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.

Firstly, create an empty priority queue and then insert each integer from the list into the priority queue. It follows that after ordering all the elements, start removing each element one by one from the priority queue. Finally, store each removed element in a new list and continue to remove 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:

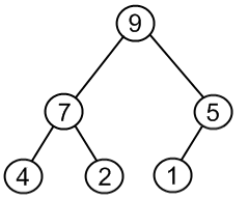
a. an unsorted array based list:

+ The total asymptotic bounds for the running time of the algorithm is O(n^2) since the Insert method for n elements in take O(n) times while with n removals, the Remove method leads to O(n^2) because the array is unsorted, then we need to iterate the array each time to remove the smallest (the biggest) element.

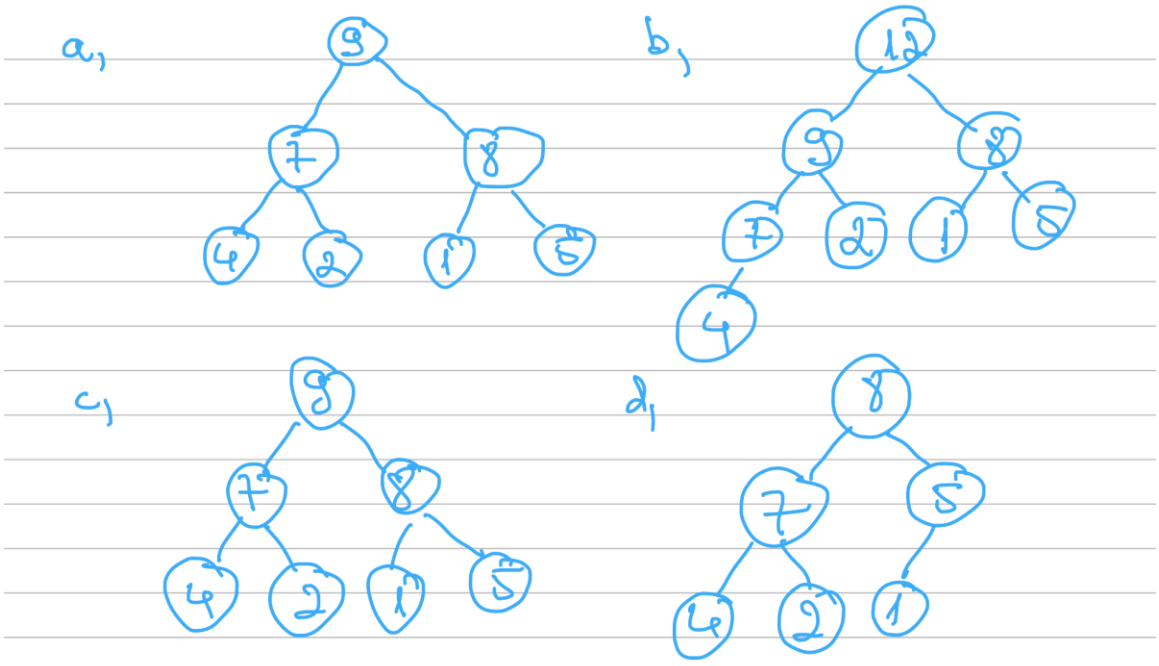
b. a heap.

+ The total asymptotic bounds for the running time of the algorithm is O(n logn) since the Insert method takes O(log n) times but when inserting n elements, we need to trickle up, resulting in O(n logn), and for the Remove method takes O(logn) but the for n elements, we need to trickle down, leading to O(n logn)

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.

The IllegalArgumentException is thrown: Heap is not valid because the compareTo method does not show the desired orderings with “A” as the highest priority, resulting in the fact that the heap validation fails.

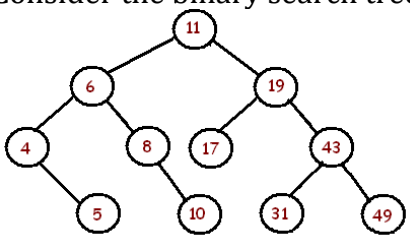
b. Modify the `compareTo` method so that the given keys form a valid heap with "A" having higher priority than "B", and "B" higher priority than "C", etc. Just give the body of your `compareTo` method as your solution for this exercise.

```
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 o.key.compareTo(this.key);
}
```

c. Explain why your solution in part b works?

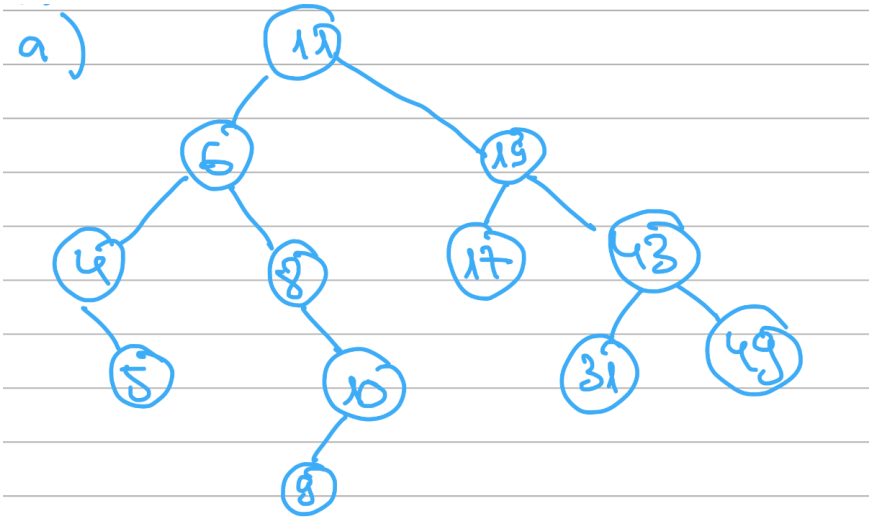
The solution part(b) works because by changing `key.compareTo(o.key)` to `o.key.compareTo(this.key)`, making the order reverse, then the string with lower values will come first in the order and "A" will serve as the root of the MinHeap.

