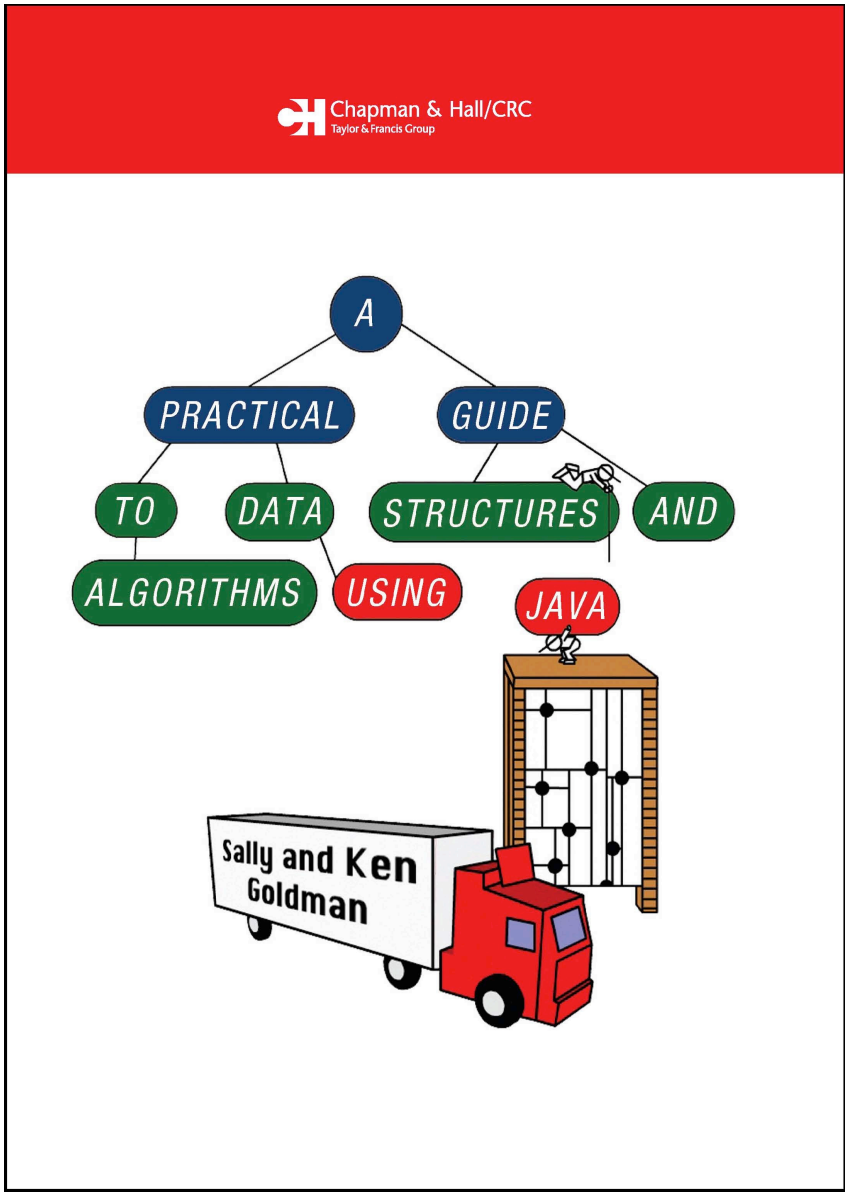
Instructor's Guide



http://goldman.cse.wustl.edu/crc2007/

Key Features of this Book

A fresh alternative to traditional algorithm surveys, *A Practical Guide to Data Structures and Algorithms using Java* by Goldman & Goldman presents a systematic, applications-centered approach to data structure design and practical implementation. The book provides comprehensive coverage, with complete integrated Java implementations, of a wide variety of important data structures, including many useful abstract data types not provided in the standard Java libraries. Fundamental algorithms, also with complete implementations, are presented within the context of their supporting data structures. Case studies, examples, decision trees and comparison charts throughout the book illustrate and support a methodology for careful selection and application of data structures and algorithms. Appendices summarize major features of the Java programming language, introduce asymptotic notation and complexity analysis, and discuss design patterns illustrated throughout the book. Software developers will reach for this guide to quickly identify the best data structure or algorithm for their problem.

This book has the following features that support its use as a textbook for a course on data structures and algorithms.

