```java
/**
 * WordAnagram class is a class for object where there are two Strings,
 * the original word and the word that has letters sorted alphabetically.
 * @author Caroline Kim (ID: 2895696)
 *
 */
public class WordAnagram {
    private String word;
    private String sortedWord;

    /**
     * Constructor
     * @param w: original word
     * @param s: sorted word
     */
    public WordAnagram(String w, String s){
        word=w;
        sortedWord=s;
    }

    /**
     * get word
     * @return
     */
    public String getWord(){
        return word;
    }

    /**
     *   get sorted words
     * @return
     */
    public String getSorted(){
        return sortedWord;
    }
}
```

```java
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileWriter;
import java.io.IOException;
import java.util.Scanner;
/**
 * Anagram class takes in a file(dictionary) and sorts each word's letters into alphabetical order.
 * Then the bigger array that contains all the words that their letters are sorted into alphabetical order gets sorted.
 * Iterating through that bigger array, anagrams are found by looking at words that have the same alphabetical order of letters.
 * ex:(spot -> opst, tops -> opst)
 *
 * @author Caroline Kim (ID: 29855696)
 *
 */
public class Anagram {

    private File dictionary; //dictionary file
    private WordAnagram[] alphaArr; //data structure (array) where the word and the word sorted in alphabetical order of letters will be
stored
    private int wordCount=0; // number of words in the dictionary

    /**
     * Constructor
     * - Takes in a file, counts how many words are in the file, and creates WordAnagram array
     * @param f: file that is assumed to be the dictionary
     */
    public Anagram(File f){
        dictionary = f;

        try {
            Scanner count = new Scanner(dictionary);//incrementing wordCount by iterating through the dictionary
            while(count.hasNextLine()){
                wordCount++;
                count.nextLine();
            }
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        }

        alphaArr = new WordAnagram[wordCount]; //creating array

    }

    /**
     * merge method for the char array
     * @param left: left division of the char array
     * @param right: right division of the char array
     * @param arr: whole char array
     */
    private void merge(char[] left, char[] right, char[] arr){
        int i = 0; //pointer for left array
        int j = 0; //pointer for right array
        int k = 0;//pointer for the whole array

        while ((i < left.length) && (j < right.length)) {//compare i-th element in left array and j-th element in right array
            if (left[i] <= right[j]) {
```

```
            arr[k] = left[i];
            i++;
            k++;
        } else {
            arr[k] = right[j];
            k++;
            j++;
        }
    }
    while (i < left.length) {//if right array was emptied first, insert rest of left array elements
        arr[k] = left[i];
        k++;
        i++;
    }
    while (j <right.length) {//if left array was emptied first, insert rest of right array elements
        arr[k] = right[j];
        k++;
        j++;
    }

}

/**
 * mergesort for char array (recursive)
 * @param arr: array of chars that is to be sorted in alphabetical order
 */
private void mergesort(char[] arr){
    if(arr.length>=2){//base case
        int mid = arr.length/2;//divide arr into two halves
        char[]left= new char[mid];
        char[]right= new char[arr.length-mid];

        for(int i=0;i<mid;i++){//copy left array
            left[i]=arr[i];
        }
        for(int i=mid;i<arr.length;i++){//copy right array
            right[i-mid]=arr[i];
        }
        mergesort(left);//conquer left

        mergesort(right);//conquer right

        merge(left, right, arr);//merge left and right arrays
    }
    else
        return ;
}

/**
 * method to fill alphaArr array
 * For each words in the dictionary, it creates a new WordAnagram object with the original word
 * and the word that is sorted into alphabetical order of the letters, and inserts it into the array
 * @throws FileNotFoundException
 */
public void fillArr() throws FileNotFoundException{
    Scanner scan = new Scanner(dictionary);//scanner to iterate through the dictionary
```

```
        for(int i=0; i<alphaArr.length; i++){
            String curr = scan.nextLine();//original word in dictionary that is currently being iterated through
            char[] temp = curr.toCharArray();//creates char array and inserts each letter of the word into this array
            mergesort(temp);//sorts the letters in the word alphabetically and stores it in temp
            alphaArr[i]=new WordAnagram(curr, String.valueOf(temp));//inserts WordAnagram object with the original word and sorted word into alphaArr
