15
CSCI 136: Spring 2017
May 9

Final Exam Study Guide

Your final will be a "closed book" self-scheduled. From the registrar's webpage, a self-scheduled exam:

> may be taken starting with the Reading Period and may be picked up between the hours of 8:30 a.m. and 6:00 p.m. from the monitor in the Registrar's Office on the second floor of Hopkins Hall, on any day (including Saturday and Sunday) May 13 through May 21. Self-Scheduled exams must be returned to the monitor within two and one-half hours after they are taken out. The last day for taking a self-scheduled exam is Sunday, May 21st.

You are responsible for anything we covered in class or in lab, everything in the assigned reading from *Java Structures*, and the handouts/labs. The exam is cumulative, but it will heavily weight topics from the second half of the course. However, we used arrays, Vectors, and Lists to implement many data structures; we used and big-O notation to evaluate and compare data structures; we used recursion to traverse trees; etc.. The second half of the semester built heavily on previous topics.

The following non-exhaustive list may be helpful in reminding you about some of the key topics we have covered:

- **Pre-Midterm**
  - Java syntax, as we have used it in our programming assignments.
  - Classes, abstract classes, and interfaces and their respective roles.
  - Information hiding (abstraction) and why it's good.
  - Extending classes with inheritance.
  - Generic classes and their use
  - Pre- and post-conditions, and assertions.
  - The meaning of `static` (and non-static) as applied to variables and methods
  - Vector, its implementation in the structure5 package, and its methods.
  - Complexity: Big "O" definition.
    - Determining the asymptotic behavior of mathematical functions
    - Determining the time and space complexity for a given algorithm.
    - Worst and best case analysis.
  - Linear and binary search.
  - Recursion and induction.
  - Sorting.
    - Bubble sort, selection sort, insertion sort, merge sort, quicksort, heapsort.
    - Using `Comparator/Comparable` for sorting.
  - Linked lists: Singly, Doubly, Circularly, and Chain-style list
- **Post-midterm**
  - Stacks (LIFO)
    - List and Vector implementations
    - Relationship with recursion and graph/tree traversals (DFS)

- Queues (FIFO)
    - List, Vector, and fixed-size array implementations
    - Relationship to graph/tree traversal strategies (BFS)
- Priority queues
    - OrderedVector Implementation
    - Heap implementation
        - heap property
        - array representation and tree fullness/completeness
        - heap insert/remove
- Trees
    - Array/Vector-based representation
    - Recursively-defined, pointer-based representation
    - Binary search trees
    - Traversing trees (In-order, post-order, pre-order; Breadth-first, depth-first)
    - Tree (un)balance
- Iterators
- Bitwise operations
- Graphs
    - Directed/undirected
    - Weighted/unweighted
    - Adjacency List representation
    - Adjacency Matrix representation
    - Reachability/traversal (Breadth-first, depth-first)
- Hashtables
    - Hashing function
    - Load factor
    - Managing collisions (linear probing/external chaining)

Our goal is to test concepts, so it is not important to memorize the exact code or method signatures for every data structure. Pseudo-code and descriptive variable/method names are enough to demonstrate understanding. However, it *is* important to know the types of operations that different data structures do/do not support. For example, we cannot access arbitrary elements in a queue: we can only add to the back and remove from the front.

Answers to odd-numbered book questions can be found in the appendix, and we have posted a sample exam on the course webpage. Good luck!