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.

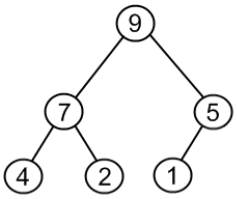
- First, insert each integer from the list into the Priority Queue. In this situation, insert the integers into a Min Heap. By inserting the integers in a Min Heap, the integer with the smallest value will have the most priority, so it will be placed on the top of the queue.
- Secondly, extract each minimum value repeatedly from the Priority Queue and append it to a new list. Hence, each extraction removes the next smallest element in the Priority Queue and then adds to the new list. This makes the list sorted in ascending order. This process can be done by the method poll() to extract the elements from the Priority Queue.
- After all elements are extracted from the Priority Queue, the list is formed by integers and 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. an unsorted array based list.
- To add n elements to an unsorted array, it takes O(n) insertion time.
- To extract the minimum element in the Priority Queue, we need to scan through the whole array to find the smallest element. This would take O(n) time. As we have to extract n elements, so the total time is O(n^2).
  ➔ Total execution time: O(n) + O(n^2) = O(n^2)

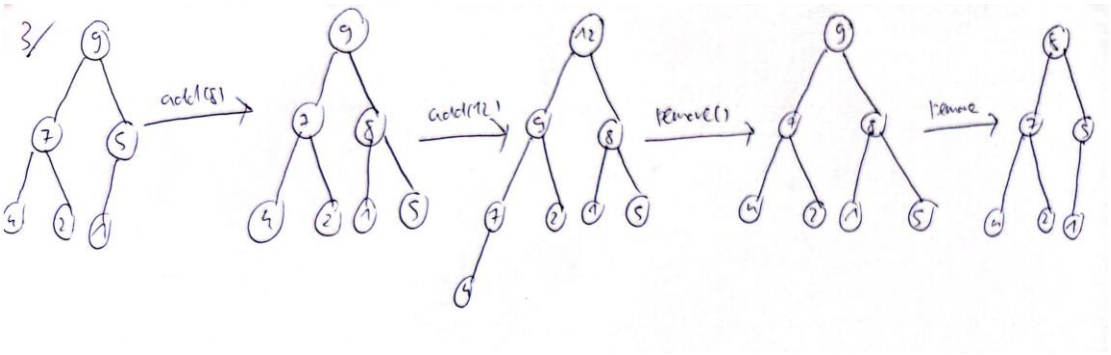
b. a heap.
- To add a element into a heap, the heap needs to percolate up to maintain the heap property, so it takes O(logn) time. So, adding n elements would take O(nlogn) time.
- To extract each element in the heap, the heap needs to trickle down to maintain the heap property when removing elements from the heap, so it takes O(logn) time. So, extracting n elements from the heap would take O(nlogn) time.
  ➔ Total execution time: O(nlogn) + O(nlogn) = O(nlogn)

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<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.

- An exception is thrown with the statement: “Heap is not valid”. Because the CS232ArrayHeap construcrot checks if the elements in the initial array satisfies the property of a Max Heap or not.
- In the StringMinHeapKey class, the objects are constructed in the alphabetical order (“A”, “B”, “C”….). And the method compareTo() used in this class currently uses the String.compareTo, which makes “B” larger than “A”.
- The COMP232ArrayHeap class expects elements to satisfy the max-heap property according to the compareTo method in StringMinHeapKey. By default, "A" is considered less than "B", so "A" would be placed lower in the heap. Because the array doesn’t satisfy this max-heap condition, the constructor throws an IllegalArgumentException to indicate an invalid heap.