- The practical top-down approach guides you to the most natural and efficient data structure for your application. Case studies and examples throughout the book provide motivation and illustrate the methodology. This feature supports the objective of teaching students to select an appropriate ADT based on application requirements, and to then select among competing data structures and algorithms according to the performance requirements of an application.

- Complete thoroughly explained Java implementations exploit object-oriented design principles for maximum clarity and to expose key differences among related data structures and algorithms.

- The organized and stylized presentation puts the right level of detail at your fingertips. Accessible comparison tables support rapid assessment of trade-offs among implementations, along with complete asymptotic analysis for detailed comparison.

- Design decisions, optimizations, and correctness highlights are presented to deepen understanding and provide opportunities for guided customization of the Java implementations, which are provided with complete documentation on the accompanying CD. This supports the goal of students learning how to customize and/or combine standard data structures and algorithms to meet the specific needs of applications.

- A true marriage of theory and practice, this book sets a new standard as a comprehensive practical guide to data structures and algorithms. When used as a text book, the course can provide a scaffolding of understanding to learn about new data structures and algorithms.

- No single course can "cover" the entire book, but by the end of the course, students will have the appropriate scaffolding to make productive use of the portions not covered. An advantage of using this book versus a standard text book is that it is an excellent resource book that the students can use throughout the rest of their education and careers.

Guide to Instructor

An introductory data structures and algorithms course could begin with Part I, with an emphasis on selecting abstract data types and implementations appropriate for applications. Then, based on the interests of the instructor and students, a selected subset of the ADTs could be covered in detail. It is not necessary, or even feasible, to present every data structure for each ADT, but instead the comparison tables can be used to highlight the differences, and then students can concentrate on one or two representative implementations of each. For courses with a more applied focus, homework and projects might concentrate on empirical comparisons of the provided implementations, modifications based on the optimizations suggested in the chapters, and projects based on the case studies. A more theoretical course might cover the complexity analysis material in the appendix early in the course, and focus more on theoretical analysis of the algorithms and correctness proofs.

This packet highlights the online educational materials will are available at

`http://goldman.cse.wustl.edu/crc2007/`

The course from which these materials are drawn is a sophomore level introductory algorithms and data structures course. Students taking this course have already studied basic data structures such as linked lists and (unbalanced) binary search trees. These materials could be easily adjusted for a data structures course by replacing the more algorithmic components by the introductory data structures material. As just one other example, an advanced data structures course could be created by covering some of the more advanced data structures presented in the book that are not covered in the course from which these materials are drawn.

This packet includes the following:

sample syllabus - Sample syllabus for an introductory data structures and algorithms course

lectures/note - A detailed summary of each lecture where a link is provided for each lecture component to a recorded lectures (avi files that include full audio and high resolution screen capture of all activity on the tablet PC) and the completed lecture notes (pdf files of the complete content of each page of the lecture).

projects - sample implementation exercises of varying length and difficulty along with code provided to support the project. Solutions are available to qualifying instructors from the publisher.

homework problems - an organized repository of homework exercises with a brief overview of the goals for each homework. Solutions are available to qualifying instructors from the publisher.

Sample syllabus for an Introductory Data Structures and Algorithms Course
(90-minute twice weekly over a 15 week semester)

