Po Linn Chia
Prof. I. Fleming
CS2720 Final Project Report

## Project Description

My final project builds a Library that loads books from text files (some are provided in the ZIP, going from text1.txt to text9.txt). Upon loading a book, it gives the user statistics about the text: a word count, unique word count, list of top frequent words, and list of top frequent unique words. The user can then search the entire library for books that fit a word count or keyword criteria.

## Implementation

Library and Book are each class files.

A Library contains an array of up to 50 pointers to Books (the "shelf"). An array allows for users to get the latest or *nth* Book from the shelf with a single operation.

A Book is more complex, and contains several data structures:

- A hash table of common words (a "stoplist") to be excluded in some frequency counts;
- A tree containing all unique words from the input file;
- An array containing all unique words sorted by frequency;
- A balanced binary search tree sorted by frequency.

A hash table was used for the stoplist in order to reduce look-up times as much as possible. This save operations, for example, when getting unique frequent words: the most-used words are easily compared against the stoplist.

A tree was used to initially store words from the input file because it manages repeated words well. When a repeated word is inserted, the frequency of the word is simply updated. The resultant tree contains unique words and their frequencies.

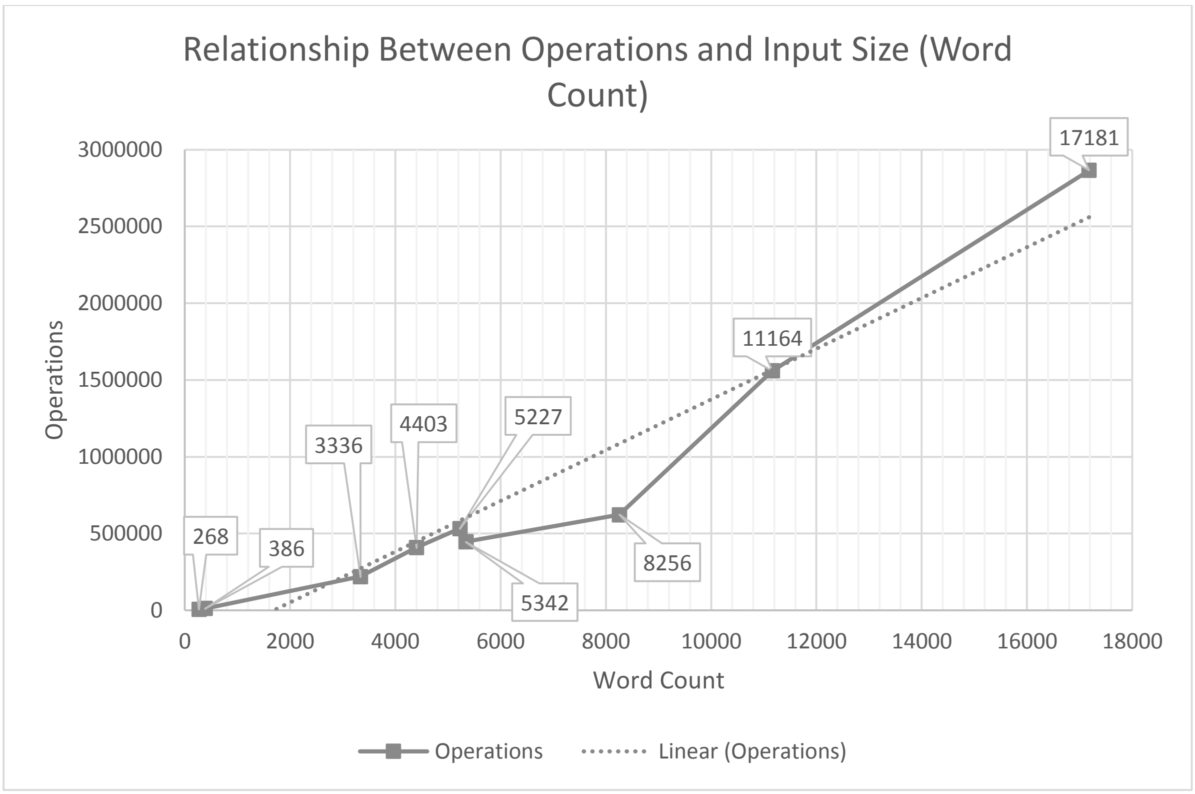
An array was used to store the contents of the tree. An array was chosen because of it can be sorted relatively easily, and because it made accessing the most and least frequent words trivial.

A balanced binary search tree was created as an experimental attempt at reducing search times for words. Since the initial tree may be extremely unbalanced and the array slow to iterate through, words were copied into a balanced binary tree by frequency. The *n/2*th most frequent word was used as the root.

## Complexity Analysis

For the purposes of this report, only the function to read in a new Book was analysed. This involves the following processes:

- Parsing words to be added (removing punctuation, transforming to lowercase)
- Adding words to the initial tree
- Copying the tree to an array
- Sorting the array
- Building a balanced tree from the sorted array
- Returning the title, word count, unique words, frequent words, and frequent unique words of the Book



| Total words | Total unique words | Operations |
|---|---|---|
| **268** | 135 | 7579 |
| **386** | 177 | 12244 |
| **3336** | 798 | 219632 |
| **4403** | 1145 | 409244 |
| **5227** | 1325 | 533046 |
| **5342** | 1171 | 446752 |
| **8256** | 1282 | 622171 |
| **11164** | 2247 | 1559377 |
| **17181** | 2993 | 2864830 |

Table 1: Total Words, Unique Words, and Operations

The complexity of this section of the programme remains relatively linear. One major factor affecting the number of operations is the density of unique words in the input file. More unique words require more operations due to copies being made to both an array and a second tree. For example, a file containing roughly 4,000 words but only 1 unique word requires only ~10,000 operations, compared to the 400,000 required by one with around 1,000 unique words (see table). This accounts for the dip in the graph at 8,256 words – that file (Grimm's Fairy Tales) had a relatively low density of unique words for its size.