The purpose of this assignment is to practice your knowledge of sorting, testing, and algorithm analysis. The assignment must be completed in pairs.

1. The Problem
   As part of a larger word puzzle game, we have been asked to construct a program that determines if two words are anagrams and finds the largest group of anagrams in a list of words. Two words are anagrams if they contain the same letters in the same frequency. For example, alert and later are anagrams.

   Now that we know how to implement an insertion sort and Shellsort, we are eager to use sorting to solve these two problems.

   - To check if two words are anagrams, simply sort the characters in each word. If the sorted versions are the same, the words are anagrams of each other.
   - To find the largest group of anagrams in an array of words, sort the array with a `Comparator` that compares the sorted character representations of the words. After the sort, any group of words that are anagrams of each other will be adjacent in the array.

2. Requirements

   - Create a class `AnagramUtil` in package `assignment4`. Add the following five methods. (You may also add private helper methods, as needed.) You may also create extra helper classes in separate java files if necessary. Make sure you hand in all java files necessary for your program to work correctly.

     - `public static String sort(String)`
       This method returns the sorted version of the input string. The sorting must be accomplished using an insertion sort.

     - `public static <T> void insertionSort(T[], Comparator<? super T>)`
       This generic method sorts the input array using an insertion sort and the input `Comparator` object.

- `public static <T> void shellSort(T[], Comparator<? super T>)`

  This generic method sorts the input array using a Shell sort and the input `Comparator` object.

- `public static boolean areAnagrams(String, String)`
  This method returns true if the two input strings are anagrams of each other, otherwise returns false.

- `public static String[] getLargestAnagramGroup(String[])`
  This method returns the largest group of anagrams in the input array of words, in no particular order. It returns an empty array if there are no anagrams in the input array. Use either an insertion sort or a Shell sort to achieve this. The reason for having both sorting algorithms is for the analysis portion of this assignment. In order to properly complete the analysis portion of this assignment, this method must use the algorithm described in section 1 of this assignment.

- Create and submit your test class(es). Using JUnit for some or all tests is encouraged, but not required. Ensure that your tests are organized and cover a broad range of possible input.
- The following are some notes to help guide your implementation.
  - We will concern ourselves only with word anagrams and not anagrams that are phrases, which may contain whitespace and punctuation.
  - You may assume that the word list given as input to both `getLargestAnagramGroup` methods does not contain duplicates.
  - There are many websites devoted to listing anagrams, such as this one.
  - As an example, consider sample_word_list.txt. Invoking `getLargestAnagramGroup` on an array of the words contained in sample_word_list.txt should return an array containing these strings, in no particular order: carets, Caters, caster, crates, Reacts, recast, traces. moderate_word_list.txt is another slightly larger test file. And here is a web site with a very large list of English words.
  - Download this file, which contains a static method for reading strings from a text file in to a String array, and a method for generating a random String. Download sample_word_list.txt in to your project directory for this example to work. Use this method to read in

sample_word_list.txt in to an array of Strings to pass to your getLargestAnagramGroup method, or use it to generate a large list of random words. You do not need to use this class as a starting point for your test class, but you may if you wish.
    - As demonstrated by the example above, words with the same letters in the same frequency, but different cases are still anagrams. E.g., Begin and being are anagrams. Hint: Java's built-in String method `toLowerCase` will come in handy when sorting and determining if words are anagrams, but you must still output them in their original case.

3. When preliminary coding is complete and your program compiles without error or warning, test the program thoroughly and systematically.
   Your code should be well-commented (Javadoc comments are recommended) and formatted such that it is clear and easy to read. Be sure to put the names of both programming partners in the header comment of each file.

   Zip your source code files (no .class files, source code only) and **upload the zip file here by 5pm on February 7**. Be sure to include all files necessary for running your code. If your code uses a separate comparator class in its own file, for example, we need it. Please submit just one solution per pair (i.e., one partner should upload the zip file and an analysis doc, the other should only upload an analysis doc).

4. Analysis document (must be written and submitted by each programming partner) due **February 7 at 5pm**.

Due date:	Thursday, 7 February 2013, 5:10 PM

Submission feedback

Daniel Kopta
Thursday, 7 March 2013, 6:44 PM

Grade: 76.00 / 100.00

Testing AnagramUtil.sort(String s)

Points: 12 / 12

Testing AnagramUtil.insertionSort(T[] data, Comparator<? super T> comp)
Using a Comparator<Integer> which sorts in descending absolute value order.

Points: 10 / 10

Testing AnagramUtil.shellSort(T[] data, Comparator<? super T> comp)
Using a Comparator<Integer> which sorts in descending absolute value order.

Points: 10 / 10

Testing AnagramUtil.areAnagrams(String a, String b)

Points: 12 / 12

Testing AnagramUtil.getLargestAnagramGroup(String [] s)

Failed to report correct anagram list on array with two large groups. -4 points.
Failed to report correct anagram list when two equal size groups exist. -4 points.
Failed on list without any anagrams. -4 points.
Failed on list with only one word. -4 points.
Failed for test file given. -6 points.

Points: 4 / 26

Style & Testing

+5 style

+1 testing
Points: 6 / 10

Summary

Total: 54 / 80

final 66/100

Comments:

-the autograder failed because the comparators you create are referencing an external class. It is better to use a nested class for the comparator for an assignment submission so that autograder does not fail. Note, no points were deducted since you included the comparator file in your submission.

-4 (testing) please note that quality and variety of tests will win over quantity. A lot of the test cases you included were redundant. When creating tests, think of the different types you might encounter such as when an anagram exists, when one doesn't exist, on a large set of inputs, etc.

analysis 12/20

justinbarsketis.pdf

Submission

Analysis4.pdf

Assignment4Final.zip

Edit these files