Date	**Topic**	**Readings (Goldman & Goldman)**	**Due**	**Assigned**
Lecture 1	Introduction: Which algorithm is best?	Appendix B.1		
Lecture 2	Divide-and-conquer algorithms	Handout 1		Project 1
Lecture 3	Divide-and-conquer algorithms (cont.)	Preface, Chapter 1		
Lecture 4	Asymptotic notation	Appendix B.2		
Lecture 5	Analyzing divide-and-conquer algorithms	Appendix B.6		
Lecture 6	ADT Taxonomy Part I, Positional Collection ADT	Sections 2.1-2.6.1, and Chapter 9	Project 1	Homework 1
Lecture 7	Quicksort, Randomized quicksort expected time complexity	Section 11.4.5 and Appendix B.5		
Lecture 8	Adversary lower bound technique	Handout 2	Homework 1	Homework 2
Lecture 9	Linear time sorting algorithms	Section 11.4.6		
Lecture 10	ADT Taxonomy Part II and Set ADT	Sections 2.6.2, 7.3.2, and Chapter 20	Homework 2	Homework 3
Lecture 11	Direct Addressing, Open Addressing	Chapters 21, 22		
Lecture 12	Separate Chaining	Chapter 23	Homework 3	Project 2
Lecture 13	Priority Queue ADT, Binary Heaps	Chapter 24 and 25		
Lecture 14	Ordered Collection ADT, Balanced Search Trees	Chapters 29, 33	Project 2	
	Midterm			
Lecture 15	Red-Black Trees	Chapter 34		Homework 4
Lecture 16	B-trees	Chapter 36		
Lecture 17	B-trees, B+-trees	Sections 37.1-37.2	Homework 4	Project 3
Lecture 18	Skip Lists	Sections 38.1-38.5		
Lecture 19	Skip List analysis, Ordered collection data structures comparisons	Section 38.6		
Lecture 20	Digitized Ordered Collections, Spatial Collections, and k-d Trees	Chapters 39, 46, Section 47.1		
Lecture 21	Graph problems, graph representations	Section 2.7, Chapter 52	Project 3	Homework 5
Lecture 22	Breadth-first search (BFS)	Section 53.4		
Lecture 23	Dijkstra's shortest path algorithm	Section 57.2	Homework 5	Project 4
Lecture 24	Greedy Tree Builder	Sections 57.1		
Lecture 25	Prim's and Kruskal's Minimum Spanning Tree (MST) Algorithms	Sections 57.3 and 57.4		
Lecture 26	DFS and topological sort	Sections 53.5, 53.6		Homework 6
Lecture 27	Garbage collection algorithms	Section 53.9	Project 4	
Lecture 28	Course Review		Homework 6	

A Practical Guide to Data Structures and Algorithms Using Java

Sally A. Goldman and Kenneth J. Goldman
Washington University in St. Louis

Course Lectures

The lectures are provided in two formats: video and pdf. The videos are avi files that include full audio and high resolution screen capture of all activity on the tablet PC. The pdf files show the complete content of each "page" of the lecture. Used in conjunction, the pdf files are helpful to quickly identify sections of lectures that you may want to view in detail within the videos.

These lectures can be used in several ways:

- as ideas for course instructors
- as reinforcement or review for students
- as optional enrichment for students on topics not covered in class
- as student preparation for active learning class sessions, so time in class is more engaging

Lecture 1: Introduction

Course Introduction (7:54) [lecture notes]
We describe the goals of this course.

Motivating asymptotic complexity (37:48) [lecture notes]
We use the problem of finding the closest-pair of points in the plane to help motivate the standard asymptotic time complexity analysis that is used by computer scientists. One goal is to demonstrate that the asymptotic value of the number of lines of code executed is a good "back-of-the-envelope" calculation that can be used to evaluate which algorithms will run fastest in practice. To help make this point, we report on empirical tests comparing execution time to the number of statements executed. We stress the value of being able to quickly rule out inferior algorithms, so that the time spent on coding and empirical testing can be used for the most promising algorithms.

Introduction of the closest-pair problem (15:39) [lecture notes]
We present the closest-pair problem, which is used at the start of the course to motivate the theoretical foundations that are going to be used throughout the course. This problem has been selected since it helps students to appreciate that even though they may be able to solve a problem from first principles, this course can help them learn to develop more efficient solutions. This lecture component also covers the brute force algorithm that computes the distance between all pairs of points, and has a high-level overview of the divide-and-conquer algorithm, built up by combining ideas suggested by the class.

Analysis of the brute force closest-pair algorithm (6:35) [lecture notes]
We analyze the brute force closest pair algorithm that computes the distance between all pairs of points including a brief discussion on ways to compute the number of pairs of points from first principles for anyone without combinatorics background.

Empirical comparisons of running times (5:19) [lecture notes]
We present actual running times (in seconds) for the brute force and divide-and-conquer algorithms for the closest-pair problem for differing input sizes on two different computers that vary significantly in processor speed. The goal is to help students appreciate the difference in real time between an $O(n \log n)$ and $O(n^2)$ algorithm.

Lecture 2: Divide-and-Conquer Algorithms

Divide-and-conquer algorithm design technique (6:45) [lecture notes] [handout]
We describe the divide-and-conquer technique for algorithm design.