8. Consider the binary search tree shown below:

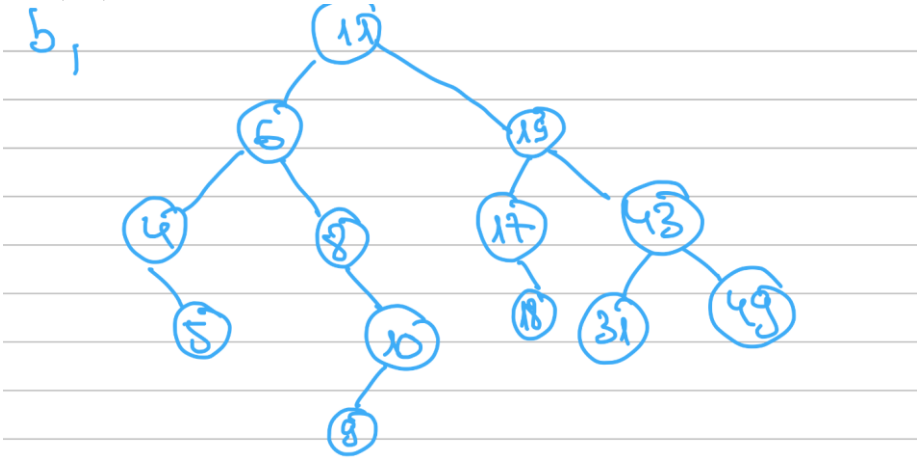


Apply each of the following operations, one after another, to the above binary search tree. Redraw the binary search tree after each operation has been performed.

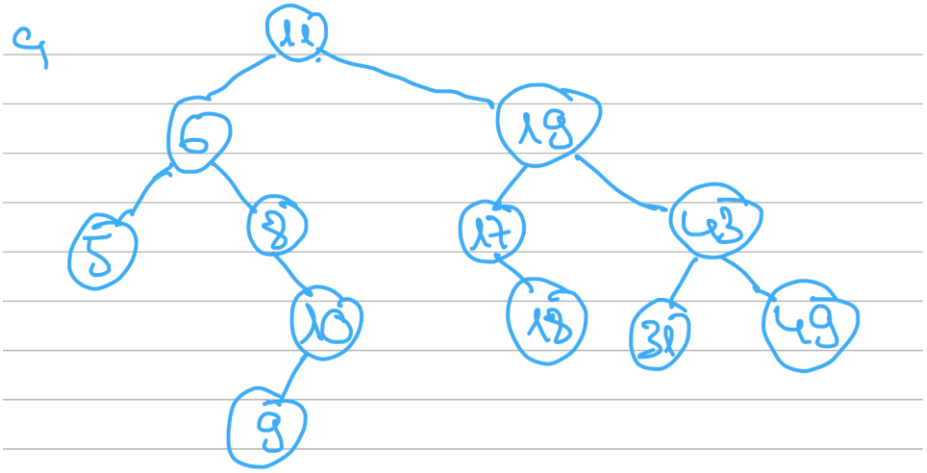
a. `add(9)`



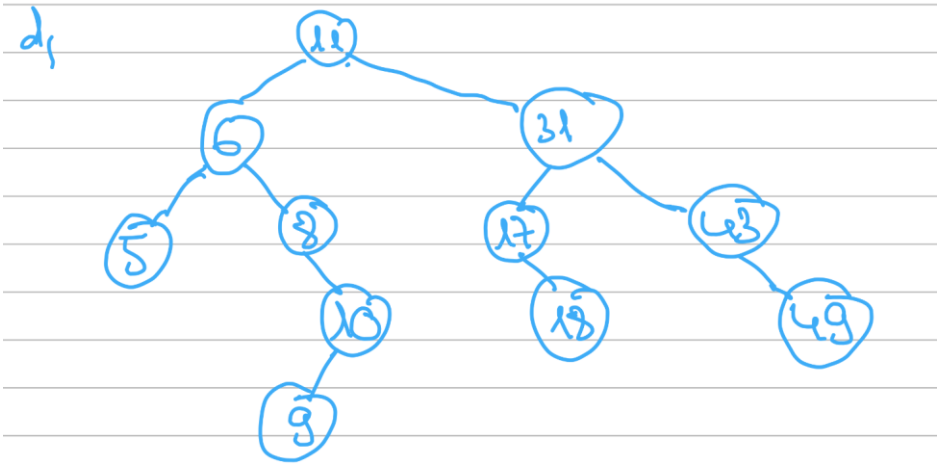
b. `add(18)`



c. `remove(4)`



d. `remove(19)`


d1
11
6
31
5
8
17
43
10
18
49
9