1. Preprocess(filename)

Algorithm:
Open the file and then convert the lines to strings, start a for loop to check the characters are between "a" and "z" so that punctuations are eliminated and the word is concatenated to mystr and after every character either a space, newline or tab
is checked and if the condition meets and a word is formed then the word is checked for if it is in the list of auxiliary verbs and then the word is appended to the final list and returned and then the string is again emptied and re-initialized for the new word.

precondition: a text file
post-condition: a list of word strings

Time Complexity: O(NM) in worst case where the N are the words and M is the maximum length of the list.

Short summary:

> *The first for loop executes in O(N) where N is the number of strings*
>
> *The append takes O(1) times to append the word into the final list*
>
> *The operation "if mystr not in aux_words" remains O(1) because the list of those words is constant.*
>
> *So the total complexity for the function is O(NM) where the rest of the complexities can be eliminated such as O(1) and O(/n) in addition will result in 2 O(N).*

Space Complexity: O(NM) where the N are the words and M is the maximum length of the list of words that will be appended in to the final list so the space needed will be O(NM)

2. wordSort(arraywords):

Algorithm:

At first the length of the longest word is checked in the array and returned. A frequency_table is created to store the word according to the current index which is achieved by getting the ascii of the current character, then the remaining words with length lower are placed into the list of ignore table and then the loop goes on until all the words are added to the frequency table and the ignore table. Then the main array is re-initialized to array of type string and then the words in the ignore table are appended back to the main array. Then the word in the frequency table is appended to the main array. This goes on until the bucket stores all the words according to the index of the character which goes on a decrement level from the length of the longest word to 0, and then it returns a final list of sorted words.

precondition: a list of unsorted array
post-condition: a list of sorted array

Time Complexity: O(NM) worst case. Where M is the maximum number of characters in a word and the counting sort is performed on each character at a time. The loop here gets decremented every time counting sort on the right most to the left most, N is the input size of list.

Short summary:

The maxlength function executes in O(N) time. Where N is the size of the list

The for loop runs N operation times with M the length of the string

The ignore table loop runs O(N) time and the pop inbuilt is O(1)

The for loop for appending the item back to the main list takes O(N) complexity and appending the item is O(1)

Space Complexity: O(NM), where N are the words and M is the longest word in the file. A frequency table of size 26 is created for the alphabets a to z using ascii. So, there are n words and the length of the longest word is M so a total of MN space is required.

3. wordcount(sortedlist):

Algorithm:

The algorithm uses two pointers, the first pointer is at 0 and the second at 1. To append the last word as well, I used a check where if in the end both i and j pointers are the same because of decrementing j when it reaches the last word. Then it appends the last word and its count and appends to the main list and the function stops. In the normal loop when the pointer has not reached the last word, the pointer at i and j compares the word and if they are the same the word gets appended to the temp list and the count is 2, and the flag is false, else the words are not the same, then it checks if there is any word currently stored in the temp list, if yes then it appends it to the main list and then empties the temp list and since the words were not the same the i pointer becomes the j pointer and j pointer is incremented by one to the next value. Then the loop again begins and in the same order the entire count is found and returned. The function stops when the i and j pointers are the same value as I decrement j as mentioned above and the final word is appended with its count to the list.

precondition: a list of sorted words

post-condition: a list with two values, total number of words and a list of word count

Time Complexity: O(NM) in worst case where the N are the words and M is the maximum length of the list.

The while loop here iterates through the length of the sorted array so O(m) and the entire array has to be checked for frequencies. So there is no exiting early on. The append operation takes O(1) complexity.

Space Complexity: O(NM) where the N are the words and M is the maximum length of the list. There are two lists created the main list and the temporary list, the temporary list gets empty as the words get appended alongside the count in the main list. So the space required here will be no more than O(MN)