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(this.key);
```

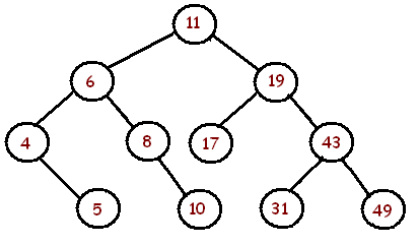
c. Explain why your solution in part b works?

- By changing the order of o.key and this.key for each other “A” is now considered greater than “B”. So the class COMP232ArrayHeap will understand that the StringMeanHeapKey follows the property of the Max Heap.
- With the modified compareTo() method, the keys[] will be organized into a valid max heap with “A” has the highest priority.

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.

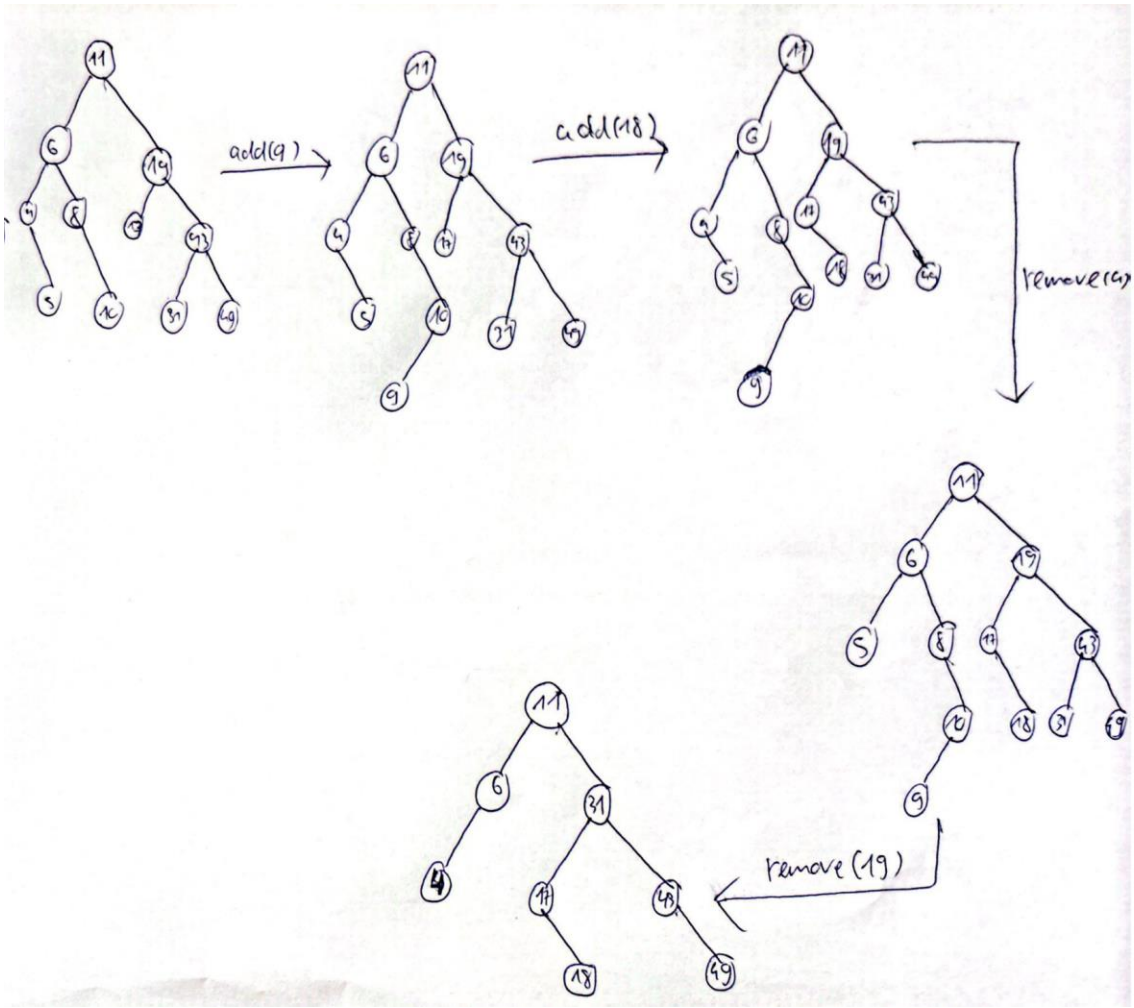
7. Complete the `adjustPriority` method in the `CS232ArrayHeap` class in your `hw05` package. The `No7Tests` class contains tests that you can use to check your implementation of this method.

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.