Indiana University
CSCI-C212: Introduction to Software Systems (S25) Lab 06
Joshua Crotts, Muazzam Siddiqui

Java Collections

What To Do:

Follow each step carefully. As you complete the lab, submit the source files (`.java`) problems to the autograder (link is in the Canvas portal). After finishing, please submit your work to the autograder and let one of the TAs know.

For this lab, please place each method inside its own class file labeled as `ProblemX`, where `X` is the problem number. The accompanying test files should be named `ProblemXTest`.

Problem 1

This exercise is based around Exercise 3.61 in *Learning Java - 1st Edition*.

Anagrams are strings that are formed by rearranging the letters of another string. For example, "plea" is an anagram for "leap", but we consider an alphabetized anagram to be the alphabetized arrangement of letters for an anagram. As an example, "aelrst" is the alphabetized anagram for "alerts", "alters", "slater", and "staler". Design the `static Map<String, Set<String>> alphaAnagramGroups(List<String> los)` method, which maps all alphabetized anagrams to the strings *ls* using the above criteria.

The output order of the map is significant: it should be alphabetized by the ordering of the strings. (Which `Map` type should you use?) Additionally, the fact that we map alphabetized anagrams to strings to sets indicates that duplicates do not matter, and that is true. The ordering of the sets is significant: it should be the insertion order of the strings. (Which `Set` type should you use?)

Use the following test case as an example, but you need to write **more** tests! If you use only this test, then your JUnit test score will be very low.

```
los = ["presorting", "plea", "introduces", "anger", "leap", "petals",
       "donate", "plates", "range", "reductions", "rediscount", "anger"
       "tapers", "pale", "atoned", "staple", "repast", "reportings"]
alphaAnagramGroups(los)
  => {{"adenot", ["donate", "atoned"]},
      {"aegnr", ["anger", "range"]},
      {"aelp", ["plea", "leap", "pale"]},
      {"aelpst", ["petals", "plates", "staple"]},
      {"aeprst", ["tapers", "repast"]}
      {"cdeinorstu", ["introduces", "reductions", "rediscount"]},
      {"eginoprrst", ["presorting", "reportings"]}}
```

You will need to design a method that creates an "ordered version" of a given string. You **cannot** use any Java sorting methods to do this, e.g., `Arrays.sort`, `Collections.sort`, and so forth. Think about how you can leverage a `PriorityQueue`! Hint: watch my videos if you haven't.

Problem 2

Design the `Set<Integer> moreThanThree(int[] A)` method, which receives an `int[]` A and returns a new `Set<Integer>` of values containing those values from A that occur strictly more than three times.