Merge sort algorithm (14:12) [lecture notes]
We present merge sort as an example of the divide-and-conquer technique.

Detailed discussion of the divide-and-conquer closest pair algorithm (31:34) [lecture notes]
We expand upon the high-level ideas introduced in the first lecture for an in-depth discussion of the divide-and-conquer closest pair algorithm.

Handling degeneracies in the divide-and-conquer closest-pair algorithm (14:13) [lecture notes]
We discuss the problem that could occur when two or more points share the same x-coordinate along the line that divides the left and right halves. In particular, we describe how a method (called `leftOf`) can be used to define a unique ordering of the points that is sorted with respect to the x-coordinate to resolve this potential difficulty.

Lecture 3: Divide-and-Conquer Algorithms (cont.)

Pseudocode for the divide-and-conquer closest pair algorithm (8:02) [lecture notes]
We wrap up our discussion of the divide-and-conquer closest pair algorithm by overviewing it through pseudo-code.

Correctness of the divide-and-conquer closest pair algorithm (6:51) [lecture notes]
We argue that the divide-and-conquer algorithm is guaranteed to return the correct answer for all possible inputs. While the

discussion uses this problem as an example, it addresses more generally how induction can be used to prove the correctness of any divide-and-conquer algorithm.

Analysis of the divide-and-conquer closest pair algorithm (42:43) [lecture notes]
We analyze the asymptotic time complexity of the closest pair algorithm. While this lecture motivates the benefits of the master method (a "cookbook" approach to solve recurrences of a specified form), it uses the recursion tree technique to solve the recurrence.

Lecture 4: Asymptotic Notation

Asymptotic Notation (38:19) [lecture notes]
We formally define asymptotic notation (big-Oh, big-Omega, big-Theta, little-oh, little-omega), and discuss the limiting behavior as n approaches infinity. We build upon relationships already understood in working with inequalities to develop intuition about asymptotic notation.

Working with asymptotic notation (31:10) [lecture notes]
We use example problems to develop a better understanding of asymptotic notation. It's important to be able to determine relationships between pairs of functions. As part of this lecture, L'Hopital's rule for computing limits is reviewed.

Analyzing code fragments that are nested loops (6:17) [lecture notes]
We analyze the asymptotic time complexity of example code fragments with nested loops.

Lecture 5: Analyzing Divide-and-Conquer Algorithms

Analyzing divide-and-conquer algorithms (12:36) [lecture notes]
We discuss how to define a recurrence relation to express the asymptotic time complexity of a divide-and-conquer algorithm. Next we present the master method to quickly get an asymptotic solution, or how to use a recursion tree to obtain an exact solution.

Master method to asymptotically solve a recurrence relation (19:04) [lecture notes]
We present the master method, a "cookbook" method to asymptotically solve typical recurrence relations that occur for divide and conquer algorithms when all subproblems are roughly the same size. In particular, it can be used to solve recurrences of the form $T(n) = a\ T(n/b) + f(n)$ where $f(n)$ is of the form $n^l(\log n)^k$ for constants $l \geq 0$ and $k \geq 0$.

Exactly solving a recurrence relation (29:50) [lecture notes]
We illustrate how to use a recursion tree to exactly solve recurrence relations that typical of divide and conquer algorithms.

Lecture 6: ADT Taxonomy Part I and Positional Collection ADT

Overview of the ADT taxonomy (11:57) [lecture notes]
We overview the ADT taxonomy used throughout the book.

Introduction of the manually positioned collection (2:42) [lecture notes]
We provide an introduction to manually positioned collections.

A discussion of the positional collection data structures (62:29) [lecture notes]
We present positional collection data structures, including the singly-linked list, doubly-linked list, array, dynamic array, circular array, and tracked array.

Lecture 7: Quicksort, Expected time complexity

Quicksort (59:31) [lecture notes]
We present quicksort with median-of-three partitioning and also randomized quicksort. We also discuss the common optimization of terminating when the input size is a small value (around 30), and then using insertion sort to complete the sort.

Expected time complexity definition (9:00) [lecture notes]
We formally define expected time complexity.

Analysis of randomized quicksort (10:11) [lecture notes]
We analyze the expected time complexity of randomized quicksort.

