CS 367: Data Structures **Summer 2011**
Exam 2 Practice Problems

Trees

1. Turn the following fully-parenthesized algebraic expression into a parse-tree:

$$((((3+x)^3)*2)-(6/((4-y)^5)))$$

2. Describe a recursive algorithm to evaluate a parse-tree. (Assume the value stored in each node is a string. Do not assume the tree is binary.)

3. What is the BST property? Why is it important?

4. Imagine you have an empty BST. Perform the following operations, showing the tree at each intermediate step:

   - insert(7)
   - insert(5)
   - insert(6)
   - insert(15)
   - insert(10)
   - insert(20)
   - insert(16)
   - insert(17)
   - delete(15)
   - insert(8)
   - delete(16)
   - delete(8)
   - delete(7)
   - delete(5)

5. What is the worst-case runtime for `insert()` in a Binary Search Tree? How can we change the BST data structure to improve this?

6. Give a concrete, detailed example for how deletion from an AVL tree may require multiple rotations to restore balance.

7. The private `insert()`, `delete()`, and `balance()` methods of our AVL tree implementation all returned the root of the sub-tree on which they were working. Why was this?

8. What is the “shape” property of a heap? What is the “order” property? Why are these important?

Hashing

1. In programming, a "dictionary" is typically a structure for associating key → value pairs. There are many different ways a dictionary could be implemented. Compare the following implementations: Binary Search Tree, AVL Tree, hash table. Which is the best? Why? Does it depend on anything?
2. What is the average runtime for accessing values in a hash table? Under what circumstances does it achieve this runtime? Describe two things you can change to make achieving this runtime more likely.
3. What are the major concerns associating with choosing a hash function?
4. What are the pros and cons of resizing your hash table? Is it always neccessary?

Graphs

1. Give an example scenario in which a depth-first-search on a graph is superior to a breadth-first-search.
2. Give an example scenario in which a breadth-first-search on a graph is superior to a depth-first-search.
3. Imagine you have a maze implemented as an undirected graph such that each spot in the maze is a node with edges leading to all spots from which it is accessible (its neighbors). What would be more appropriate for solving this maze: depth-first-search or breadth-first-search? Why? Do both work? For large mazes, how do their running times compare?
4. Turn the following adjacency matrix into a graph (imagine there are six nodes, labeled A-F):

$$\begin{pmatrix} 0 & 0 & 2 & 0 & 1 \\ 2 & 0 & 0 & 8 & 1 \\ 2 & 0 & 0 & 1 & 2 \\ 0 & 4 & 0 & 1 & 0 \\ 1 & 0 & 3 & 0 & 0 \end{pmatrix}$$

5. Break out a Risk board. Connect each territory to 2-5 other arbitrary territories (does NOT have to be neighboring territories) with colored yarn. Measure each piece of yarn and label it with its length (be sure to use the metric system!). Finally, use Dijkstra's algorithm to find the shortest path to each territory from Greenland. (Alternatively, make up your own weighted graph on paper and perform Dijkstra's algorithm on it.)

Sorting

1. What is the worst sorting algorithm we've studied in this class? Why? (BE DETAILED!)
2. What is the best sorting algorithm we've studied in this class? Why? (BE DETAILED!)
3. Give a full, detailed analysis of the runtime for Merge Sort. Be sure to give best and worst case scenarios.
4. Perform a Quick Sort on the following list, being sure to show all intermediate steps: []
5. Perform a heap sort on the following list. Display the heap as both a tree and an array at each step. [18, 35, 22, 19, 14, 37, 11, 14]