Description:

My algorithm sorts letters of a word alphabetically (sortedWord) for all the words in the dictionary. WordAnagram is the class that stores the original word and the sortedWord. WordAnagram object is created for every single word in the dictionary and the objects are all sorted in an array. That array gets sorted by the alphabetical order of the sortedWord of WordAnagram objects. Original word of the first WordAnagram in the array is written in the file by FileWriter. After that, the array gets iterated through again, starting from index 1. It compares the sortedWord of that WordAnagram with the sortedWord of the WordAnagram previous to that. When those sortedWords are different, the FileWriter writes "\n" to continue on to the next line, and when they're the same, the FileWriter writes " " to put the word on the same line as the previous word. After putting either "\n" or " ", the original word of the WordAnagram gets written by the FileWriter. When the array is iterated through all the way, the FileWriter gets closed and the file is returned.

Correctness:

I use mergesort of char array to sort the letters and make sortedWord out of the original word. Anagrams have same sortedWords. For example, "spots" and "posts" have the same sortedWord of "opsst". Since all anagrams should have same sortedWord, using mergesort to sort the array by alphabetical order of sortedWord should group anagrams next to each other. So comparing sortedWord of the previous WordAnagram object in the array and the current sortedWord will check if the previous word was an anagram of the current word.

RunTime:

n words in the dictionary

k letters in a word (in average)

Counting how many words are in the whole dictionary:

- Iterating through the whole file = $\theta(n)$

Sorting letters of word alphabetically:

- One word = $\theta(k\log k)$
- For all the words in dictionary = $\theta((nk)\log k)$

Sorting array in alphabetical order of sortedWords

- Sorting array with size of n = $\theta(n\log n)$

Writing in the file:

- Iterating through and comparing sortedWords = $\theta(n + cn)$

TOTAL TIME = $\theta((nk)\log(nk))$

Run Time tested:

Time was implemented into the java file. For dict1, time was 465 milliseconds. For dict2, time was 23316 milliseconds. When I used the time command for both dict1 and dict2 together, the real time was 0m 23.950s, the user time was 1m49.484s, and the sys time was 0m3.500s.

I used my laptop HP Envy, and ran these time tests in EDLAB.

Outputs:

In dict1, there are 67606 anagram classes, and in dict2, there are 320750 anagram classes.

When printing out anagram classes with more than 5 words for dict1, the output is the following:

```
ardeb barde beard bread debar debra

caret cater crate react recta trace

leapt lepta palet petal plate pleat

least setal slate stale steal stela teals

elva lave leva vale veal vela

aril lair liar lira rail rial

reins resin rinse risen serin siren

luster lustre result rustle sutler ulster
```