212 Test 2 Information

This test will be similar in form to the first test, although there will be fewer instances where you can try a question multiple times. Hopefully there will be less likelihood that trying a question again could reduce your score, but it could still happen.

You may bring a single 8.5x11" page of notes to the test, if it is hand-written or typed (not photocopied). You may not refer to any other resources or talk to any other people while taking the test. Even if you don't get around to creating a page of notes, you may not look things up during the test.

The test will be on everything we covered in chapter 2. Here is a partial list:

Chapter 2: Elementary data structures

- Abstract data types
  - What is an ADT?
  - How do you decide what ADT your program needs?
  - What data structure should you use to implement the ADT you need, given the frequencies of use of each operation?
  - ADTs stack, queue: operations?
- Growable array-based stacks: operation runtimes?
- Trees
  - terminology: height, depth, leaves, ancestors, descendents, parents, children, etc.
  - Binary trees
  - Tree traversals: preorder, inorder, postorder
- ADT priority queue
- Heaps
  - definition,
  - operations (how they work),
  - runtimes of operations
  - Heapsort
- ADT dictionary
- Hash tables
  - hash function,
  - collision resolution: chaining, linear probing, double hashing
  - dynamic hashing

Here are some sample questions for the dead-tree form of the test; they'll be modified appropriately for the dead-electron (Moodle) format:

1. Define ADT ordered dictionary.
2. What is the runtime of the best algorithm studied for building a heap of n numbers?
3. What is the average and worst-case runtime for inserting n items into a hash table using double hashing?
4. Show how to insert the numbers 10 14 18 31 35 into a 7-element hash table using linear probing for collision resolution. Let the hash function be n mod 7.
5. Show how heapsort would sort the array 5 4 3 2 1 in place. Use a heap with the largest item at the root. Show the contents of the array after each swap.
6. Show the results of pre-, in-, and post-order traversals of the following tree: [tree given here]
7. What data structure we studied would be best for implementing a priority queue when we have a very large number of inserts but very few delete_min operations? How about when we have many min and delete_min operations but few inserts?