**CSCD 300 Homework 7 and Homework 8 ( Project )**

**Auto Completion and Predictive Input Using Prefix Tree and Hashtable**

**Total 200 points**

**Turn in:** On EWU Canvas, CSCD300🡪 Assignments🡪Hw78🡪Submit.

Please put all your source code (.java) files and other required files and folders together into a zip file. Name the zip file with your last name followed by first initial plus hw78.zip. For example, smithjhw78.zip is for John Smith.

**If you forget to include your source code in the zip file, you get a zero credit for this homework. If your code shows a compile-time error, you get a zero credit. If you turned in a corrupted file, or a file that cannot be opened, you get a zero.**

**What is provided?**

1. You are provided with two dictionaries that are generated from homework 2 and named as **dictSmall.txt** and **dictionary.txt** respectively. English words in both dictionaries are **sorted** in alphabetical order. Format of the dictionary file record is as follows,

a,7300

aa,6

aac,3

aachen,1

…..

………

The numeric value following the comma shows **the frequency** of that word. The higher frequency number a word has, the most frequently used it is. You can debug your program using the dictSmall.txt, because it is a very small dictionary of the same format.

1. Trie2.java includes the code to implement a typical prefix tree (Trie). In this project, we only focus on its **search**(in that class, we call it findWord) and **insertion**( in that class we call it insertString) operations. In the project, you are **free to modify** the provided Trie2.java file in order to fulfill the requirements of the project.
2. AutoCompleteStudent.java implements the GUI. You have to **change very few** places, in order to display the most frequently used words returned from a method in Trie data structure. Please refer to the comments I left in AutoCompleteStudent class, starting with **YOU HAVE TO DO**. **Similarly, you probably have to add a few instance variables and change (or add) new constructor in order to fulfill the requirements.**

**Hw7 Problem Description (130 points)**

Based on the materials we discussed and my demonstration in classroom, you are required to implement a predictive input application (or auto completion). A similar program is the google search box. If we type in **u** in google search box, it automatically pops up four words (or strings), **[usps, ups, us bank, ups tracking].** We observe that all four strings start with **u**. If we continue and type in **n,** now we get **un** in the box**,** it will show a different set of strings **[ united airlines, university of idaho, university of washington, unemployment].** All strings in this set start with **un.**

**Back to** **our GUI** application, each time we press an alphabetical letter on keyboard, we are creating a **prefix** of a word. On each keystroke, we have to send the **prefix** to a Trie structure, in order to retrieve all English words that contain the same prefix. Let us assume this operation returns a set of words **S**. English words in **S** are first sorted, then are displayed in the middle of the input area in the GUI window. Here, we like **S** to be sorted according to the frequency of a word, because we like to put the most frequently used words in the front of the display list.

The GUI class **AutoCompleteStudent.java** will display the set of words **S**, the list of most frequently used words with a specific prefix **P.** So that a user could pick the first word(left most one) in **S** by pressing Space key or Enter key on the keyboard. The chosen word will be inserted on the top of the input area. A user could also press the number key associated with a word in the sorted **S**, for picking a word. After choosing a word from **S**, the user is supposed to start inputting a new prefix.

In this application, we use a Trie data structure to represent all words in a dictionary and all prefixes of each word. Each time we press an alphabetical letter on keyboard, we have to send the current **prefix** to the Trie structure. The Trie retrieves all English words that have that prefix. Also, the list of words that the Trie returns has to be sorted according to the frequency number of the words.

