Day 1 will focus on things you might have to do in the "real-world" when it would be inconvenient or infeasible to look up extra information on the internet.
Code modeling and design decisions will be the focus, but there will be other questions.

Day 2 will focus on things where memorization is not expected, or we have had less time to practice, and a cheat sheet would be of use.

Compared to the midterm, expect fewer mechanical questions, and more questions applying the concepts you've learned.

Both days:

- Graph fundamentals
  - definitions (e.g., directed, undirected, weighted, unweighted, walks, paths, cycles, self-loops, parallel edges, trees, DAGs, strongly, weakly connected, components etc.)
- ADT/Data structure fundamentals
  - For all ADTs and data structures (both before and after the midterm) how they work and the associated running times for all important operations
- Coding Projects
  - Implementation of each data structure
  - Best / in-practice / worst case runtime for each method of each data structure
  - Testing (coming up with test cases (see Exercise 2 problem 4))
  - Debugging (locating code with bugs (see Exercise 2 problem 4))
- Design Decisions
  - Given a scenario, what ADT, data structure implementation and/or algorithm is best optimized for your goals?
    - What is unique or specialized about your chosen tool?
    - How do the specialized features of your chosen tool contribute to solving the given problem scenario?
    - How expensive is this tool and its features in terms of runtime and memory?
  - Given a scenario, what changes might you make to a design to better serve your goals?
- Pre-midterm topics
  - all ADTs and data structures
    - We will expect you to know the running times and how operations work.

    - But will not ask you to mechanically execute operations on data structures already covered on the midterm (e.g. we won't ask you to do an AVL rotation, but would expect you to remember you only need to do at most two rotations on an insertion)
  - Asymptotic analysis
    - Code Modeling (including recurrences)
    - Complexity Classes
    - Big O, Omega, Theta
    - Master Theorem

Day 1

- Heap Details
  - Visual tree representation and heap properties
  - Array implementation
  - Execute any of the heap algorithms we've discussed (including buildHeap)
- P vs. NP
  - What "P" and "NP" stand for
  - What P, NP, and NP-complete classes are
  - What should you do if you want to solve an NP-complete problem?
  - The current state of P vs. NP (e.g. it's not solved, but it's widely believed they are not equal)
  - Understand what a reduction is

Day 2

- Sorting
  - Sorting algorithm properties (stable, in-place)
  - insertion sort, selection sort, heap sort, merge sort, quick sort
  - Runtimes of all of the above (best, worst, and in-practice) and their properties
  - Given a scenario be able to choose one over the other
- Graphs
  - definitions (e.g., directed, undirected, weighted, unweighted, walks, paths, cycles, self-loops, parallel edges, trees, DAGs, etc.)

  - implementations (Adjacency list, Adjacency matrix, and their pros and cons)
  - traversals (BFS and DFS)
    * When you use one traversal over another.
  - For each of the following algorithms, know what problem the algorithm solves, what it's running time is, and how the algorithm can fail (e.g. Dijkstra's fails with negative weight edges):
    * Dijkstra's
    * Prim's
    * Kruskal's
    * Topological Sort
    * Strongly Connected Components algorithm
  - Additionally, for the following algorithms, be able to execute the algorithm by hand:
    * BFS
    * DFS
    * Dijkstra's
    * Prim's
    * Kruskal's
  - Additionally, for the following problems, be able to find a correct output by hand (we won't ask you to execute the algorithm from class, though you may)
    * Topological Sort
    * Strongly Connected Components
  - Framing/modeling problems with graphs (like Exercise 5 problem 3)
- Disjoint sets details
  - Visual forest representation and array implementation
  - Implementations of Union, FindSet, and MakeSet operations

The following topics will NOT be covered in the final

- Tree Method / Unrolling for finding the closed form of a recurrence
- Radix Sort
- Writing Java / JUnit syntax.
  - Java code and pseudocode for you to read may show up on the exam, but if you need clarification on syntax you may ask.