Lecture 8: Adversary Lower Bound Technique

Adversary lower bound technique (40:47) [lecture notes] [handout]
Using "20 questions" as an example problem, we describe the adversary lower bound technique. In particular, we focus on the basic adversary strategy of making a list of all possible solutions, and then answering to maximize the number of remaining solutions consistent with all past answers. This adversary strategy gives the same bounds as one obtains using the decision tree technique.

Comparison-based sorting lower bound (16:17) [lecture notes]
We apply the adversary lower bound technique to prove an $\Omega(n \log n)$ lower bound for any comparison-based sorting algorithm.

Lower bound for finding the minimum (13:08) [lecture notes]

We use the problem of finding the minimum element in an array to illustrate that for some problems alternate adversary strategies can yield a better lower bound.

Lecture 9: Linear Time Sorting Algorithms

Counting sort (24:39) [lecture notes]
We present counting sort.

Digitizer interface (9:03) [lecture notes]
We describe the digitizer interface. Similar to the way in which Java's Comparator interface allows a comparison-based sorting algorithm to be applied to any data elements that have a comparator, our Digitizer interface allows any sorting algorithm that treats each element as a sequence of digits to be applied to any data elements for which a Digitizer is defined.

Radix sort (50:19) [lecture notes]
We present radix sort.

Bucket sort (7:19) [lecture notes]
We present bucket sort.

Lecture 10: ADT Taxonomy Part II, Set ADT

ADT taxonomy for algorithmically positioned collections (38:28) [lecture notes]
We return to our earlier discussion of the ADT taxonomy used in the book, focusing on algorithmically positioned collections.

Set ADT (6:58) [lecture notes]
We describe the Set abstract data type.

Lecture 11: Direct Addressing and Open Addressing

Direct addressing (18:56) [lecture notes]
We present the direct addressing data structure.

Introduction to hashing based Set data structures (27:39) [lecture notes]
We introduce hashing, and provide an overview of separate chaining and open addressing.

Open addressing (44:32) [lecture notes]
We present the open addressing data structure.

Lecture 12: Separate Chaining

Separate Chaining (10:18) [lecture notes]
We present the separate chaining data structure.

Comparison of the trade-offs between Set data structures (11:08) [lecture notes]
We systematically compare the advantages and disadvantages of direct addressing, open addressing, and separate chaining.

Lecture 13: Priority Queue ADT and the Binary Heap Data Structure

Priority Queue ADT (7:00) [lecture notes]
We describe the priority queue abstract data type.

Binary Heap (56:34) [lecture notes]
We present the binary heap data structure.

Tracked Binary Heap (5:25) [lecture notes]
We illustrates how the elements stored in a binary heap can be tracked.

Comparison of the trade-offs between priority queue data structures (6:14) [lecture notes]
We systematically compares the advantages and disadvantages of a binary heap, leftist heap, pairing heap, and Fibonacci heap.

Lecture 14: Ordered Collection ADT, Balanced Search Trees

Ordered collection ADT (5:31) [lecture notes]
We describe the ordered collection abstract data type.

Abstract search tree (14:24) [lecture notes]
We discuss the generalization of a binary search tree in which each node can have any number of elements per node. As part of this discussion, we present the properties of an abstract search tree that allow efficient search, and discuss how an inorder traversal can be used to iterate over the elements in sorted order.

Binary search tree review (17:23) [lecture notes]
We review the binary search tree data structure with a focus on how to find the successor (and symmetrically predecessor) of an element, and how to remove an element.

Balanced search tree (27:03) [lecture notes]
We describe the rotations that are used to help create balanced binary search trees. This discussion includes a high-level overview of the ways in which data structures that maintain a balanced binary search tree select when to perform rotations.

Review for Midterm

Midterm topics (15:07) [lecture notes]
A review of the topics that will be covered on the midterm.

Lecture 15: Red-Black Trees

Red-black tree properties (12:24) [lecture notes]
We introduce the red-black tree by describing its properties. We also derive an upper bound on the depth of a red-black tree holding n elements.

Red-black tree methods (54:49) [lecture notes]
We describe the red-black tree methods, starting with a brief discussion of how the non-mutating methods are exactly as for a standard binary search tree. Then we present the mutators. Insertion is presented in depth, followed by a discussion of the high-level approach used by deletion, but without the details of the specific cases that occur.