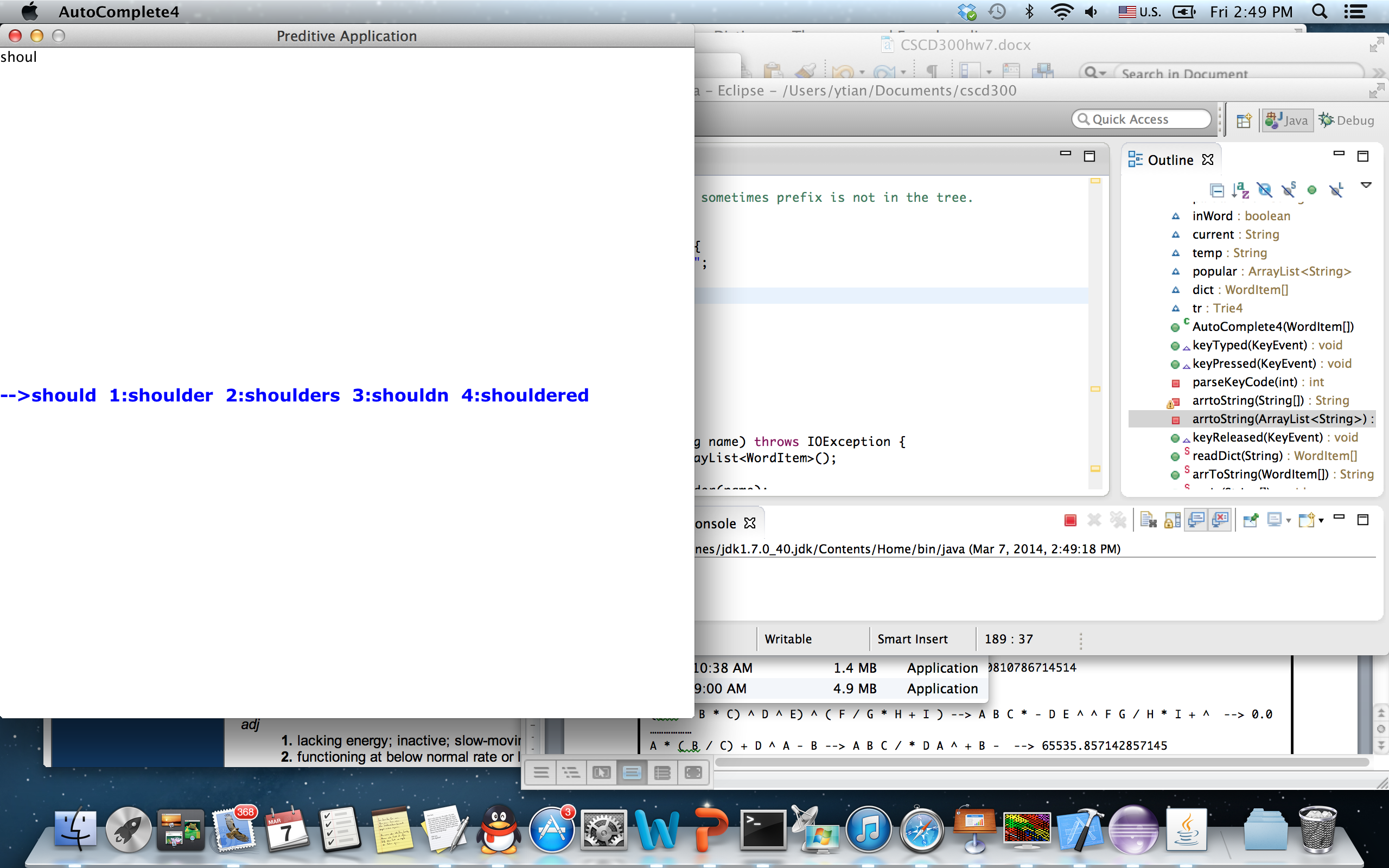
Hw7 Requirements

1. Put all your source code and input files for hw7 into a folder **hw7**.
2. You are required to use a Trie data structure to represent all words in a dictionary and all prefixes of each word.
3. Please read in **either** the dictionary.txt **or** dictSmall.txt file to build a Trie in memory. As pointed out, dictSmall.txt is small and suitable for debugging.
4. What you input in GUI is a prefix string, the prefix string has to be sent into a Trie’s method, which in turn returns a list of most frequently used words (symbolized as **S**) with that prefix. The size of **S** should be less than or equal to 9. If more than **9** words with that prefix are found, the Trie method only returns the first **9** most frequently used words.
5. Also, the list of words **S** that the Trie returns has to be **sorted** according to the frequency number of the words.
6. Display the list of most frequently used words (**S**) in the GUI window, so that a user can choose one of them. The most frequently used word (with highest frequency in S) is displayed at the first place (left most position). The word with second highest frequency in **S** will be displayed at the second place, and so on so forth.
