GRADING SUMMARY

Name: Niyousha Raeesinejad

ID: 30038699

PROGRAM:		/20
REPORT:		/20
TOTAL		**/40**

PROGRAM

Program builds correctly	Yes	No
Runs: completes with zero run-time errors	Yes	No

IMPORTANT: Source code that does not compile or produces run-time errors will receive 0%

Criteria	TOTAL
Functionality: produces correct output	
Readability: code is clear and easy to read	
Modularity: code is easy to extend, or use	
Generality: easy to extend to more sophisticated applications	
Documented: code classes/functions/inline commented	
TOTAL:	**/20**

REPORT

Criteria	TOTAL
Efficiency: can the program handle a large input file	/4
Correctness: all guidelines followed input/linked list usage/etc.	/4
Question #1:	/4
Question #2:	/4
Question #3:	/4
TOTAL:	**/20**

Questions

1. Describe the sorting method(s) used in your program – i.e., sorting the array storing all words from the input file (LIST A, step #2) and sorting the letters of two words from LIST A (step #2) to determine whether they are anagrams. Justify your selection of algorithm(s).

   This program uses the quick-sort algorithm to sort the 1-D array containing all words read from the user/input file (List A) to take into account larger inputs while maintaining an expected running time of $O(nlog(n))$.

   This algorithm is implemented through method *sortListA*, which calls method *trivialList* if the array contains 3 elements or less, sorting that array through brute force search. Otherwise, it calls method *divideList* to partition the array and then recursively calls itself to keep breaking down the array until the size of the sub-array is trivial and easily sorted, which is then returned. The method *divideList* functions by rearranging all array elements smaller than the element at a particular index to the left of that index and similarly all array elements larger than that element to the right of its index. This index serves as a pivot and is pre-determined by calling method *median* which finds the median element between the elements at the first, middle, and last indices of the array. The reason for selecting the pivot this way rather than assigning it to the first, last, or a random index in the array is to ensure that the pivot would still be assigned to the median element in the worst-case scenario of the array of inputs being reversely sorted.

   In order to sort the letters of two words from List A to determine whether they are anagrams, this program uses Selection-Sort; This particular elementary sort was chosen based on the assumption that each word from List A would be comprised of less than 100 letters and that it has the lowest running time compared to Bubble-Sort – which involves too many unnecessary swapping for a small input – and Insertion-Sort – whose performance differs only by a constant.

   The Selection-Sort algorithm in this program is implemented in method *sortWord*, which works down the array from its last index, finding the largest character and inserting it along the way.

2. Let N be the number of words in the input word list and L be the maximum length of any word. Give an estimate of the big-O running time of your program. Justify your answer.

   The estimated big-O running time of this program is $O((LN)^2)$, which was determined by considering the individual estimated big-O running times of each method called in order, as demonstrated by the following table:

Line Range	**Big-O Running Time**	**Comments**
18 – 30	$O(LN)$	Method *readInput* consists of a while-loop and calls method *convertToArray* which also contains a while-loop, indicating an estimated linear running time.
163 – 171	$O((LN)^2)$	Method *sortListA* calls *trivialSort* under a certain condition, which has a constant running time due to consisting of only primitive operations. On the other condition, it calls method *divideList* which has an estimated quadratic running time due to containing nested while-loops. Method *sortListA* then calls itself recursively, indicating an estimated linear running time for the recursion segment. The overall estimation of these individual running times is then quadratic.
234 – 257	$O((LN)^2)$	Method *sortListB* uses nested for-loops to search for anagrams by calling method *isAnagram* and this method compares 2 words after they are sorted through method *sortWord*, which uses nested for-loops to implement a Selection-Sort algorithm. The nested for-loops indicate an overall estimated quadratic running time for method *sortListB.*

3. What is the big-O running time when the input word list contains only two words?

   In the case of the input word list containing only two words, the estimated big-O running time for method *sortListA* would be $O(NL) = O(2L) \approx O(L)$ since it would only call method *trivialList* which has an estimated linear running time. However, method *sortListB* would still have an estimated big-O running time of $O((LN)^2) = O((2L)^2) \approx O(L^2)$, because it would nevertheless call method *sortWord* which contains nested for-loops for its implementation of Selection-Sort. Therefore, the overall estimated big-O running time of this program would remain as $O(L^2)$.