Lecture 16: B-Trees

Intuition behind the B-Tree design (33:19) [lecture notes]
We discuss secondary storage and use it to motivate the design of a B-tree in terms of its goal of reducing the number of page faults during a search.

B-Tree properties (7:00) [lecture notes]
We describe the B-Tree properties.

B-Tree insertion (19:49) [lecture notes]
We describe both the bottom-up and top-down B-Tree insertion methods.

2-3-4 trees and their relation to red-black trees (17:08) [lecture notes]
We describe the 2-3-4 tree, which is simply a B-tree of order 2 (t=2), and its relationship to the red-black tree.

Lecture 17: B-Trees (cont.) and B+-Trees

B-Tree deletion (18:47) [lecture notes]
We describe the B-Tree deletion with a focus on the high-level ideas to develop a good intuition behind the design of this method.

B-Tree analysis (16:27) [lecture notes]
We derive an upper bound on the height of the B-tree, and use it to analyze the asymptotic time complexity of the B-Tree methods.

B+-tree (17:31) [lecture notes]
We briefly overview the B+-Tree data structure.

Lecture 18: Skip Lists

Intuition behind the skip list design (18:54) [lecture notes]
We motivate the design of a skip list in terms of additional navigation support for a doubly-linked list.

Skip list representation (5:24) [lecture notes]
We describe the internal representation of the skip list.

Skip list methods (17:17) [lecture notes]
We describe the skip list methods, focusing on searching, insertion, and deletion.

Lecture 19: Skip List Analysis, and Comparisons of Ordered Collection Data Structures

Skip list analysis (30:55) [lecture notes]
We analyze the time complexities of the skip list methods.

Relationship between a skip list and B+-tree (3:50) [lecture notes]
We describe the relationship between a skip list and B+-tree.

Comparison and trade-offs among ordered collection data structures (8:39) [lecture notes]
We systematically compare the ordered collection data structures.

Lecture 20: Digitized Ordered Collections, Spatial Collections/k-d Tree

Building a trie (16:02)
This video shows a short visualization of building a trie.

Digitized ordered collection ADT and data structures (33:43) [lecture notes]
We describe the digitized ordered collection abstract data type, and the illustrate by example the trie, compact trie, compressed trie, suffix tree, and indexing trie data structures..

Spatial collection ADT (9:37) [lecture notes]
We describe the spatial collection abstract data type.

kd-tree data structure (14:34) [lecture notes]
We describe the kd-tree data structure for the spatial collection.

Quad tree data structure overview(3:26) [lecture notes]
We briefly describe the quad tree data structure for the spatial collection.

Lecture 21: Graph Problems and Graph Representations

Introduction to graphs (23:50) [lecture notes]
We introduce graphs and describe a variety of problems that are best modeled by graphs.

Graph representations (64:38) [lecture notes]
After reviewing the different types of graphs, we discuss the adjacency list, adjacency set, and adjacency matrix representations and the trade-offs between them.

Lecture 22: Breadth-First Search (BFS)

Breadth-first search (51:17) [lecture notes]
We consider the problem of finding the shortest path in an unweighted path using breadth-first search (bfs). We also introduce the shortest path tree as a way to represent the output from breadth-first search.

Lecture 23: Dijkstra's Shortest Path Algorithm

Dijkstra's single-source shortest path algorithm (38:55) [lecture notes]
We present Dijkstra's algorithm to find the shortest path in a weighted graph (with non-negative edge weights) to all vertices from a specified source vertex.

Short overview of Dijkstra's algorithm (11:32) [lecture notes]
A closing overview of Dijkstra's single-source shortest path algorithm.

Guidance on the design for Project 4 -- Travel Agent Application of Dijkstra's Shortest Path Algorithm (22:37) [lecture notes]
An overview of the internals of the implementation for the last project.

Lecture 24: Greedy Tree Builder

Minimum spanning tree problem definition (4:15) [lecture notes]
We define the minimum spanning tree problem.

Maximum bottleneck problem definition (3:11) [lecture notes]
We define the maximum bottleneck problem.

Greedy tree builder (55:41) [lecture notes]
We define a generalization of Dijkstra's single-source shortest path problem and also Prim's minimum spanning tree problem. We discuss how the greedy tree builder can be used to solve either of these two problems, and also to solve the maximum bottleneck problem.

