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.

(Assume that we want the list sorted in the same order that the priority queue would prioritize elements.)

For each element in that list of integers, add their value to the priority queue.
Then for each element in the original list, set its value to be the one removed from the queue.

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.

$O(n^2)$

`add()` is O(n) since we don't need to re-sort the array, but in the worst case when the array is out of space, we need to create a new one and copy all the element from the old array to the new array.
`remove()` is O(n) since the array is unsorted and we need to search for the highest priority element to remove.

`set()` is O(1).

Hence the upper bound is $n*O(n) + n*O(n) + n*O(1) = O(n^2)$.

b. a heap.

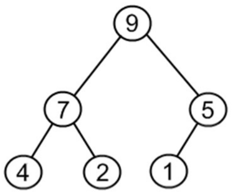
O(nlgn)

`add()` is O(lgn) since we may have to percolate up, `remove()` is O(lgn) since we may have to trickle down.

`set()` is O(1).

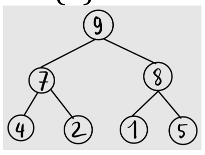
Hence the upper bound is n*O(lgn)+ n*O(lgn)+n*O(1)= O(nlgn).

3. Consider the heap shown below:

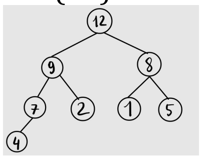


Apply each of the following operations, one after the other, to the above max heap. Redraw the heap after each operation has been performed.

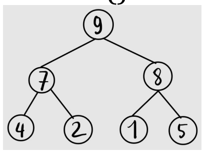
a. add(8)



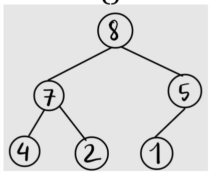
b. add(12)



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.

We have an unhandled `IllegalArgumentException` in main. The constructor created a heap in which a node of key “B” is child of a node of key “A”. “A” should be “greater than” “B” since the root has higher priority than its children. However, the `compareTo()` method in the `StringMinHeapKey` class compares alphabetical order. Hence “A” < “B”, making the constructor throw the `IllegalArgumentException`.

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

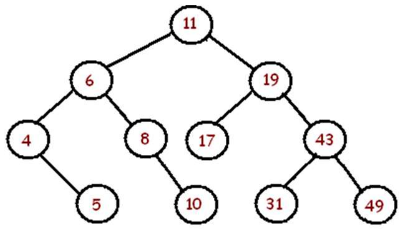
c. Explain why your solution in part b works?

Still using default `compareTo()` of `String` class to compare the values, but now the return value will be reversed since `key` and `o.key` are swapped. Another way to do this is to multiply the whole expression with -1, which is `(-1)*(key.compareTo(o.key))`. Now the `compareTo()` method of class `StringMinHeapKey` should return proper value, hence the constructor for the max heap won’t throw the exception.

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.

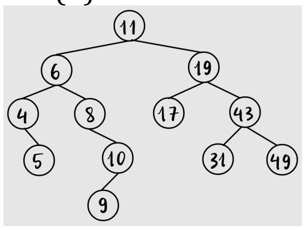
7. Complete the `adjustPriority` method in the `CS232ArrayHeap` class in your `hw05` package. The `No7Tests` class contains tests that you can use to check your implementation of this method.

8. Consider the binary search tree shown below:

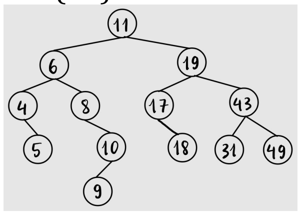


Apply each of the following operations, one after another, to the above binary search tree. Redraw the binary search tree after each operation has been performed.

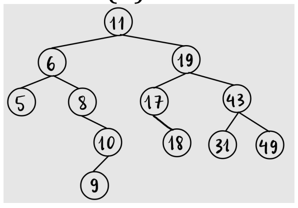
a. add(9)



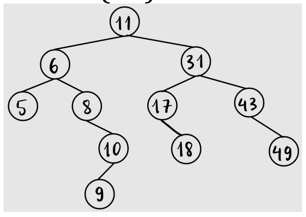
b. add(18)



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.