7. If you use sort algorithm, you have to use either **merge** sort or **quick** sort, because they are quick. Please clearly document where and what sorting algorithm you used on top of that source file, if any.
8. If you search in an array, you are required to use **binary search, because it is quick**. On top of your source file, please document in which method you used binary search, if any.
9. Your final deliverable program has to satisfy the **real-time** requirement. That is, no matter how quickly a user types on the keyboard, your program is able to return the list of most frequently used words in a **real-time manner**. No sluggish display is allowed when using the provided dictionary.txt (the big one) as dictionary.
10. The time cost for reading in the **dictionary.txt** **and** building a Trie for that whole file should be less than 3 seconds. So please time your program using System.currentTimeMillis() and on **stdout** show a message about the time cost for reading in the file plus building the tree.

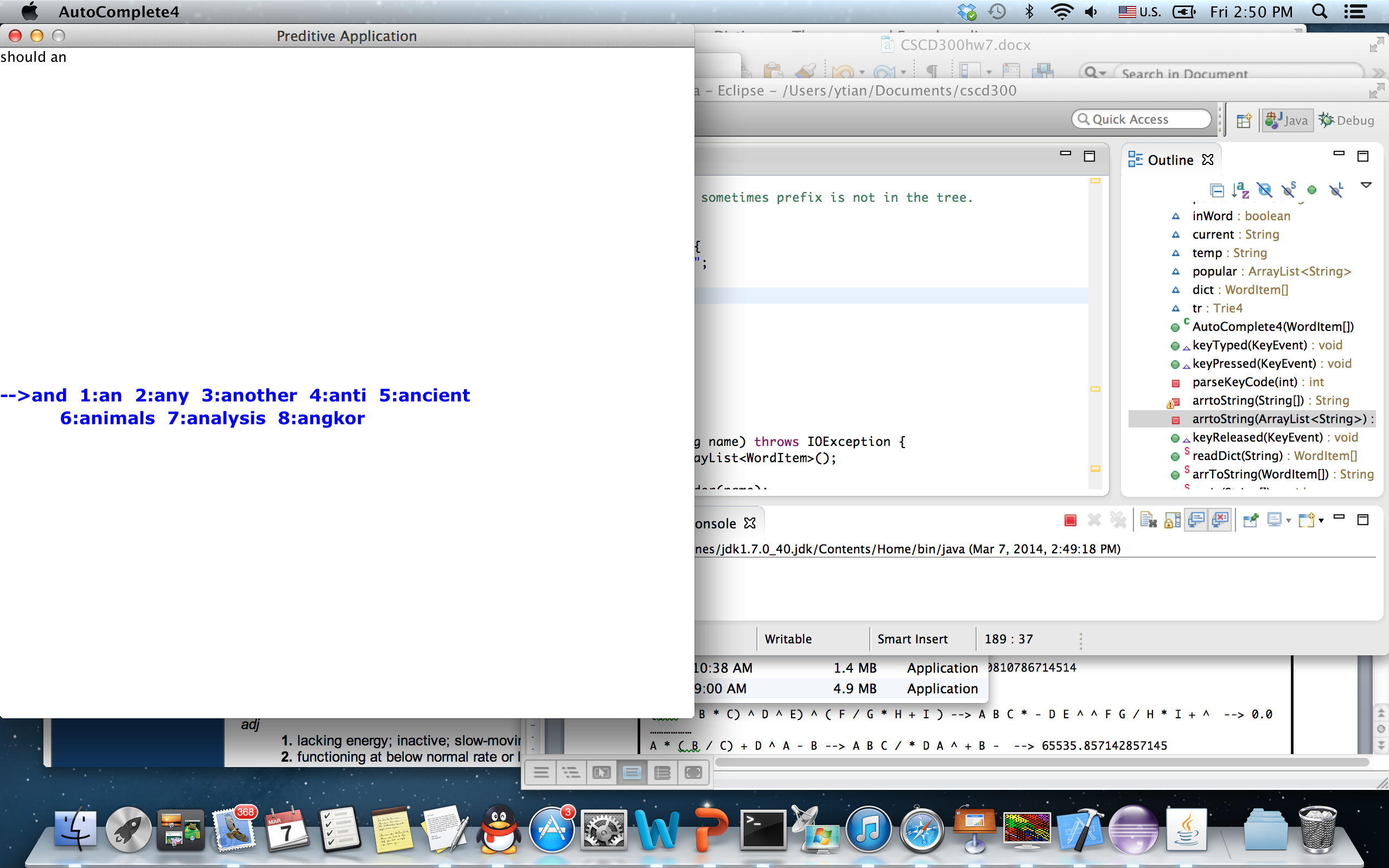
**Hint: please carefully design your solution, in terms of performance requirements. You probably need multiple data structures to work together to solve this problem.**

