Midterm 1 Study Guide

This midterm covers chapters 5 through 7. Here is a list of the main ideas:

1. What is the general approach in the **divide-and-conquer** technique? How does it differ from the *decrease-and-conquer* technique?
2. What does the **master theorem** say about the solution to the recurrence relation $T(n) = aT(n/b) + f(n)$? How does this relate to the divide-and-conquer technique? What do the parameters a, b, $f(n)$ represent in the divide-and-conquer context?
3. Be able to use the master theorem (e.g. exercise 5.1.5)
4. Describe the **mergesort** algorithm, and outline the general algorithm for it.
   - In particular, be able to write down the both the Mergesort and the Merge algorithms of page 172.
   - Use the master theorem to determine the worst case time efficiency of mergesort.
   - Is mergesort a stable algorithm?
5. Describe the overall plan for the **quicksort** algorithm.
   - Describe what a **partition** is.
   - Describe the two partition algorithms we saw, Lomuto (page 159) and Hoare (page 178). You do NOT need to be able to write the exactly, but you do need to be able to describe how they achieve their partition.
   - Explain how the time efficiency of the quicksort algorithm depends on a good choice of a pivot, then describe some possible approaches to pivot selection.
6. Explain what *internal* and *external* nodes are in a binary tree and why the number of external nodes is always one more than the number of internal nodes.
7. Be able to write down common divide-and-conquer algorithms for binary trees such as determining the tree height, the number of external nodes, and whether a tree is in fact a binary *search* tree.
8. Be able to name, describe and use the three traversal approaches to binary trees.
9. What are the three different forms that **transform-and-conquer** algorithms take? Describe each form.
10. Show instances where **presorting** an array can lead to more efficient algorithms. In particular, be able to explain how presorting can help with finding if there are any duplicate elements in the array and also to find what the smallest absolute difference between array elements is. Determine the time efficiency of solutions to these problems using presorting and also using a more brute-force approach.

11. **AVL Trees**:

    - What is the definition of an AVL Tree? What is the **balance factor** of a node?
    - Describe graphically the four different "rotations" that are used in the implementation of AVL trees.
    - Be able to carry out these transformations to bring a tree into the AVL form.

12. **2-3 Trees**: Describe the two kinds of nodes that 2-3 trees have, what further property 2-3 trees have, and how to search in a 2-3 tree.

13. **Heaps**:

    - A heap is a binary search tree with a very specific structure, satisfying two key conditions. Describe that structure.
    - How can we think of a heap instead via an array? How do the indices in the array correspond to the tree structure?
    - What are **priority queues**? Why is a heap a particularly good way to implement a priority queue?
    - Describe the heap-bottom-up construction. What is its time efficiency?
    - Describe how maximum key deletion in a heap occurs.
    - Show how the heapsort algorithm works.

14. **Hashing**:

    - Describe the main ideas behind hashing: *hash table*, *hash function*, *hash address*, *load factor*, *collisions*.
    - Describe some possible hash functions for strings and what their drawbacks are.
    - Describe how *open hashing* and *closed hashing* work and how they differ from each other. Describe the *linear probing* approach to closed hashing.