Do Practice HW Problems!

Algorithm Analysis

- Be able to look at code and determine if it's O(1), O(log(n)), O(n), etc.
- Given N_1, T_1, O(N) – predict T_2 from N_2 (and vice versa)
    - In other words, given the time it takes to process N_1 items, be able to determine the time it would take to process N_2 items, or determine the number of items that can be processed in a different amount of time.

For each of the Data Structures covered so far (see below), know:

- All relevant Operations
    - O(???) for each
        - Worst case, Best case Average case
    - Theoretical (non-Python) Algorithms for each (including recursive ones), and including drawing pictures of linked-lists, arrays, and trees.
    - Be able to Read/Write/debug Python code for each operation
- Data structures:
    - Binary Search Tree
    - Priority Queue (Binary Heap implementation)
    - Hash Table

Data Structures since midterm 1:

Binary Search Tree, as implemented in Lab 5

- Operations:
    - add
    - remove/delete
    - find/get/contains
    - find min/max
    - tree height

Priority Queue, as implemented in Lab 7

- Array-based binary heap implementation
- Operations:
    - enqueue
    - dequeue
    - bottom-up heap construction
    - percolate up
    - percolate down
    - heap sort

Hash Table

- Array-based implementation
    - Separate Chaining
    - Linear Probing
    - Quadratic Probing
    - Load Factor – restrictions for each type of collision resolution strategies
- Operations:
    - add
    - find/get/contains
    - remove

Sorts – Basic implementation

- Sorts:
    - Insertion
    - Selection
    - Merge
    - Heap
    - **Tsort – Just know the basic algorithm**

- Know:
    - General Algorithm
    - Big-Oh for best, average, worst (time complexity)
    - Be able to read and understand code

Code Reading/Writing

- Be able to read/write Python code for operations of the covered data structures, especially for the operations implemented in labs/projects.
    - Labs: 5, 6, 7
    - Projects: 3a, 3b