CMSC 206 Final exam SAMPLE QUESTIONS, Spring 2008 (Prof. Wonnacott)

Name:

**Time limit:** none — actual exam will be limited to 3 hours

**Due date:** Do these SAMPLE questions any time before the actual exam;

**Instructions you can expect on the actual exam:**

1. Put your **name** on this exam!

2. The **actual exam** has a **three hour** time limit. As usual, you can expect that this practice exam will take more time than the actual exam.

3. The **actual exam** is **open book**; you may use either of the course textbooks, your notes from lecture, or notes you get from someone else before you start the exam. You may also look at your lab programs or the answer keys, but you may **not** run Python programs to see how they work during the exam.

4. You may, if you wish, **use a calculator** to perform simple arithmetic.

5. When you start, please **check your copy of the exam** to make sure it has all 5 questions on 7 pages (counting this one).

6. I recommend that you also **read all of the questions** before starting to answer any of them, and **budget your time** to avoid spending too much on any one question.

7. **If any question is unclear** in any way, you can send me a text page via the email address davew-phone@cs.haverford.edu (include your phone number at the start of the message). If you don’t hear from me promptly, make a note of what you found unclear and do your best to answer the question anyway.

8. **Show all your work.** If you need scratch paper or more room than is provided, use the back sides of the pages.

9. Write all your **answers on the exam**; if your final answer is not in the space below the question, **indicate** in this space **where your final answer is to be found**.

10. The **actual exam** is **not** to be shared with students who take this class in subsequent years, nor circulated in any manner.

**Note** that, in any question in which you are asked to write an answer in Python, I will grade the correctness of your idea, not the details of Python. You will not lose points for missing colons after “if”s, mixing up colons and commas in subscripts, etc., as long as your idea is still clear.

Point values are given with each problem. **Total Score:** _______ / 80

You will be asked to sign that accept full responsibility under the Haverford Honor System for the **actual exam**.
1. Specialized Trees:
   a) Draw a binary search tree with 7 values in it; draw a binary tree with 7 values that is not a valid binary search tree.
   b) Draw a balanced heap with 7 values in it; draw a binary tree with 7 values that is not a valid heap.

2. Complexity:

   Draw a directed acyclic graph showing the following functions, in which each node is a collection of functions that are the same complexity (such as \(n^2\) and \(n^2 + 1\)) and there is an arc from node \(X\) to node \(Y\) if the functions in \(X\) must have lower "big-O" complexity (so there would be an arc from \(n\) to \(n^2\) but not the other way around). The functions are: \(n, n^2, n^2 + 1, \log_2(n), 2^n, 5n^2, n^2 + n, n^2 + n, n + m, n \cdot m\).

3. Abstraction, Test suites for data types, Axioms, and Structural Induction:

   We define a “Gonczi-tree” as a binary tree in which no node has only one child (in other words, every node is either a leaf, with a value but no children, or an internal node with exactly two non-empty children).
   
   a) Draw two example Gonczi-trees, and one example of a binary tree that is not a Gonczi-tree (and indicate which is not a valid Gonczi-tree).
   b) Give a set of constructors and accessors for Gonczi-trees. Include operations to count the number of leaves and total number of nodes in a Gonczi-tree.
   c) Give a test suite for your operations in b.
   d) Give axioms for your operations in b.
   e) Describe the relationship between the number of leaves and the total number of nodes — either (i) give a precise relationship, if one exists, and prove your answer is correct (via structural induction), or (ii) give both a tight upper bound and a tight lower bound and prove that one of your bounds is correct (again, via structural induction) [For the purpose of this problem, a bound is tight if it is possible to actually achieve that value — for example, for a binary tree with \(n\) nodes, \(n - 1\) and \(n^2\) are both upper bounds on the tree height (where we count a single leaf as having height 0), but \(n - 1\) is a tight upper bound and \(n^2\) is not.]

4. Representation Invariants:

   Write a representation invariant for a hash table with “internal chaining” for which only a single hash function is used, and collisions are resolved by putting the to-be-inserted object in the next available space.

5. Abstraction Functions:

   Write an abstraction function for a class that uses a binary search tree to represent a set. In other words, you should assume you have an object with fields \(\text{value}, \text{left}, \text{and right}\), and wish to produce a string of the form \(\text{Set}(4).\text{union}(\text{Set}(6)).\text{union}(\text{Set}(5))\) (or, of course, since the order doesn’t matter, \(\text{Set}(6).\text{union}(\text{Set}(4)).\text{union}(\text{Set}(5))\), etc.)

6. Removal from a hash table: (I’m not sure if this is covered in the book; if so, try to solve it without looking at the book’s discussion of removal operations, though you may look at the other hash table operations).

   Suppose we want to provide a remove mutator for dictionary represented by a hash table. In other words, given a dictionary \(D\) that contains an entry with key \(k\), we want to be able to write \(D.\text{remove}(k)\) and thereby ensure that \(D\) no longer contains the entry with key \(k\).
   
   a) Give a test suite for this operation
b) Give axioms for this operation

c) Discuss the challenges of implementing this operation on a hash table. Discuss both the difficulty of programming the operation and the run-time complexity of the resulting operation if “collisions” (two different keys with the same hash value) are extremely rare but not impossible (in other words, if you have to write your algorithm to handle collisions, but they never occur, what would the \( O() \) complexity be?). Consider external chaining, internal chaining with a secondary hash function, and internal chaining using the next available space. Which of these approaches to collision management would you recommend? (Your answer should be consistent with your answer to Question 4.

7. An unfamiliar data type (note I wrote this from memory, so it may not correspond to the official definition — if it is internally inconsistent, please let me know):

A **trie** is a tree that can be used to store values that can be represented by a sequence of symbols from a finite alphabet (as English words can be represented by a sequence of letters of the roman alphabet, or integers by a sequence of arabic numerals). Each node in the trie is either empty or has a vector of elements corresponding to the values in the alphabet (e.g., it would have 26 elements if we were to store English words, or 6 elements if we were storing only words that could be created with the letters a-f (such as feed)). A value is present in the trie if the corresponding path from the root is present. So a trie of words made of the letters a-f that contains “a”, “fad”, “fed”, and “fee” would have a root in which the first element of the array (corresponding to “a”) refers to an empty trie (with a six-element empty array) and in which the sixth element (corresponding to “f”) refers to a trie containing “ad”, “ed” and “ee”.

Create a class for tries of strings made from the letters a-f, including a test suite, axioms, and descriptions of the complexities of each operation. You must be able to build any such trie and check to see if a value is present, but you don’t have to provide other operations (such as abstraction function or representation invariant).