Patrick Quan
10152770
CPSC 319

CPSC 319 - Assignment 2: Linked Lists and Sorting

Complexity Analysis

Source Code	Operations
for (int j = i + 1; j < wordArr.length; j++) {	1 + (n + 1) + n
tempI = wordArr[i].toCharArray();	3n
tempJ = wordArr[j].toCharArray();	3n
sorting.insertionSort(tempI);	n + n OR n + n^2
sorting.insertionSort(tempJ);	n + n OR n + n^2
tempStringOne = new String(tempI);	2n
tempStringTwo = new String(tempJ);	2n
if(tempStringOne.equals(tempStringTwo)) {	3n
wordList[i].add(wordArr[j]);	3n
wordArr[j] = "";	2n
}	
}	
wordList[i].alphaSort();	n + n OR n + n^2
}	
}	

Portion of the code that determines big-O notation (grey): $[(1 + (n + 1) + n] \times 2n \times [(1 + (n + 1) + n] \times (20n + 2n^2) \times (n + n^2)] = O(n^2)$

1. The worst case complexity for the algorithm to determine if two words are algorithm is when the words are significantly large, and the letters within the words are in descending order (Z — A) resulting in a big O notation of $O(n^2)$. This is due to the algorithm to determine if two words are an anagram rearranging the letters within the words alphabetically, resulting in the two words to be anagrams if they are equal to each other once they have been alphabetically sorted. Thus, as the number of letters in each word, *k,* approaches infinity, assuming that the letters are in descending order, the insertion sorting algorithm will process k^2 elements due to every element having to be inserted into the proper position within the array.
2. The big-O characterization for this anagram program is $O(n^2)$ as calculated above. This is due to the number of input words as well as the length of the words increasing the length of processing time by n^2, where *n* is the number of elements within the input file. This is due to insertion sort being the less efficient sorting algorithm in average and worst cases; due to processing each element individually with previous elements to determine the proper placement. If the input file anagrams approaches 0, the processing time increases

significantly due to a single element being compared to every element to every other element to determine if they are anagrams, rather than just a few until an anagram is found. This is also evident in the total runtimes, in which input file medium has ~4 times as many words as input file small, but takes ~16 times (4^2) as long to process.
ie. input file differences = medium ÷ small = 105989 ÷ 24867= ~4
runtime differences = medium ÷ small = 822.4 ÷ 56.2 = ~16 = 4^2 = $O(n^2)$

Input file (# of words)	**Anagram sort time (s)**	**Quicksort time (s)**	**Total runtime (s)**
Example_1 (8)	3.44×10^{-4}	1.29×10^{-5}	0.0748
Example_2 (10)	4.24×10^{-4}	1.75×10^{-5}	0.0768
Example_3 (25124)	23.0	0.0274	24.5
Small (24867)	55.2	0.0125	56.2
Medium (105989)	813.8	0.0480	822.4
Large (157858)	1999.2	0.0593	2003.6