Anastasiia Halchenko

COMP232 – Data Structures & Problem Solving
Fall 2024

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.

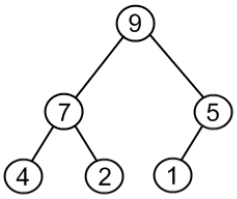
Create a new empty Priority Queue (Max or Min Heap). Add all the integers into the priority queue. Based on the one we choose (Max or Min), the integers will be stored in descending (the biggest at the top) or increasing order (the smallest at the top). Then remove the rightmost child (Max heap) or the root (Min heap) until all the elements are extracted and add them to a new sorted list.

2. Give and briefly justify an asymptotic bound 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.
We need to add all the elements O(n) *n so O(n^2) and remove all the elements in order O(1) but we need to remove all n elements so O(n).
The total running time upper bound O(n^2).

b. a heap.
We need to add all the elements O(lgn) and remove all the elements in order O(logn) but we need to remove all n elements so O(nlogn).
The total running time upper bound O(nlogn).

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<T extends 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.

Error: Heap is not valid. The tree is invalid because the compareTo() method puts the keys in a reverse alphabetical order and when we try to create the tree the min-heap property is not maintained and we have the error of an invalid tree.

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.

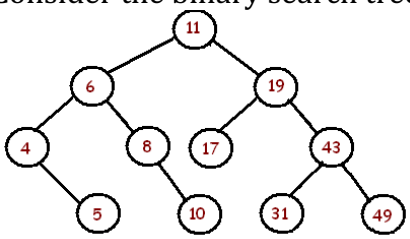
```
public int compareTo(StringMinHeapKey o) {

    return -1*key.compareTo(o.key);
   }
```

c. Explain why your solution in part b works?
Now it is working because when the String is alphabetically higher compared to another String, it will return -1, meaning "A" has a higher priority than "B" and the values are in correct order.

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.