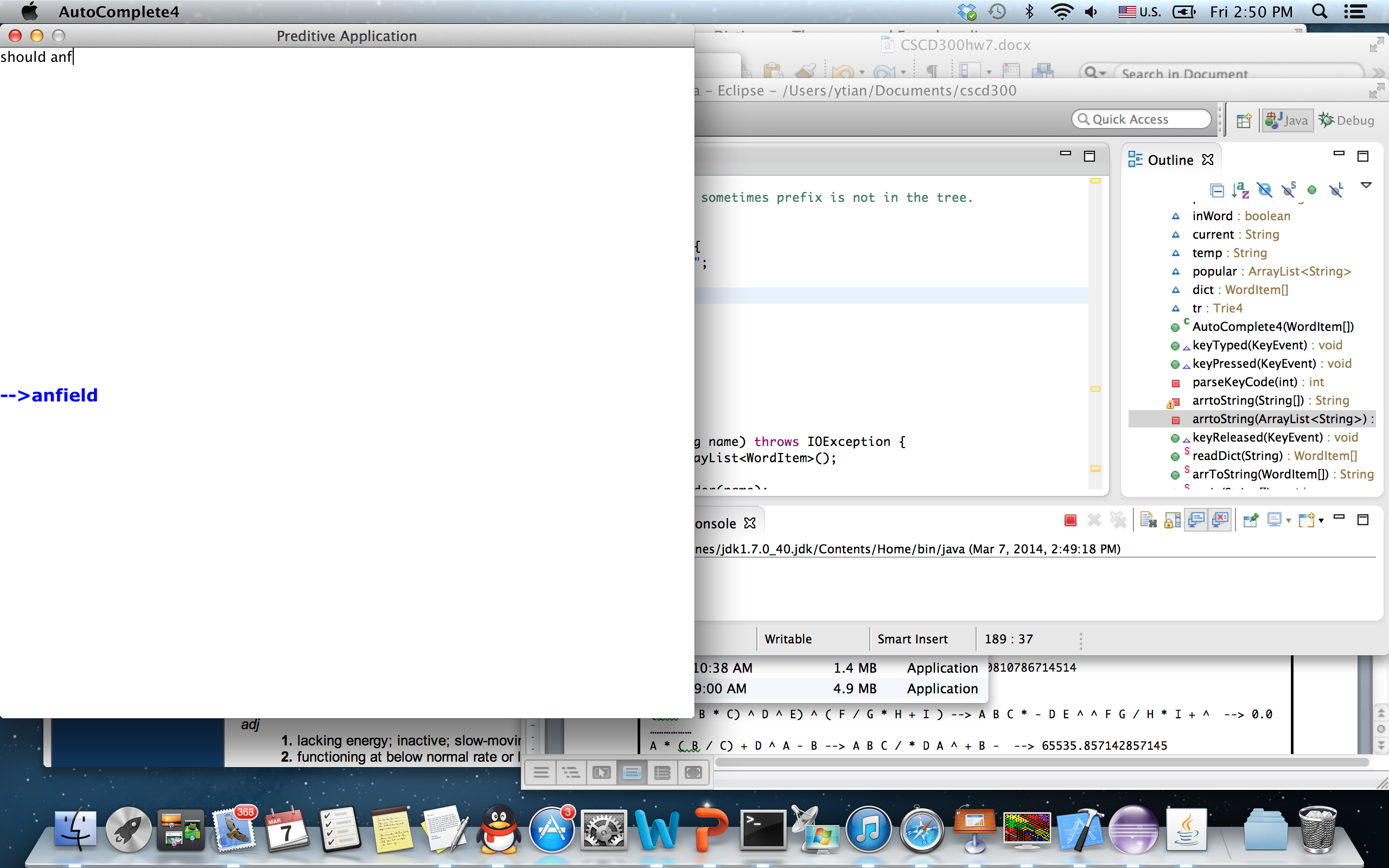
Test Cases when using dictionary.txt in the Trie,

