Report

Question 1:

Sorting Method for LIST A (Array of Words): The program uses a custom implementation of the Merge Sort algorithm to sort the array of all words from the input file. Merge Sort is a divide-and-conquer algorithm that breaks the array into smaller arrays, sorts those arrays, and then merges them back together in sorted order. This method is particularly effective for sorting because it is efficient for large datasets, and I don't know how big the input files used to test my code will be.

Sorting Method for Letters within Words: To determine if two words are anagrams, the program sorts the characters within each word using the same Merge Sort algorithm. This sorting standardizes each word to its alphabetically sorted form, allowing for easy comparison between words to check for anagrams.

Question 2:

The overall time complexity of the program can be estimated by considering the two main operations: sorting the list of words (LIST A) and sorting the characters within each word.

- Sorting LIST A: Sorting N words using Merge Sort has a time complexity of O(N log N).
- Sorting Characters within Each Word: Each word is sorted individually, with a time complexity of O(L log L) for each word, where L is the length of the longest word. Since each of the N words undergoes this process, the total time complexity for this part is O(N * L log L).

Combining these two operations, the total estimated time complexity of the program is O(N log N + N * L log L). In scenarios where N is significantly larger than L, the N log N term dominates. Conversely, if L is large (but still much smaller than N), the N * L log L term can become significant, especially for very long words. However, for practical purposes and typical datasets where L is much smaller than N, the overall running time can be closely approximated by O(N log N), acknowledging that the sorting of characters within words adds an additional factor based on the length of the words.

Final answer: O(N log N)