1)

For list A of words I implemented mergeSort as it is efficient and runs on O(n logn) . It works also for large inputs of words, compared to selection sort, bubble sort and insertion sort that has quadratic running time and would break down for large input values, and because we were only asked to implement one sorting function, it will be safer to use a sorting method that works for larger input values. I also implemented a method to count the number of lines in the input text file to create a 1-D array accurate to the size needed.

For sorting letters of two words to determine if they were anagrams, I implemented the following:

The word(string) would be passed to sortString(String) method, I would create a char array of the string and then sort the array using a built in method (Arrays.sort(char [] arr)), and then return a new sorted string from the sorted char array in order to be compared to other word string using the "string.compareTo(string)" method. This was the easiest algorithm for me to use, but it might cost some time as it will have to do this for each word string in list A.

2) To get the big-O of my code we have to study these specific methods/procedures where N is the number of words in the list:

a) mergeSort : O(NlogN)

b) build(in ArrayList class): $O(N^2)$

LN#	build procedure	Cost	Times	comments
1	For(int i=0,i<N;i++)	C1:1	N+1	Basic for loop
2	New SinglyLinkedList()	C2:1	N	Initializing new SinglyLinkedList
3	For(int j=i;j<N;j++)	C3:1	N*(N/2)	Nested for loop,
4	If(sortString.compareTo(sortString))	C4:L^2	N*(N/2)	comparison
5	List.add	C5: 1	N*(N/2)	Adding link to list

F(n)=C1*N+C1+C2*N+C3* N*(N/2)+C4* N*(N/2)+ C5*N*(N/2)

In fact, since each call to compareTo in line4 in my build procedure performs a linear comparison between two strings , and it includes a call to sortString(discussed below) that has a quadratic running time L^2 , then the cost C4=L^2. Then my build procedure will have a running time of $O(L^2N^2)$.

c) Array.sort(char []) (in ArrayList class in sortString method) : it's a built in function based on selection sort that has quadratic timing, and it is $O(N^2)$, but because we are given that the max length of the word is L, then our total cost of the sortString method can be cut to $O(L^2)$, to be a constant.

Abstractly, my program has f(n)=L^2N^2+ NlogN+L^2, which implies that my running time will asymptotically be $O(L^2N^2)$, that is the dominant term.

3)

If the list only contained 2 words, the running time can be assumed to be constant, and it can be cut down to $O(4L^2)$.

References:

CSPC319-lecture slides : 04-Sorting Algs. (4) Merge-Sort—1 sld-pp

Tutorial lecture slides: https://pages.cpsc.ucalgary.ca/~mdmamunur.rashid1/CPSC319-W19.html