Muhammad Usman Memon **Student ID: 6010878**

CSCI-203 - Assignment 1

Overall Solution: Since the problem required to read a text file and store the words and remove unwanted characters from it and only unique words to be stored and count their occurrences and how many words in total were in the file.

I chose to write the solution to this assignment in Python 3.9 since I was more comfortable in Python and have no experience in C or C++.

I read the file word by word and removed unwanted characters from words and stored the words in a HashTable as Key,Value pair of (Word, 1) with reference to HashIndex. The program checked each word read if it was in the Hash table it incremented the value of the word (key) by 1. If the word was not in the index then it is added as Key and a value of 1. The HashIndex is calculated as the sum of the ASCI codes for the word divided by the HashTable size (in this program 4000). The HashIndex helps to store words uniquely and easy to search making it a complexity of O(1) in general.

After the file is read and stored in a HashTable, a list is created and the program loops through all the values in the HashTable and only stores unique count values in the new list(uniquelist). The **uniquelist** is sorted by mergesort and then we loop through it reverse order. In the loop we filter HashTable values where the keyValue (countWords) equals to the unique value iter. The filtered KeyValues are inserted into a temporary list **tatti** that is passed into mergesort where the words are sorted alphabetically, then appended to final list **testin** and this process is repeated till the loop ends.

Finally, we have the list testin that has the all the words sorted by their count values first then alphabetically.

Standard Algorithms used: The standard algorithms used is the Merge Sort algorithm to sort the lists with unsorted words and values(counts). The Merge Sort algorithm has a complexity of O(NlogN) thus making it as one of the affective ones to use from the sort algorithm arsenal. Moreover, it was the one that I could understand better and would suit my program based on my approach.

Data Structures Used:

1. **Nested Lists** are used in the HashMap class which implements a HashTable algorithm. The Nested lists are used to avoid cluttering at HashTable nodes, such as in the event of hashIndex collision with some keys having the same HashIndex, can be stored on the same HashIndex as an Array. The complexity of a HashTable is averagely O(1) which is good but due to collisions it does do a bit more than that on times. For keys with same HashIndex the HashMap.get() will loop through the array on that HashIndex and compare the key.
2. **Lists** are used to store multiple keys, values or both as tuples in the HashMap class under the keys,values and items functions to return a sequence of values/keys/items. Lists are also used to store unique count values of words to loop through later for sorting words according to counts in decreasing order. And Lists are also used to store the final sorted array/list after applying merge_sort. The reason for storing the final sorted result in a list is so that it is easy to fetch the starting and final results by using the slicing feature of lists.
3. **HashTable** was implemented using a modulo based HashIndex for storing data. Since HashTables are very affective with average complexity of O(1) since it search using the

unique HashIndex it made searches very easy. Plus it also made stoing unique word counts as keyValue pairs easy and efficient. With the HashTable , the program built a HashTable with 4000 nodes meaning that there were spaces for 4000 unique indexes and the capacity to store multiple keyValues in each HashIndex. Since the maximum number of different words to be given was 50000, we used 4000 meaning that in the case of collision on HashIndexes the program would store on average a list of length 10 to 20 on each index which is not complex to loop through.
Thus the nodes were kept to 4000, if the nodes are increased it reduces the program speed unnecessarily because of the extra amount of nodes most of which would remain empty and take up space.

Output Produced :

```
Please enter the file name with extension: sample-long.txt
The filename you have entered is sample-long.txt
The results will be display below for this file

The total number of words in the file are : 3441

The number of unique words in the file are : 875

The first 15 words are :
 Word : the  , Count : 131
 Word : of  , Count : 95
 Word : and  , Count : 94
 Word : to  , Count : 91
 Word : you  , Count : 74
 Word : a  , Count : 64
 Word : i  , Count : 63
 Word : he  , Count : 60
 Word : was  , Count : 48
 Word : mr  , Count : 47
 Word : his  , Count : 45
 Word : in  , Count : 44
 Word : that  , Count : 44
 Word : with  , Count : 43
 Word : her  , Count : 41
```

```
The Last 15 elements of the list are:
 Word : week  , Count : 1
 Word : whatsoever  , Count : 1
 Word : whichever  , Count : 1
 Word : whom  , Count : 1
 Word : wife’s  , Count : 1
 Word : window  , Count : 1
 Word : withdrew  , Count : 1
 Word : within  , Count : 1
 Word : without  , Count : 1
 Word : wonderfully  , Count : 1
 Word : wore  , Count : 1
 Word : worth  , Count : 1
 Word : wwwgutenbergorg  , Count : 1
 Word : yet  , Count : 1
 Word : yourself  , Count : 1
```

References:

1) Merge Sort - Data Structures & Algorithms Tutorial

https://www.youtube.com/watch?v=nCNfu_zNhyI&list=PLeo1K3hjS3uu_n_a__MI_KktGTLYopZ12&index=17

2) Chaining HashMap Implementation
https://www.geeksforgeeks.org/implementation-of-hashing-with-chaining-in-python/