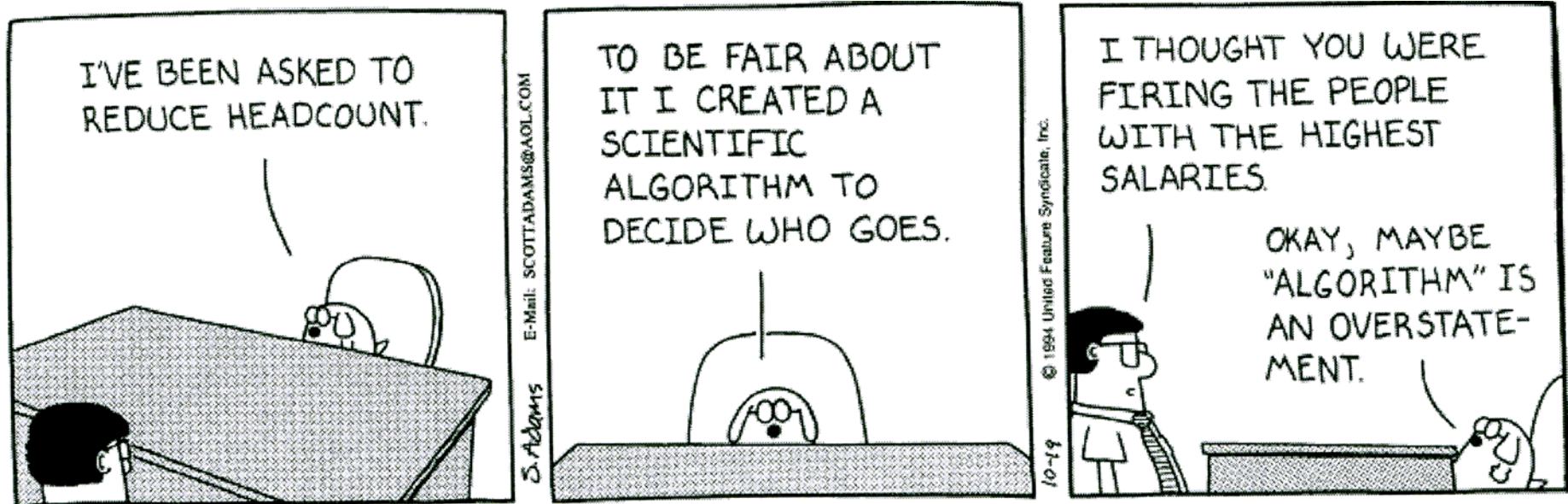


# Advanced Data Structures

---

- 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):  $O(V+E)$

# Algorithm Design



**Dilbert** by Scott Adams From the ClariNet electronic newspaper Redistribution prohibited info@clarinet.com



# 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

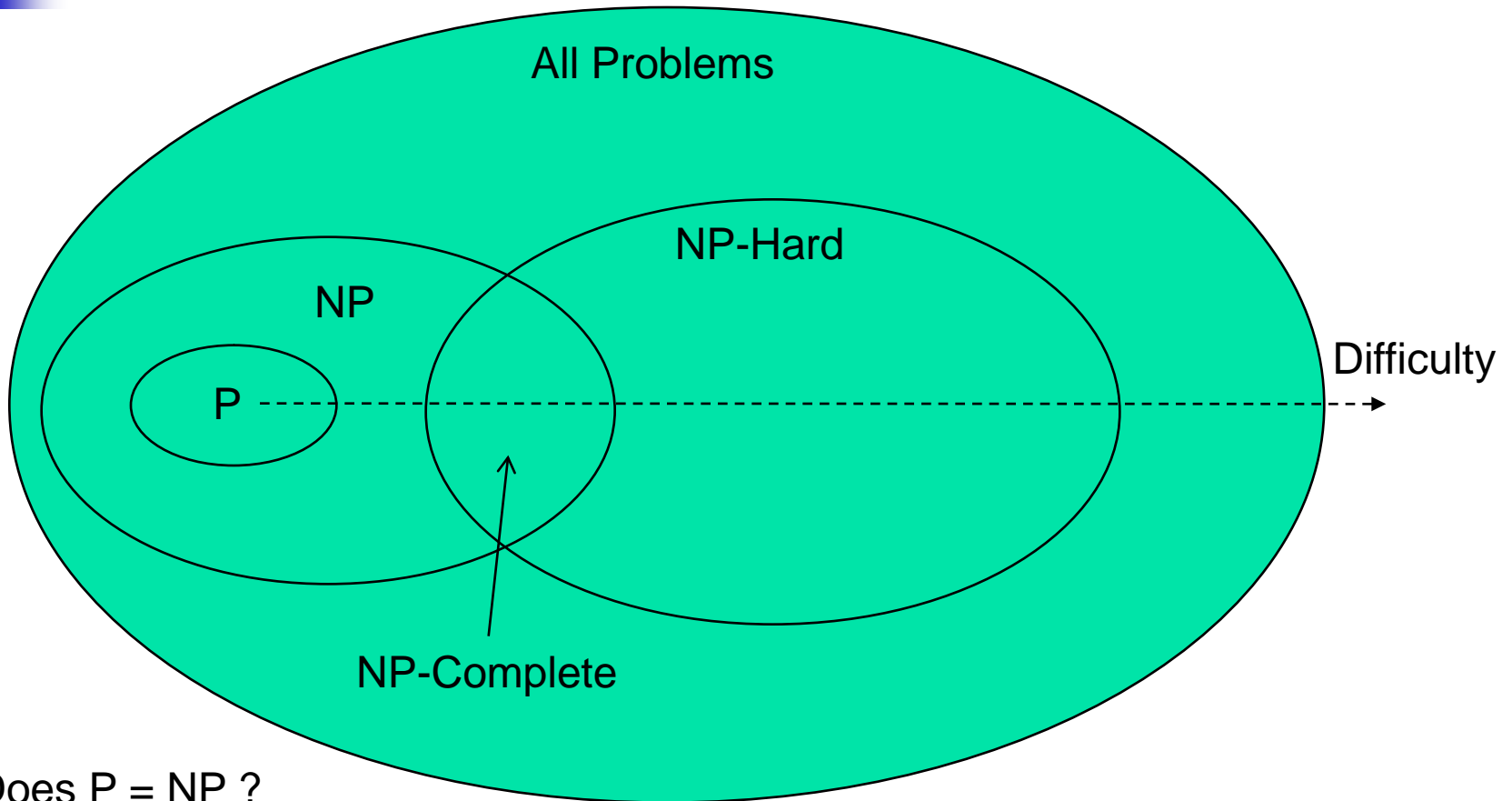
Sort	Worst Case	Average Case	Best Case	Comments
InsertionSort	$\Theta(N^2)$	$\Theta(N^2)$	$\Theta(N)$	Fast for small N
MergeSort	$\Theta(N \log N)$	$\Theta(N \log N)$	$\Theta(N \log N)$	Requires memory
HeapSort	$\Theta(N \log N)$	$\Theta(N \log N)$	$\Theta(N \log N)$	Large constants
QuickSort	$\Theta(N^2)$	$\Theta(N \log N)$	$\Theta(N \log N)$	Small constants



# Hard Problems

Input Size vs. Complexity	10	20	30	40	50	60
$n$	.00001 s	.00002 s	.00003 s	.00004 s	.00005 s	.00006 s
$n^2$	.0001 s	.0004 s	.0009 s	.0016 s	.0025 s	.0036 s
$n^3$	.001 s	.008 s	.027 s	.064 s	.125 s	.216 s
$n^5$	.1 s	3.2 s	24.3 s	1.7 min	5.2 min	13.0 min
$2^n$	.001 s	1.0 s	17.9 min	12.7 days	35.7 years	366 centuries
$3^n$	.059 s	58 min	6.5 years	3855 centuries	$2 \times 10^8$ centuries	$1.3 \times 10^{13}$ centuries