Time complexity analysis of the greedy tree builder (15:45) [lecture notes]
We analyze the time complexity of the greedy tree builder (and hence of both Dijkstra's single-source shortest path algorithm and Prim's minimum spanning tree algorithm) in terms of the time complexity of the priority queue ADT methods that dominate the time complexity of the greedy tree builder.

Lecture 25: Prim's and Kruskal's Minimum Spanning Tree (MST) Algorithm

Correctness argument for Dijkstra's algorithm (23:35) [lecture notes]
We argue that Dijkstra's single-source shortest path algorithm is correct.

Kruskal's minimum spanning tree algorithm (11:48) [lecture notes]
We present Kruskal's minimum spanning tree algorithm and prove its correctness. We also briefly introduce the union-find data structure as a way to efficiently implement Kruskal's algorithm. Finally, we analyze the time complexity of Kruskal's algorithm and briefly discuss it in relation to Prim's algorithm.

Lecture 26: DFS and Topological Sort

Depth-first search (33:43) [lecture notes]
We present depth-first search (dfs).

Topological sort (30:14) [lecture notes]

We present how depth-first search can be used to order a set of processes in a precedence graph, and prove that the algorithm is correct. This problem is called topological sort since it is trying to order (or sort) the processes in a way that adheres to the given precedence constraints.

Strongly connected components (5:51) [lecture notes]
We define a strongly connected components, and then briefly describe how depth-first search can be used to solve this problem.

Lecture 27: Garbage Collection Algorithms

What is garbage collection? (11:56) [lecture notes]
We discuss memory management and how one defines which memory cells are garbage.

Mark-and-sweep garbage collection algorithm (51:02) [lecture notes]
We describe the mark-and-sweep garbage collection algorithm, including in-place depth-first search and why that is so important for this application. We close with a brief analysis of the mark-and-sweep algorithm and an example where novice Java programmers often unnecessarily create garbage.

Copying garbage collection algorithm (14:16) [lecture notes] We briefly overview the copying collection garbage collection algorithm and discuss the trade-off between mark-and-sweep and copying collection.

Lecture 28: Course Review

Graphs (15:23) [lecture notes]
We review the portion of the course on graphs and graph algorithms.

ADT taxonomy (11:59) [lecture notes]
We review the ADT taxonomy introduced and used throughout this course to select the best ADT(s) for a given application.

Data structures (13:12) [lecture notes]
We review the data structures covered in this course.

Other topics (7:05) [lecture notes]
We review the remaining topics in this course.

A Practical Guide to Data Structures and Algorithms Using Java

Sally A. Goldman and Kenneth J. Goldman
Washington University in St. Louis

Projects

Project 1: Closest Pair of Points (provided code)

Project description and code to support students in implementing and performing some benchmarking for algorithms to compute the closest pair of points in the plane, including the divide-and-conquer algorithm. The goals of this lab are to:

- Ensure that the students understand the divide-and-conquer closest pair algorithm described in the supporting handout.
- Provide the students with a hands-on understanding of the practical benefits of designing more sophisticated algorithms, in particular, using the divide-and-conquer design technique.
- Allow the students to design their own algorithm to solve this problem and compare its performance to the divide-and-conquer algorithm.
- As one student said, "Lab 1 was a good introduction to what the class was trying to teach. It got me thinking in the right mindframe for the rest of the class." Another student said, "I learned about divide and conquer algorithms."

Project 2: Bloom Filters for Comparing Genomic DNA Sequences (provided code, provided data)

Project description and code to support students in learning about and implementing a Bloom filter and combining it with a standard mapping implementation to improve the efficiency for comparing genomic DNA sequences to find important regulatory sites.

- Introduce Bloom filters, and stress the importance of being prepared to learn about a new data structure as needed for an application.
- Expose students to an application where hash tables are used to make a more efficient algorithm.
- Provide an exciting application area and actual data for students to see how data structures can really make a difference in our ability to solve imporatant real-life problems.
- As one student summarized, "I learned the importance of efficient data structures and that specialty data structures can be very useful." Another said, "This was a really cool way to help us understand hashtables and why they are useful."

