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.

I would use a min-heap priority queue and iterate through the list of integers to add each one. Then while the priority queue is not empty, I will remove the minimum element and add it to a new sorted list in which they will be sorted in ascending order.

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:

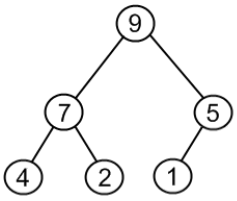
a. Unsorted array based

   $O(n^2)$ – Inserting an element to an array is constant because you will simply append it to the end. But to find the minimum element you would need to do a linear search through the whole array which has a Big O of n. This is conducted n times for each time we remove the minimum element until we don't have anymore.

b. Heap

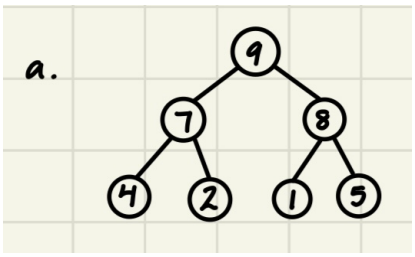
   O(n log n) – Each insertion and extraction has O(log n) because we may need to trickle down/percolate up to maintain our structure. We do this n times for each integer in the list.

3. Consider the heap shown below:

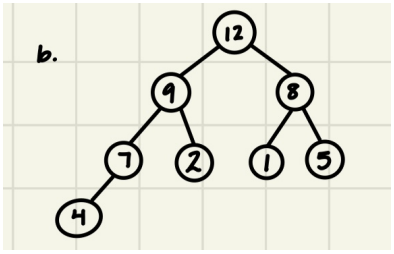


Apply each of the following operations, one after another, to the above max heap. Redraw the heap after each operation has been performed.

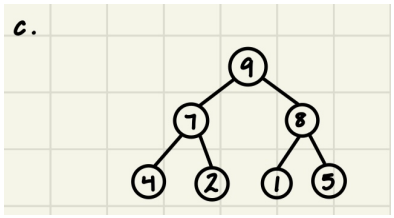
a. `add(8)`



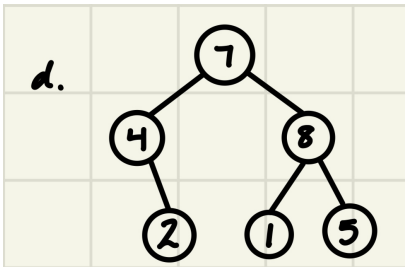
b. `add(12)`



c. `remove()`



d. `remove()`



4. 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<N> {

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.

After running the main method in the StringMinHeapKey class we get an Illegal Argument Exception stating our heap is not valid. When we try to create an instance of the COMP232ArrayHeap class, we call upon the checkHeapProperty method to ensures that every parent's key to greater than its children's keys. This comparison is made by the compareTo method that assigns letters later in the alphabet a higher priority than those earlier on which is producing this error.

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.

```
return o.key.compareTo(key);
```

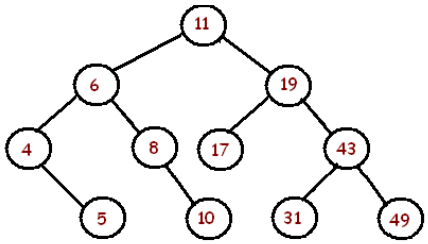
c. Explain why your solution in part b works?

This line will still return 0 if the keys are equivalent but it will return the inverse for the greater than and less than signals.

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.

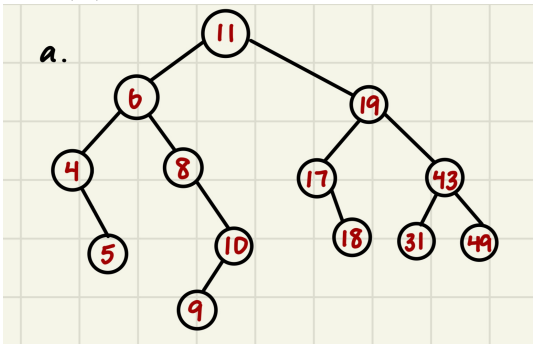
7. Complete the `adjustPriority` method in the `CS232ArrayHeap` class in your `hw05` package. The `No7Tests` class contains tests that you can use to check your implementation of this method.

8. Consider the binary search tree shown below:

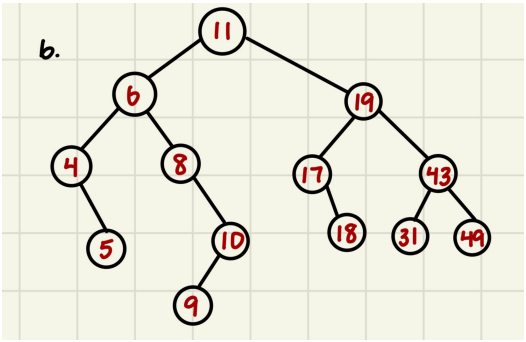


Apply each of the following operations, one after another, to the above binary search tree. Redraw the binary search tree after each operation has been performed.

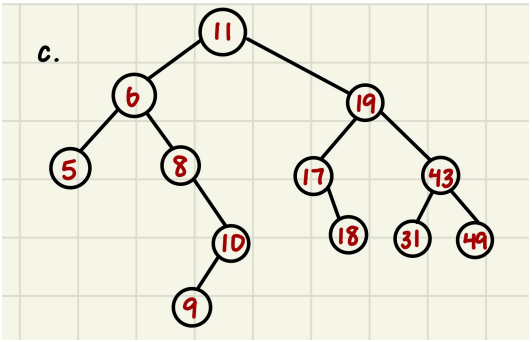
a. `add(9)`



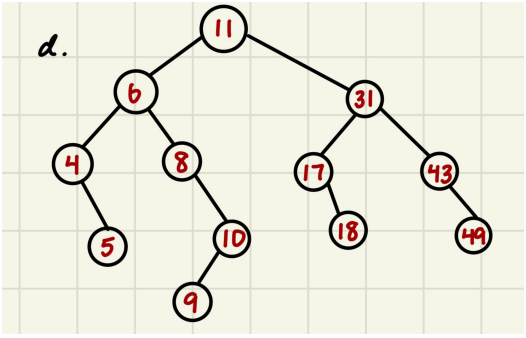
b. `add(18)`



c. `remove(4)`



d. `remove(19)`



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.