**Homework #5**

**Binary Tree Applications**

total points: 210

0

1. 10 10(10 points) Describe an algorithm for using a Priority Queue to sort a list of integers. You do not have to write any code, just describe the algorithm in sufficient detail to convince me that the code could be written fairly easily.

2. Give and briefly justify asymptotic bounds for the running time of the algorithm you described in #1, assuming that the backing store for the priority queue is:

a. 10 20(10 points) an unsorted array based list.

b. 10 30(10 points) a heap.

3. 20 50(20 points) Consider the heap shown below:



Apply each of the following operations, one after the other, to the above max heap. Redraw the heap after each operation has been performed.

 a. add(8)

 b. add(12)

 c. remove()

 d. remove()

4. [Optional ungraded question. Tests your knowledge of the wildcard type parameter “?”.] The GenericBox class in the linear.generic package of the COMP232-SampleCode project is a generic class with the type parameter T. How would you change the definition of the GenericBox class such that it can only store types that are numeric (e.g. of type Integer, Float, Short, Byte, Long, Double etc.). You need only give the first line of the class definition (e.g. public class GenericBox…) as your solution for this question. Hint: These classes share a common super class.

5. The CS232ArrayHeap class is designed to maintain a max heap with respect to the compareTo method implemented by the key type. The compareTo method in the String class places items into alphabetical order, thus strings later in a dictionary would come out of the heap before words earlier in a dictionary. This may be backwards for many applications. This can be addressed by defining a new type for the key that provides a compareTo method that orders the keys appropriately. This is (almost) done in the StringMinHeapKey class in the hw05 package.

a.10 60(10 points) Run the main method in the StringMinHeapKey class in the hw05 package. What exception is thrown? Why? Hint: Look at the constructor being used. Read the Javadoc for that constructor carefully. Now examine the keys being passed in and the compareTo method implementation.

b.5 65(5 points) Modify the compareTo method so that the given keys form a valid heap with “A” having higher priority than “B”, and “B” higher priority than “C”, etc. Hint: this can actually be achieved by inserting only a single character, but other solutions are also possible.

c. 10 75(10 points) Explain why your solution in part b works.

6. 25 100(25 points) Complete the add method in the CS232ArrayHeap class in your hw05 package. The No6Tests class contains tests that you can use to check your implementation of this method.

7. 25 125(25 points) Complete the adjustPriority method in the CS232ArrayHeap class in your hw05 package. The No7Tests class contains tests that you can use to check your implementation of this method.

8. 20 145(20 points) Consider the binary search tree shown below:



Apply each of the following operations, one after another, to the above binary search tree. Redraw the binary search tree after each operation has been performed.

 a. add(9)

 b. add(18)

 c. remove(4)

 d. remove(19)

9. 15 160(15 points) Complete the get method in the CS232LinkedBinarySearchTree class in your hw05 package. The No9Tests class contains tests that you can use to check your implementation of this method. Suggestion: Writing a helper method here like we did with get in LinkedBinaryTree will help you with the next two questions.

10. 10 170(10 points) Complete the set method in the CS232LinkedBinarySearchTree class in your hw05 package. The No10Tests class contains tests that you can use to check your implementation of this method.

11. 40 210(40 points) Complete the remove method in the CS232LinkedBinarySearchTree class in your hw05 package. The No11Tests class contains tests that you can use to check your implementation of this method. Suggestion: Handle each of the 3 cases for remove in turn. Implement one, when that passes the appropriate tests; add the next case and so on. Suggestion 2: When you write the 3rd case, add a helper method to find the node with the smallest key in a subtree. Hint: The smallest key will always be in the leftmost node in the subtree.