Program Design:

The design of our Indexer consists of several attributes. Firstly, the dirent struct of the input file/directory is analyzed to determine whether or not it is a directory. If it is a directory, the program traverses through the directory and reads the contents of all files contained in the directory. The program uses a 1024 byte buffer to store the contents read from the files, and expands this buffer if necessary. Using this buffer, the program tokenizes any words it finds and stores them inside a Tokenizer. Then, the strings are extracted from the Tokenizer and are stored in a hashtable of the following structure:

1. The hashtable holds the information of every unique word that is found in these files using a hashNode to store each unique word.
2. Each hashNode points to a sorted list that stores the file name and frequency information of its attributed word.

Define n = total number of words contained by all the files found by the program
f = total number of files found by the program
n/f = average number of words contained by one file
n >> f : n is significantly larger than f

Overall, this procedure consists of the following phases:

Traversing the input file/directory: *Time Complexity:* O(f).

Reading n words: *Time Complexity:* O(n). *Space Complexity:* Worst Case: O(n) Average Case: O(1)

Storing n words into a Tokenizer: *Time Complexity:* O(n). *Space Complexity:* O(n) * sizeof(Token)

Storing n words into a HashTable: *Time Complexity:* Worst Case: O(n^2). Average Case: O(n). *Space Complexity:* 10000 + (O(n)*sizeof(hashNode))

Extracting n words from a HashTable and sorting them for Output (Quicksort):
Time Complexity: O(n) + O(n^2) Worst Case. O(n) + O(nlogn) Average Case
Space Complexity: O(n)

Total: *Time Complexity:* O(n^2) Worst Case. O(nlogn) Average Case
Space Complexity: O(n)*(sizeof(Token) + sizeof(hashNode))