# Data Structure in Java - Final Review

Paul Blaer

April 26, 2017

## Weiss Textbook Chapters

- Chapter 1 (entirely)
- Chapter 2 (entirely)
- Chapter 3 (entirely)
- Chapter 4.1, 4.2, 4.3, 4.4, 4.6 (note, 4.6 covers tree traversals the way we covered them in class and makes for good supplemental reading) and 4.8 (note, 4.8 covers java sets and maps, this is supplemental reading only)
- Chapter 5.1, 5.2, 5.3, 5.4, 5.5, and 5.6 (note, 5.6 covers java hash implementation, this supplemental reading only)
- Chapter 6.1, 6.2, and 6.3
- Chapter 7.1, 7.2, 7.3 (skim only), 7.5, 7.6, 7.7 (you can omit 7.7.6)
- Chapter 8 - (skim only, just have sense for how unions and finds are relevant to Kruskal's algorithm)
- Chapter 9 - 9.1, 9.2, 9.3 (9.3.1 and 9.3.2 and skim 9.3.3), 9.5, 9.7

## Material from Outside the Textbook

- Tower of Hanoi
- Josephus Problem (this was an example of Queue's, it's really not import on its own for the final, but I'm mentioning it here for completeness)
- Selection Sort (as compared to insertion sort, and as the degenerate of quicksort)

## General Concepts

- Abstract Data Types vs. Data Structures.
- Recursion.
- Basic proofs by induction.

## Java Concepts

- Basic Java OOP: Classes / Methods / Fields. Visibility modifiers.
- Generics.
- Inner classes (static vs. non-static).
- Interfaces.
- `Iterator`/`Iterable`.
- `Comparable`.

## Analysis of Algorithms

- Big-O notation for asymptotic running time: $O(f(n))$, $\Theta(f(n))$, $\Omega(f(n))$.
- Typical growth functions for algorithms.
- Worst case, best case, average case.
- *Skills:* Compare growth of functions using big-O notation. Given an algorithm (written in Java), estimate the asymptotic run time (including nested loops and simple recursive calls).
- Basic understanding of recursion (Towers of Hanoi, Binary Search) and run-time behavior of recursive programs. Logarithms in the run-time. Tail recursion.

## Lists

- List ADT, including typical List operations.
- ArrayList:
  - running time for insert, remove, get, contains at different positions in the list.
  - increasing the array capacity when the array is full.
- LinkedList:
  - single vs. doubly linked list.

  - running time for insert, remove, get, contains at different positions in the list.
  - sentinel (head/tail) nodes.
- *Skills:* Implement iterators. Implement additional algorithms on lists (removing duplicates, intersection, etc.).
- Lists in the Java Collections API.

## Stacks and Queues

- Stack ADT and operations (push, pop, peek). LIFO.
- Queue ADT and operations (enqueue, dequeue). FIFO.
- All operations should run in $O(1)$.
- Stack implementation using List data structures, and directly on an array.
- Stack applications:
  - symbol balancing, detecting palindromes, ...
  - reordering sequences (in-order to post-order, train cars,...).
  - storing intermediate computations on a stack (evaluating post-order expressions).
  - building expression trees.
- Tail recursion.
- Queue implementation using Linked List.
- Stacks and Queues in the Java Collections API (java.util.LinkedList supports all stack operations).
- *Skills:* Implement stacks and queues. Use stacks and queues in applications.

## Trees

- Tree terminology (parent, children, root, leafs, path, depth, height)
- Different tree implementations (one field per child, general trees: siblings as linked list).
- Binary trees:
  - full / complete/ perfect binary trees.
  - tree traversals: in-order, pre-order, post-order.

  - expression trees - pre-fix, post-fix (reverse Polish notation), and in-fix notation.
  - Constructing an expression tree using a stack.
  - Relation between number of nodes and height of a binary tree.
  - Inductive proofs over binary trees.
- *Skills:* Perform tree traversals on paper. Implement different tree traversals using recursion (different versions). Use these traversals to implement operations on trees. Convert between in-fix, post-fix, pre-fix notation using a tree. Simple inductive proofs for tree properties.

### Binary Search Trees

- BST property.
- BST operations: contains, findMin, findMax, insert, remove
- Run-time performance of these operations, depending on the height of the tree.
- Lazy Deletion
- *Skills:* Perform BST operations on paper.

### AVL Trees

- Balanced BSTs. AVL balancing property.
- Maintaining AVL balance property on insert:
  - Outside imbalance, single rotation.
  - Inside imbalance, double rotation.
  - Verifying that a tree is balanced. Finding the location of an imbalance (bottom-up).
- *Skills:* Perform AVL rotations on paper, detect imbalances.

### Hash Tables

- Hash functions.
- Collision Resolution Strategies
  - Separate Chaining
  - Linear Probing
  - Quadratic Probing
  - Double Hashing (this was on the homework, but isn't so likely to end up on the final)

- Rehashing (note lazy deletions required, prior to the rehash for probing methods)
- HashSets and HashMaps in the Java API.
- *Skills:* Do hash inserts and rehashes on tables with pen and paper. Understand the consequences of the various collision resolution strategies. Be able to recognize a bad hash function.

## Priority Queues

- Simple implementations.
- Min-heaps vs. Max-heaps.
- Binary Heaps.
- Tree and Array representations of Binary Heap.
- Properties of a binary heap including the complete tree and heap order property.
- The insert and deleteMin/deleteMax operations.
- Percolate up and percolate down.
- Calculating the parent and child nodes in the array representation.
- The buildheap operation and its cost (not the proof of the cost though).
- *Skills:* Perform inserts and removes on a binary heap. Be able to perform a buildheap operation. Be able to recognize whether the properties of the heap are being adhered to.

## Sorting

- Insertion sort
- Selection sort (not in the book)
- Heap Sort (we covered this at the end of the heap section of the class)
- Merge Sort (and the merge operation)
- Quick Sort, partitioning, and pivot selection (such as median-of-three).
- *Skills:* Know the big-O costs of all of the algorithms listed (best and worst case if different, average case as well with Quick Sort). Be able to perform the sort algorithms on paper. Understand when it might be appropriate to use one algorithm over the other.

## Graphs

- Basic definitions - vertices and edges, directed vs. undirected, weighted vs. unweighted, DAGs, connectivity (strong vs. weak in the case of directed graphs), complete graphs. Dense vs. sparse.
- Graph representations: adjacency matrix vs. adjacency lists.
- Topological Sorting.
- Single-source unweighted shortest paths.
- Dijkstra's algorithm (single-source weighted shortest paths).
- The consequences of weighted edges.
- Minimum Spanning Trees:
  - Prim's algorithm (almost identical dijkstra's with a different cost update, prims only looks at the cost of the next edge).
  - Kruskal's algorithm (be familiar with how disjoint sets are used in this)
- P vs. NP and NP-completeness
- The Traveling Salesperson Problem (both the NP-complete version and the NP-hard version)
- Nearest Neighbor Approximation vs. Brute Force Solution to TSP.