Case 1, for prefix **shoul**

****

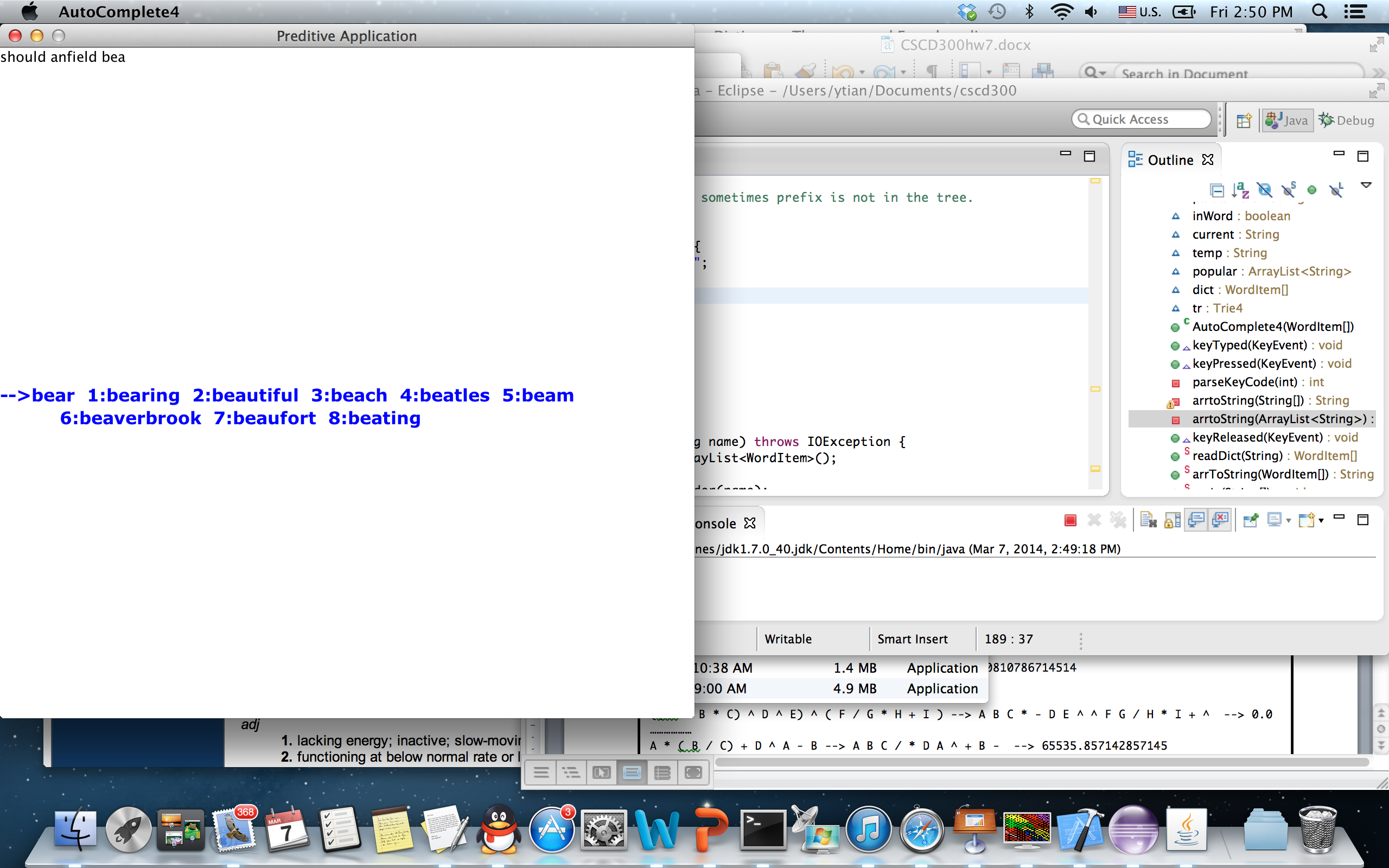
Case 2, for prefix **an**



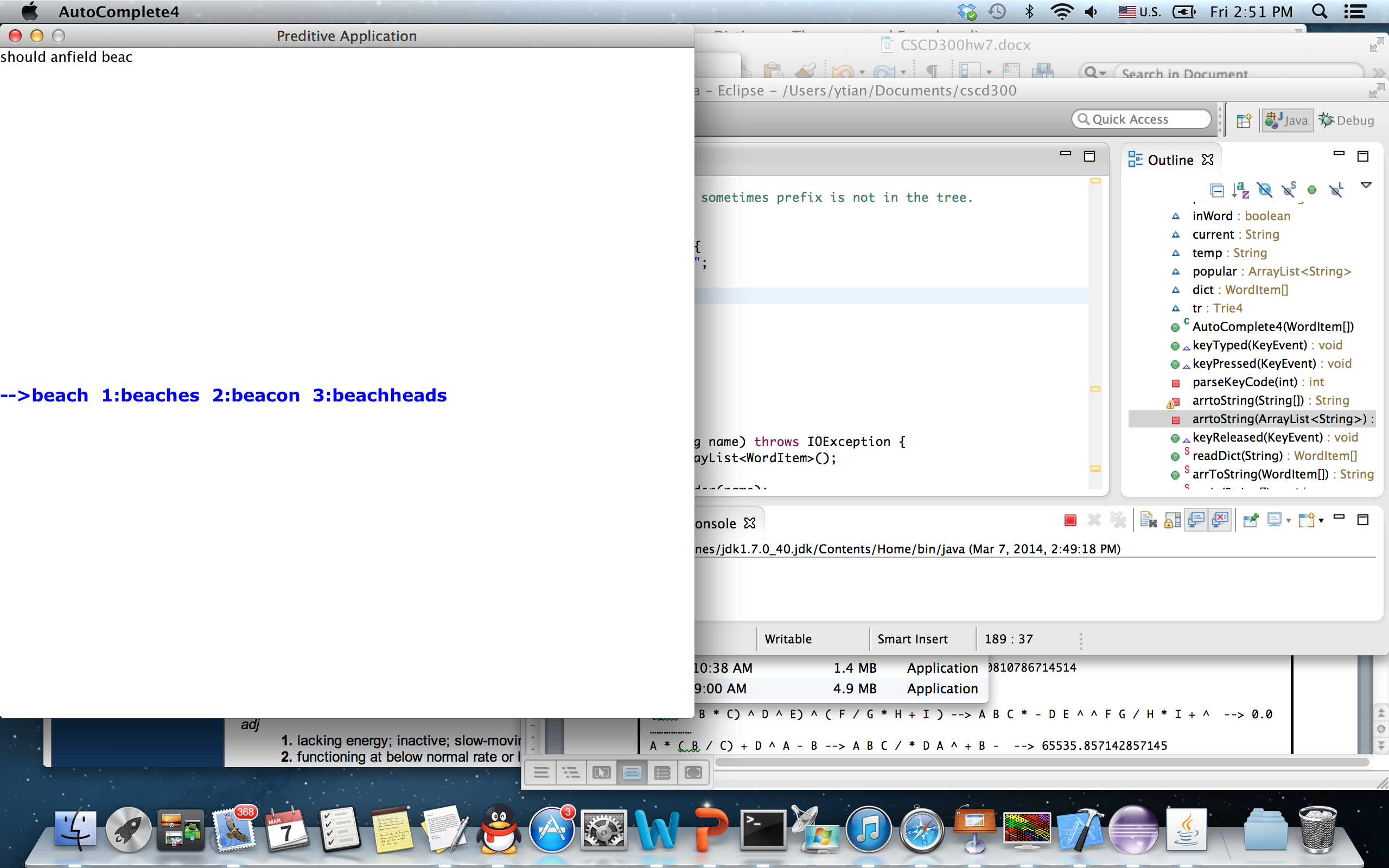


Case 3, for prefix **anf**

Case 4, for prefix **bea**



Case 5, for prefix **beac**

****

**Hw8 Problem Requirements (70 points)**

1. You are required to solve the same problem as homework 7 but using a different data structure. Put all your source code and input files for hw8 into a different folder than hw7, named **hw8**.
2. You are required to use a hashtable data structure to represent all words in a dictionary and all prefixes of each word. You have to implement your own Hashtable class.
3. Please read in **either** the dictionary.txt **or** dictSmall.txt file to build a hashtable in memory. As pointed out, dictSmall.txt is small and suitable for debugging.
