COMP232 – Data Structures & Problem Solving
Fall 2020

Homework #5
Binary Tree Applications

1. 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.

Insert each number from the input list into the priority queue to make it from smallest to greatest. Then repeatedly move the smallest number to the output and continue until the priority queue is empty.

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

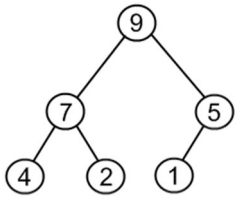
a. an unsorted array based list.

The loop loops through the list to find the minimum element, which takes o(n) then we have to remove the element and fill the gap left by the removed element, which takes o(n) because it also requires scanning the list
So we end up at O(n^2)

b. a heap.

To insert into the heap, for each element we must search at least half of the heap. So that is n times lgn so the worst case is O(nlgn).

3. 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.

4. 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 (i.e. of type `Integer`, `Float`, `Short`, `Byte`, `Double`, `Float`, 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.

Public class GenericBox<Number>

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. Run the `main` method in the `StringMinHeapKey` class in the `hw05` package. What happens? Why? Hint: Look at the constructor being used, the keys being passed in and the `compareTo` method implementation.

The keys are being compared in reverse order, which makes it an invalid heap since B has higher priority than A

b. 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. Just give the body of your `compareTo` method as your solution for this exercise.

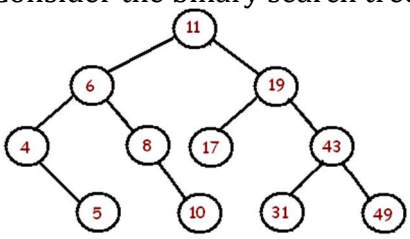
```
public int compareTo(StringMinHeapKey o) {
   /*
    * The line below uses the compareTo method in String to compare the
    * keys. This means "B" has a higher priority than "A".
    *
    * Fix this so that the keys as specified below form a valid heap in
    * the given order. I.e. "A" has a higher priority than  "B", etc.
    */
   return -key.compareTo(o.key);
}
```

c. Explain why your solution in part b works?
The compareTo returns a negative integer, 0 or a positive integer, so effectively by negating the compareTo, it returns the opposite number. The way the compareTo method works is by returning a negative number if the number is lexicographically less than the one being compared to, so negating this value will return a positive value.

6. 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. 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. 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.

9. 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. 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. 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.