CIS 212 Assignment 6: 100 points

The goal of this assignment is to provide exposure to building a new generic data structure. Specifically, this project will involve building a new Set structure that maintains the "add" counts of its unique elements, so that they can be iterated in sorted order with respect to how many times they've been added to the set. The Set implementation will be optimized for efficient add/remove/contains over efficient iteration.

1. [70] Create a new class OccurrenceSet<T> that implements Set<T>. Your OccurrenceSet implementation should create and maintain a HashMap<T, Integer>; doing so will allow you to easily track the integer "add" count for each element in the set. All methods should function as specified in the Set documentation. Additionally:

- (10) The *add* and *addAll* methods will need to keep track of how many times an element has been added to the set. We are optimizing for efficient add, so *add* should be *O(1)* and *addAll* should be *O(n)* for *n* added elements.
- (10) The *remove*, *removeAll*, and *retainAll* methods should remove the necessary elements from the set completely (i.e., not just decrement their counts). We are optimizing for efficient remove, so *remove* should be *O(1)* and *removeAll* should be *O(n)* for *n* removed elements.
- (10) The *contains* and *containsAll* methods should behave as documented and operate in *O(1)* and *O(n)* time (i.e., for *n* query elements), respectively.
- (10) The s*ize* method should return the number of unique elements currently in the set (i.e., not considering "add" counts). This method should be *O(1)*. The *clear* and *isEmpty* methods should behave as documented.
- (20) The *iterator* and *toArray* methods should return an Iterator or array, respectively, with elements sorted by their "add" counts in ascending order. We are optimizing for efficient add/contains over iteration, but these methods should still be *O(n lg n)*. The Iterator *remove* method can be left blank for this assignment.
  Hint: Creating a new List and using the Collections *sort* method is reasonable here, though note that you'll need to implement a Comparator object so that the elements can be sorted by their associated counts.
- (10) Add a *toString* method that returns a string representation of the elements in the list in sorted order (i.e., ascending with respect to their "add" counts). This method should be *O(n lg n)*.

2. [20] Create a Main class that creates a few OccurrenceSets of various types to test the functionality of your new data structure. For example, a test like:

```
OccurrenceSet<Integer> intSet = new OccurrenceSet<Integer>();
```

```
intSet.add(1);

intSet.add(3);

intSet.add(5);

intSet.add(5);

intSet.add(3);

intSet.add(3);

intSet.add(3);

System.out.println(intSet);

OccurrenceSet<String> stringSet = new OccurrenceSet<String>();

stringSet.add("hello");

stringSet.add("hello");

stringSet.add("world");

stringSet.add("world");

stringSet.add("world");

stringSet.add("here");

System.out.println(stringSet);
```

Should have the output:

```
[1, 5, 3]

[here, hello, world]
```

3. [10] Write code that is clear and efficient. Specifically, your code should be indented with respect to code blocks, avoid unnecessarily repetitive code, avoid code that is commented out or otherwise unused, use descriptive and consistent class/method/variable names, etc.

Please zip only your Java source file(s), i.e., .java file(s), into a zipped file, rename that file as Assignment6.zip, e.g., BillGatesAssignment6.zip, and then upload that file to Canvas. Do not put the Java source file(s) in a folder and zip that folder; instead, please directly zip all the Java source files into a single zipped file. You are free to use whatever tools/IDEs you prefer to complete your assignment.