4. You will use the same GUI code for your display. What you input in GUI is a prefix string, the prefix string has to be sent into the get() method in hashtable, which in turn returns a list of most frequently used words (symbolized as **S**) with that prefix. The size of **S** should be less than or equal to 9. If more than **9** words with that prefix are found, the get() method only returns the first **9** most frequently used words for the prefix.
5. Also, the list of words **S** that the hashtable returns has to be **sorted** according to the frequency number of the words.
6. Display the list of most frequently used words (**S**) in the GUI window, so that a user can choose one of them. The most frequently used word (with highest frequency in S) is displayed at the first place (left most position). The word with second highest frequency in **S** will be displayed at the second place, and so on so forth.
7. If you use sort algorithm, you have to use either **merge** sort or **quick** sort, because they are quick. Please clearly document where and what sorting algorithm you used on top of that source file, if any.
8. If you search in an array, you are required to use **binary search, because it is quick**. On top of your source file, please document in which method you used binary search, if any.
9. Your final deliverable program has to satisfy the **real-time** requirement. That is, no matter how quickly a user types on the keyboard, your program is able to return the list of most frequently used words in a **real-time manner**. No sluggish display is allowed when using the provided dictionary.txt (the big one) as dictionary.
10. The time cost for reading in the **dictionary.txt** **and** building the entire hashtable for that whole file should be less than 3 seconds. So please time your program using System.currentTimeMillis() and on **stdout** show a message about the time cost for reading in the file plus building the hashtable and computing all Most Frequently Used lists.
11. When running your hw8 program, the output is same as that of hw7.