Schedule

Help Center

Overview

Each Friday at 12:01pm EST, we will release the course materials for the week: two lectures, two sets of exercises, a programming assignment, and two sets of job interview questions.

- Exercises: due two weeks after they are released.
- Programming assignments: due two weeks after they are released.
- Job interview questions: for your own enrichment and not assessed.

In the final week, the programming assignment is replaced by a final exam.

Week 1

We begin our study of algorithms with a motivating example and an overview of the use of the scientific method for studying algorithm performance.

Lecture: Union-Find. We illustrate our basic approach to developing and analyzing algorithms by considering the dynamic connectivity problem. We introduce the *union-find* data type and consider several implementations (quick find, quick union, weighted quick union, and weighted quick union with path compression). Finally, we apply the union-find data type to the percolation problem from physical chemistry.

Lecture: Analysis of Algorithms. The basis of our approach for analyzing the performance of algorithms is the scientific method. We begin by performing computational experiments to measure the running times of our programs. We use these measurements to develop hypotheses about performance. Next, we create mathematical models to explain their behavior. Finally, we consider analyzing the memory usage of our Java programs.

Exercises. Drill exercises on the lecture material.

Programming Assignment: Percolation. Your programming assignment will give you an opportunity to apply these concepts to a fundamental problem in physical chemistry. It is the first of many examples where a good algorithm—in this case, weighted quick union—makes the difference between being able to efficiently solve a problem and not being able to address it at all.

Job Interview Questions. Algorithmic interview questions based on the lecture material.

Suggested Readings. Section 1.4 and 1.5 in *Algorithms, 4th edition*.

Week 2

You may be familiar with several of the algorithms and data structures that we consider this week, but perhaps not with our approach to data abstraction and Java language mechanisms for implementing them, so it's worthwhile to pay close attention. In the week's first lecture, we consider robust implementations for stacks and queues. In the week's second lecture, we begin our study of sorting algorithms. In both cases, we consider applications that illustrate the efficacy of careful modular programming when implementing algorithms.

Lecture: Stacks and Queues. We consider two fundamental data types for storing collections of objects: the *stack* and the *queue*. We implement each using either a singly-linked list or a resizing array. We introduce two advanced Java features—generics and iterators—that simplify client code. Finally, we consider various applications of stacks and queues ranging from parsing arithmetic expressions to simulating queueing systems.

Lecture: Elementary Sorts. We introduce the sorting problem and Java's Comparable interface. We study two elementary sorting methods (*selection sort* and *insertion sort*) and a variation of one of them (*shellsort*). We also consider two algorithms for uniformly *shuffling* an array. We conclude with an application of sorting to computing the convex hull via the *Graham scan* algorithm.

Exercises. Drill exercises on the lecture material.

Programming Assignment: Deques and Randomized Queues. Your programming assignment will involve developing implementations of two conceptually simple "collection" data types—the *deque* and the *randomized queue*---which are quite useful in practice. Properly implementing these data types will require using a linked data structure for one and a resizing array for the other.

Job Interview Questions. Algorithmic interview questions based on the lecture material.

Suggested Readings. Section 1.3 and 2.1 in *Algorithms, 4th edition*.

Week 3

Our lectures this week are based on two classic algorithms that were invented over 50 years ago, but are still important and relevant today, as implementations of one or both of them are found in virtually every software system and research on new variants of these classic methods is ongoing. Our treatment ranges from the mathematical models that explain why these methods are efficient to the details of adapting them to real-world applications on modern systems.

Lecture: Mergesort. We study the *mergesort* algorithm and show that it guarantees to sort any array of N items with at most $N \lg N$ compares. We also consider a nonrecursive, bottom-up version. We prove that any compare-based sorting algorithm must make at least $\sim N \lg N$ compares in the worst case. We discuss using different orderings for the objects that we are sorting and the related concept of stability.

Lecture: Quicksort. We introduce and implement the *randomized quicksort* algorithm and analyze its performance. We also consider randomized quickselect, a quicksort variant which finds the kth smallest item in linear time. Finally, consider 3-way quicksort, a variant of quicksort that works especially well in the presence of duplicate keys.

Exercises. Drill exercises on the lecture material.

Programming Assignment: Collinear Points. Your programming assignment is a typical example of a problem that could not be solved without a fast sorting algorithm, properly applied. It is a classic problem in computational geometry: Given a set of points in the plane, design an algorithm to find all line segments that contain 4 or more points.