        }
    }

    /**
     * merge method for WordAnagram array by comparing sortedWords
     * @param left: left division of WordAnagram array
     * @param right: right division of WordAnagram array
     * @param arr: whole WordAnagram array
     */
    private void merge2(WordAnagram[] left, WordAnagram[] right, WordAnagram[] arr){
        int i = 0; //pointer for left array
        int j = 0; //pointer for right array
        int k = 0;//pointer for the whole array
        while (i < left.length && j < right.length) {
            if (left[i].getSorted().compareTo(right[j].getSorted())<=0) {//compare i-th element's sortedWord in left array and j-th element's sortedWord in right array
                arr[k] = left[i];
                i++;
                k++;
            } else {
                arr[k] = right[j];
                k++;
                j++;
            }
        }
        while (i < left.length) {//if right array was emptied first, insert rest of left array elements
            arr[k] = left[i];
            k++;
            i++;
        }
        while (j < right.length) {//if left array was emptied first, insert rest of right array elements
            arr[k] = right[j];
            k++;
            j++;
        }

    }

    /**
     * mergesort for WordAnagram array (recursive)
     * @param arr: array of WordAnagrams that is to be sorted in alphabetical order of sortedWords
     */
    private void mergesort2(WordAnagram[] arr){
        if(arr.length>=2){//base case
            int mid = arr.length/2;//divide array into two halves
            WordAnagram[]left= new WordAnagram[mid];
            WordAnagram[]right= new WordAnagram[arr.length-mid];

            for(int i=0;i<mid;i++){//copy left array
                left[i]=arr[i];
```

```
            }
            for(int i=mid;i<arr.length;i++){//copy right array
                right[i-mid]=arr[i];
            }
            mergesort2(left);//conquer left array
            mergesort2(right);//conquer right array
            merge2(left, right, arr);//merge left and right arrays
        }
        else
            return ;
    }

    /**
     * Creates a new file by the given fileName
     * This file is a list of all the words in the dictionary with anagrams on the same line
     * @param fileName: name of the file that will be created
     */
    public void findAnagram(String fileName){

        mergesort2(alphaArr);//sort alphaArr by the order of sortedWords in each elements(WordAnagram)

        File file = new File(fileName);//create new file

        try {
            FileWriter writer = new FileWriter(file);//writer to write in this file
            if(wordCount <=0){//checks that there is at least one word in the dictionary to avoid NullPointerException
                writer.close();
            }
            writer.write(alphaArr[0].getWord());//write first word of the dictionary
            for(int i=1; i<alphaArr.length; i++){
                if(alphaArr[i-1].getSorted().equals(alphaArr[i].getSorted())){//if sortedWord of previous WordAnagram and current
WordAnagram are the same,
                    writer.write(" "+alphaArr[i].getWord());                    // then these two original words are anagram, so they're
separated by a space
                }else{                                                          // else,
                    writer.write("\n" + alphaArr[i].getWord());                 // then these two original words are not anagrams, therefore
they're separated by a line break
                }
            }
            writer.close();//close the writer


        } catch (IOException e) {
            e.printStackTrace();
        }
        return;
    }



}
```

```java
import java.io.File;
import java.io.FileNotFoundException;

/**
 * Driver class that has main
 * @author Caroline Kim (ID: 29855696)
 *
 */
public class AnagramDriver {

   public static void main(String[] args) throws FileNotFoundException {
      //dict1 testing and measuring running time
      long startTime1 = System.currentTimeMillis();

      Anagram anag1 = new Anagram(new File("/courses/cs300/cs311/kyuminkim/CS311-Anagram/dict1"));
      anag1.fillArr();
      anag1.findAnagram("/courses/cs300/cs311/kyuminkim/CS311-Anagram/anagram1");

      long endTime1 = System.currentTimeMillis();
      System.out.println("total time for dict1: " + (endTime1-startTime1));

      //dict2 testing and measuring running time
      long startTime2 = System.currentTimeMillis();

      Anagram anag2 = new Anagram(new File("/courses/cs300/cs311/kyuminkim/CS311-Anagram/dict2"));
      anag2.fillArr();
      anag2.findAnagram("/courses/cs300/cs311/kyuminkim/CS311-Anagram/anagram2");

      long endTime2 = System.currentTimeMillis();
      System.out.println("total time for dict2: " + (endTime2-startTime2));
   }

}
```