Student Number: 7547821

Student Name: Gia Bach Nhu

Assignment 1

CSCI203

Here is my pseudo code:

```
// Structure to store word data and its frequency
struct WordData:
    char word[WORD_LENGTH]
    int frequency

// Read and process the input file
function readAndProcessFile(filename, wordDataArray, numWords):
    inputFile = open(filename)  // O(1)

    if inputFile is not open:
        print "Error opening file. Program will exit."
        exit

    while not endOfFile(inputFile):  // O(N) where N is the number of words
        word = readNextWord(inputFile)  // O(L) where L is word length
        processedWord = processWord(word)  // O(L)

        if processedWord is not empty:  // O(1)
            insertOrUpdateWordData(wordDataArray, numWords, processedWord)  // O(N)

    close(inputFile)  // O(1)

// Insert or update word data in the array
function insertOrUpdateWordData(wordDataArray, numWords, word):
```

```
    for i = 0 to numWords - 1:  // O(N)
        if wordDataArray[i].word equals word:  // O(L)
            wordDataArray[i].frequency += 1  // O(1)
            return

    if numWords < MAX_WORDS:
        wordDataArray[numWords].word = word
        wordDataArray[numWords].frequency = 1
        numWords += 1

// Sort the word data array by frequency and lexicographical order
function sortWordDataArray(wordDataArray, numWords):
    // Use a sorting algorithm here, like bubble sort, with a worst-case complexity of O(N^2)

// Display the results
function displayResults(wordDataArray, numWords):
    // Print the results

// Main program
function main():
    filename = input("Please enter the name of the input file: ")  // O(1)
    wordDataArray[MAX_WORDS]
    numWords = 0

    readAndProcessFile(filename, wordDataArray, numWords)  // O(N * L * N) = O(N^2 * L)
    sortWordDataArray(wordDataArray, numWords)  // O(N^2)
    displayResults(wordDataArray, numWords)  // O(N)

    return 0

// Run the main program
```

main()

My solution strategy:

1. We start by creating an array of structures to store word data and a variable to count the number of words.
2. We ask the user to provide the name of the input file that contains words.
3. We read words from the file, remove any punctuation and convert them to lowercase. For each processed word, we either increase its frequency if it's already in our array or add it as a new word if there's space.
4. We sort the array of word data by both frequency and lexicographical order. This means words with higher frequency will come first. If two words have the same frequency, they are ordered alphabetically.
5. We show the first ten words in the sorted list along with their counts. Then, we show the last ten words in the sorted list along with their counts.
6. Finally, we list out all the unique words along with their frequencies if they appear only once.

Complexity analysis of my solution:

1. Reading and Processing the File: Reading a word from the file involves a time complexity of O(L), where L is the length of the word. Processing the word (removing punctuation and converting to lowercase) also takes O(L). Since we process each of the N words in the file, the total complexity is O(N * L), where N is the number of words and L is the average word length.
2. Inserting or Updating Word Data: In the worst case, for each word, we iterate through the array of word data. This results in a time complexity of O(N) for each word. Since we process N words, the total complexity for this step is O(N^2).
3. Sorting Word Data Array: The sorting algorithm used is bubble sort. In the worst case, bubble sort has a time complexity of O(N^2). This step operates on the number of words in the array (N), and the number of iterations required depends on N.
4. Displaying Results: Printing the results takes a linear amount of time, O(N), where N is the number of words.

   Overall Complexity:

   - Reading and processing the file: O(N * L)
   - Inserting or updating word data: O(N^2)
   - Sorting word data array: O(N^2)
   - Displaying results: O(N)
     - ⇨ I think the biggest things that make this solution take time are when we're putting new words into the list and making sure they're in the right order. This means that if we look at the whole picture and think about the hardest situations (worst-case scenarios), it's like the time the program takes can grow roughly in proportion to the number of words squared.

List of all of the data structures used, and the reasons for use them:

1. WordData Structure: This structure is like a container that holds information about a word and its frequency.Because it helps keep related information (word and frequency) together in a neat package. This makes it easier to manage and understand the data.
2. Array (wordDataArray): An array is like a numbered list of items, where each item can be accessed using an index. Because the array is employed to store the WordData structures. Since we're dealing with words and their frequencies, we want to keep them in an ordered manner. Arrays allow us to do that while being relatively easy to work with. However, in terms of efficiency, searching for a specific word in an array can become slower as the array grows, which is something to consider.
3. ifstream: The ifstream is a tool that allows us to read data from files. Because we're reading words from a file, we need a way to access and process that file's content. The ifstream makes it possible to handle the reading process easily.

A snapshot of the compilation and the execution of the program on the provided “samplelong.txt” file.

```
Please enter the name of the input file: sample-long.txt
First ten words in the sorted list, along with their counts:
the : 131
of : 95
and : 94
to : 91
you : 74
a : 64
i : 63
he : 60
was : 48
mr : 47

Last ten words in the sorted list, along with their counts:
window : 1
withdrew : 1
within : 1
without : 1
wonderfully : 1
wore : 1
worth : 1
wwwgutenbergorg : 1
yet : 1
yourself : 1

Unique words:
abuse : 1
accomplished : 1
account : 1
acknowledged : 1
acquaintances : 1
act : 1
added : 1
addressed : 1
adjusting : 1
admiration : 1
admire : 1
admitted : 1
advice : 1
affect : 1
afraid : 1
afterwards : 1
agree : 1
agreed : 1
ah : 1
air : 1
almost : 1
already : 1
altogether : 1
altogethermr : 1
amends : 1
amiable : 1
amongst : 1
amusement : 1
angry : 1
ankle : 1
anyone : 1
anywhere : 1
arrival : 1
arrived : 1
ascertaining : 1
assemblies : 1
```

```
terms : 1
these : 1
third : 1
those : 1
thought : 1
threeandtwenty : 1
throw : 1
tide : 1
times : 1
tiresome : 1
title : 1
tolerable : 1
tomorrow : 1
toward : 1
towards : 1
trimming : 1
tumult : 1
turned : 1
turning : 1
unaffected : 1
uncertain : 1
uncommonly : 1
under : 1
understand : 1
understanding : 1
universally : 1
unless : 1
unlucky : 1
unreserved : 1
unworthy : 1
upper : 1
using : 1
various : 1
venture : 1
vexed : 1
vexing : 1
village : 1
violent : 1
visiting : 1
waited : 1
walking : 1
wasting : 1
wayswith : 1
week : 1
whatsoever : 1
whichever : 1
whom : 1
wifes : 1
window : 1
withdrew : 1
within : 1
without : 1
wonderfully : 1
wore : 1
worth : 1
wwwgutenbergorg : 1
yet : 1
yourself : 1

Process returned 0 (0x0)   execution time : 7.294 s
Press any key to continue.
```