Computing and Algorithms III: Program 3

CS-203 **Summer 2012**

Assignment Weight: 1.0

Background: Anagrams

An *anagram* is a word or phrase that is created by rearranging the letters of another word or phrase. For example, the words "*eat*", "*ate*", and "*tea*" are anagrams of one another.

The Assignment

1. Write a Java program which accepts the name of a file on the command line. The file will contain an unknown number of words, with a single word on each line. Your program will read the file and store the words internally in an appropriate format.

   Your program will then find all sets of anagrams of words in the file *in an efficient manner*. Your anagram-detection algorithms should ignore distinctions between upper-case and lower-case letters. For example, "Elvis" and "lives" should be identified as anagrams of one another.

   Upon completion, the program will print the detected sets of anagrams to standard output. The manner in which you print those sets is up to you; as long as the results are neat and readable, you are free to use any format you like.

2. Create an ordinary text file called README with instructions on how to compile and run your programs.

3. Create an ordinary text file called WRITEUP with answers to the following question:

   (a) What is the theoretical worst-case running time of the algorithm you implemented (*i.e.* in Θ-notation), expressed in terms of the number of words n in the input file? Justify your answer.

Your program should use "good style". See the separate handout on style requirements for CS-203 programs. A portion of the grade on this program will depend upon your style.

Also, note that for this program, a portion of the grade on this program will depend on the efficiency of your algorithm.

Submitting Your Program

Before 11:59:59 p.m., Thursday, 30 August 2012 (8th Thursday), you must upload a zip archive to the course Blackboard assignment for Programming Assignment 3. This zip archive must contain all source code files for your program, along with the two text files named above.

Notes

1. For this and all successive programs, unless otherwise specified, you may use any classes from the standard Java library that are useful to you. (It'd be cheating to allow you to use the built-in balanced tree classes in the assignment where you have to write a balanced tree . . . duh . . .)

2. Your program should work on "large" data files. For example: most Linux systems use the file "`/usr/dict/words`" as a dictionary for spell-checking programs; such files routinely contain thousands of words. You should test your program against such a file.

3. The `String` class in Java contains methods `toLowerCase()` and `toUpperCase()`.

4. This program is based, in part, on exercise 6.1.11 from Levitin.

5. **HINT:** The section of the textbook from which this exercise was developed deals with the algorthmic technique known as *presorting*. How might presorting help with this problem?