# 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

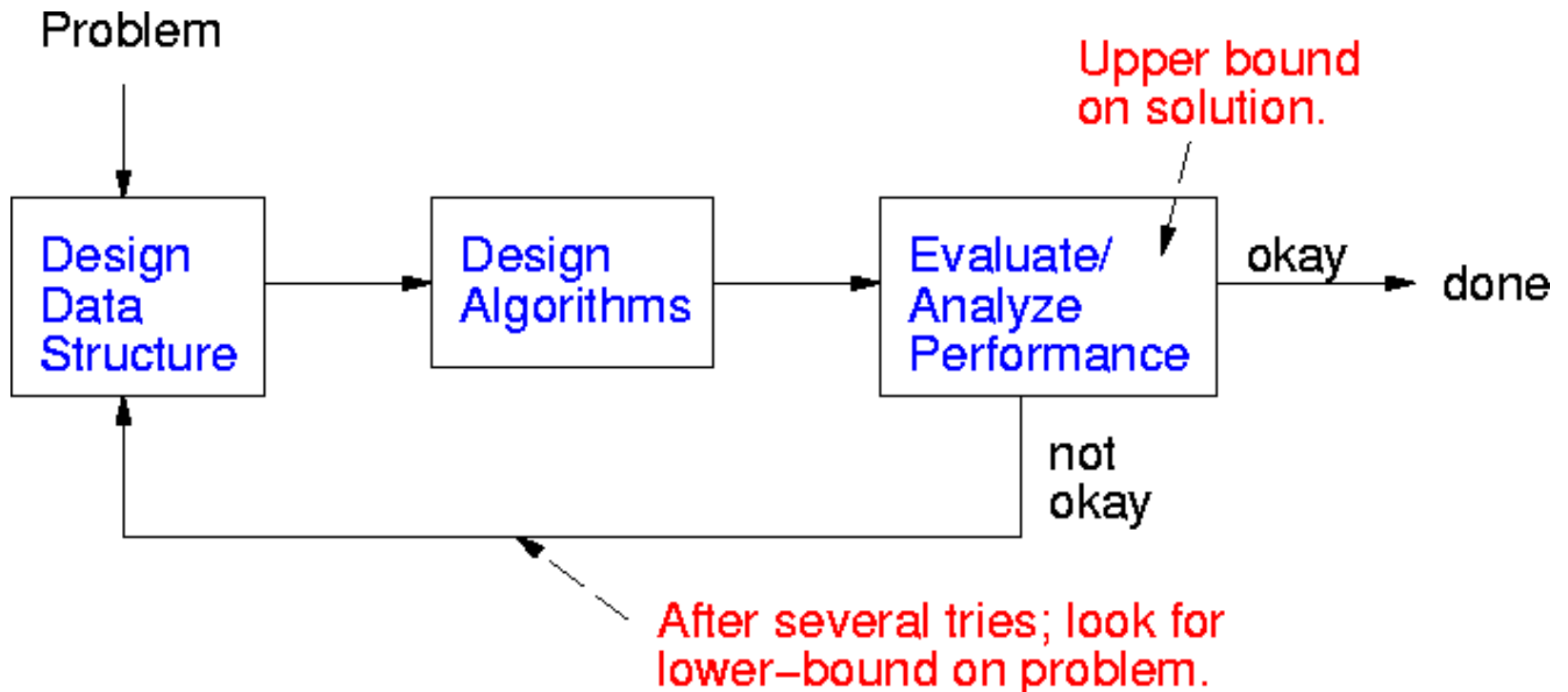


# Applications

---

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

# Problem-solving





# 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!!