Project 3: Queries on a Collection of Historical Events (provided code, provided data)

Project description and code to support students in implementing application to maintain a set of historical events. The goals of this lab are to:

- Help students gain experience in using the ADTs and data structures we have been studying to implement a basic system that allows a user to pose simple queries about a collection of historical events.
- Provide students with experience in the design process from the application to a working implementation for an application that requires multiple data structures to satisfy its needs.
- Help students become more familiar with API and use for the libraries on the CD included with the book so that they can use it to help develop applications more efficiently.
- As one student nicely summarized, "It was helpful in learning to quickly design and implement an application." Another student said, "I learned to choose data structures to fit the application's specifications."

Project 4: Shortest Paths for Travel Agent Application (provided code, provided data)

Project description and code to support students in implementing application to find the route (i.e., sequence of flights) from a given airport at a provided start time to a desired destination airport in the shortest amount of time. The goals of this lab are to:

- Bring together several of the topics we have studied this semester to create a solution to an interesting real-life problem from "scratch."
- Have the students implement a tagged binary heap in which element placed in the binary heap can be tracked. The ability to track elements is necessary for the implementation of the shortest path algorithm.
- Help students to understand that the graph representation of the flights between the airports can simply be represented by maintaining a list of outgoing flights from each airport. (This is just an adjacency list representation of the graph.)
- Make sure that students have a deep enough understanding of Dijkstra's shortest path algorithm to create the variation of it that is required to address the layover cost that occurs when the traveller is waiting at an intermediate airport between flights.
- As one student nicely summarized, "It was a great learning experience because it taught you had to apply a lot of the concepts we learned to building a program from the ground up."
- Another student said "I think overall it was a good combinaton of learning in order to show how much we had learned over the semester."

Solutions for these projects are available to qualifying instructors from the book publisher.

A Practical Guide to Data Structures and Algorithms Using Java

Sally A. Goldman and Kenneth J. Goldman
Washington University in St. Louis

Homework Problems

Practice Problems for Homework Excercise 1 (solutions)
: These practice problems help the students understand asymptotic notation, to determine the asymptotic time complexity of fragements of code with nested loops, and to be able to compute the asymptotic time complexity of a divide-and-conquer algorithm using the master method, and to understand the relationship between functions to determine which algorithm is asymptotically best.

Homework Excercise 1
: These homework problems are designed to help students be able to analyze divide-and-conquer algorithms by first creating a recurrence relation for the asymptotic time complexity, and then solving the recurrence with the master method, and also (with guidance) to create new divide-and-conquer algorithms.

Practice Problems for Homework Excercise 2 (solutions)
: These practice problems reinforce the lectures on quicksort, positional collections, an introduction to expected-time complexity, and an introduction to using the basic adversary lower bound technique.

Homework Excercise 2
: These homework problems reinforce the lectures on quicksort, positional collections (particularly, to help students understand the trade-offs between circular arrays, dynamic arrays, singly-linked lists, and doubly-linked lists), an introduction to expected-time complexity, and an introduction to using the basic adversary lower bound technique.

Practice Problems for Homework Excercise 3 (solutions)
: These practice problems reinforce the lectures on linear-time sorting. They help students to understand the gains in efficiency that can be acheived by appropriately selecting the based for each digit when using radix sort.

Homework Excercise 3
: This homework provides additional practice in applying the adversary lower bound technique. It provides very simple design problems to help students start thinking about which ADT best supports the needs of an application. Finally, this homework reinforces the lecture on linear-time sorting algorithms, providing problems of varying degrees of difficulty working with the basic design of radix sort.

Practice Problems for Homework Excercise 4 (solutions)
: These practice problems reinforce the lectures on the binary heap, red-black-tree.

Homework Excercise 4
: These homework problems reinforce the lectures on the binary heap, red-black-tree data structures, and help students understand how to apply them to a real-world problem.

Practice Problems for Homework Excercise 5 (solutions)
: These practice problems reinforce the lectures on the B-trees and help students to understand the relationship between a red-black tree and 2-3-4 tree.

Homework Excercise 5
: These homework problems reinforce the lectures on skiplists, B-trees, B+-trees, tries, k-d trees, and graph representations.

Practice Problems for Homework Excercise 6 and the Final (solutions)
: These practice problems reinforce the lectures on graph representations and graph algorithms.

Homework Excercise 6
: These homework problems reinforce the lectures on graph algorithms to help the students both learn the algorithms and to formulate real-world problems as graph problems.

Solutions to these homework excercises are available to qualifying instructors from the book publisher.