Job Interview Questions. Algorithmic interview questions based on the lecture material.

Suggested Readings. Section 2.2 and 2.3 in *Algorithms, 4th edition*.

Week 4

This week we are going to introduce two fundamental data types, address the challenges of developing algorithms and data structures that can serve as the basis of efficient implementations, and try to convince you that such implementations enable solution of a broad range of applications problems that could not be solved without them.

Lecture: Priority Queues. We introduce the priority queue data type and an efficient implementation using the *binary heap* data structure. This implementation also leads to an efficient sorting algorithm known as *heapsort*. We conclude with an applications of priority queues where we simulate the motion of N particles subject to the laws of elastic collision.

Lecture: Elementary Symbol Tables. We define an API for *symbol tables* (also known as *associative arrays*) and describe two elementary implementations using a sorted array (binary search) and an unordered list (sequential search). When the keys are Comparable, we define an extended API that includes the additional methods min, max floor, ceiling, rank, and select. To develop an efficient implementation of this API, we study the *binary search tree* data structure and analyze its performance.

Exercise. Drill exercises on the lecture material.

Programming Assignment: 8-Puzzle. Your programming assignment is to implement the famous A* search algorithm to solve a combinatorial problem, and to substantially speed it up with an efficient priority queue implementation.

Job Interview Questions. Algorithmic interview questions based on the lecture material.

Suggested Readings. Section 2.4, 3.1, and 3.2 in *Algorithms, 4th edition*.

Week 5

Can we guarantee fast search, insert, delete, min, max, floor, ceiling, rank, and select in a symbol table with Comparable keys? This week, you will learn that the answer to this question is a resounding Yes! and that a relatively compact implementation can do the job. Then, we consider applications and generalizations of binary search trees to problems in computational geometry.

Lecture: Balanced Search Trees. In this lecture, our goal is to develop a symbol table with guaranteed logarithmic performance for search and insert (and many other operations). We begin with *2-3 trees*, which are easy to analyze but hard to implement. Next, we consider *red-black binary search trees*, which we view as a novel way to implement 2-3 trees as binary search trees. Finally, we introduce *B-trees*, a generalization of 2-3 trees that are widely used to implement file systems.

Lecture: Geometric Applications of BSTs. We start with 1d and 2d *range searching*, where the goal is to find all points in a given 1d or 2d interval. To accomplish this, we consider *kd-trees*, a natural generalization of BSTs when the keys are points in the plane (or higher dimensions). We also consider *intersection problems*, where the goal is to find all intersections among a set of line segments or rectangles.

Exercises. Drill exercises on the lecture material.

Programming Assignment: Kd-Trees. Your programming assignment is to implement kd-trees, which can form the basis for fast search/insert in geometric applications and in multidimensional databases.

Job Interview Questions. Algorithmic interview questions based on the lecture material.

Suggested Readings. Section 3.3 in *Algorithms, 4th edition*.

Week 6

We conclude the course by considering hash tables, a data structure that achieves constant-time performance for core symbol table operations, provided that search keys are standard data types or simply defined. Then we consider several fundamental (and useful) examples of symbol-table clients.

Lecture: Hash Tables. We begin by describing the desirable properties of hash function and how to implement them in Java, including a fundamental tenet known as the *uniform hashing assumption* that underlies the potential success of a hashing application. Then, we consider two strategies for implementing hash tables—*separate chaining* and *linear probing*. Both strategies yield constant-time performance for search and insert under the uniform hashing assumption. We conclude with applications of symbol tables including sets, dictionary clients, indexing clients, and sparse vectors.

Exercises. Drill exercises on the lecture material.

Final exam. The final exam is cumulative and designed to make sure you understand how each algorithm works and when it is effective. The final will not involve Java programming.

Job Interview Questions. Algorithmic interview questions based on the lecture material.

Suggested Readings. Section 3.4 in *Algorithms, 4th edition*.

Beyond

If you're interested in continuing, please be sure to sign up for Algorithms, Part II, where you'll learn about graph algorithms and string processing. Many of the classic algorithms in these areas are nothing short of ingenious, as they provide compact and elegant solutions to important problems that would seem very difficult to solve at first blush. We also cover the important idea of placing these algorithms in context among problems that actually do seem to be difficult to solve, with implications for practical and theoretical computer science.

Created Sat 14 Jan 2012 12:33 AM EST

Last Modified Sat 15 Aug 2015 10:37 PM EDT