CMT219 Assessment 2

Part 1 Documentation

Tom McVeigh

C21108264

Contents

[Introduction 3](#_Toc102930911)

[Output of the code 3](#_Toc102930912)

[Comparing the “Input219” file to the “google-10000-english-no-swears” file 3](#_Toc102930913)

[MergeSort of the gathered words 3](#_Toc102930914)

[Count of the Swaps and Comparisons that took place during the MergeSort 3](#_Toc102930915)

[Justification of the design of the program 3](#_Toc102930916)

# Introduction

This document will contain screenshots showing what the program developed for this assignment outputs. As well as this the document will also contain written justification for the design of the different elements of the program.

# Output of the code

## Comparing the “Input219” file to the “google-10000-english-no-swears” file

This screenshot shows the output of the code that compared the files together and gathered all of the words that made an appearance in both files.

A picture containing background pattern

Description automatically generated

## MergeSort of the gathered words

This screenshot shows the aftermath of the MergeSort function that was implemented and what the sorted list looks like. A picture containing background pattern

Description automatically generated

## Count of the Swaps and Comparisons that took place during the MergeSort

This screenshot shows the output of the code that counted every comparison and swap during the MergeSort, the total for both being 367.



# Justification of the design of the program

The program was created in stages, this allowed me to focus on one thing at a time and improve the development time. It started with getting the data from the files, this was originally planned to be completed by the use of a buffered reader to gather all of the words and add them to one big list, but with this approach I wrote my self in to a corner and couldn’t differentiate between the word that came from the different files. A new approach was then taken, using the Scanners to read the files and add the words from the Input file to one array list and to then add the other files list of words to another. I then used the toLowerCase function to make it easier to compare the words by ensuring that they are all lowercase meaning that the application worked on a case insensitive basis. I then took the list of words gathered from the scanner and ran them through a splitter algorithm that split the words from the sentences as well as removed the punctuation from both lists to have only the words left ready to be compared against each other. I think this allowed for a more accurate comparison as otherwise a comparison between the words “there.”, and “there” may not have been properly compared as one has a full stop that isn’t present with the other instance of the word leading to a false negative, removing all punctuation gets rid of this possibility.

The next part was then comparing the words together from the two lists to see if they were a match. The approach that I took to this was by using the .retainAll() method. As this looked at the “google-10000-english-no-swears” list and checked all of the words of the input file, it then altered the list that held “google-10000-english-no-swears” to delete all of the words that did not appear in input file. This approach was chosen as it was extremely fast to develop and works very quickly each time the program is run only producing a few errors in the form of collecting singular letters such as “g” and “e”.

Next is the MergeSort. This was a bit more complicated. It takes the array that needs to be sorted, then a number as a start point then the length of the array minus 1 to know where the end point is. It then separates the array depending on where the middle of the array is and then sends each half off to the merge function that sorts each section further. This approach allowed for the smaller sections of the lists to be sorted before being brought together to display the output list above, this implementation of a MergeSort is effective in getting the sorted list ready in a timely manner in a form that can be easily shown to the user using a print function. I also tried to create meaningful names for all of the variables, using words such as “input” for the input data, “first” for the first half of the array, “pos” to show the position available in the temp array. My thought in creating the merge sort this was to make it easier for myself to see what does what, and also to improve the maintainability as anyone looking at the code could understand the basic idea of what is related to what piece of data.

Implemented with the MergeSort are the counters that track the comparisons and swaps that take place over the duration of the merge sort. The variables were initialised at the top of the file to allow them to be updated through the MergeSort and called to be displayed during the final print statement. This was done so at a glance the user can easily see how many comparisons and swaps the algorithm had to carry out to ensure that the input data was thoroughly sorted.