Overview

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. (*Sorry, we are still waiting for Coursera to migrate the exercises from the old platform.*) 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*.

✓ Complete