Assignment 1 – Written Explanation

FIT2004 S2 2018 Author ID: 27799964

Task 1: Largest group of anagrams

Step	Time complexity (worst-case)	Space complexity (worst-case)
Open the dictionary file. Generate a list where each entry contains two items. The first item of each list should correspond to a word from the dictionary; the second item is the word's 'key', consisting of the same letters in the original word sorted in alphabetical order.	O(M*N), where N is the number of words in the input dictionary and M is the length of the longest word in the dictionary.	O(N)
Sort this list alphabetically by key, using radix sort.	O(M*N) for at most M letters and N words.	O(M*N) for M output lists being generated, each of length N.
Iterate through this sorted list, keeping track of sequences of entries with the same key. A list variable called longestAnagrams keeps track of the longest sequence of entries with the same key and is updated as necessary.	O(M*N) for N words and at most M comparisons to check if key strings are equal.	O(W) where D is the number of words in the longest list of anagrams.

The overall worst-case time complexity is O(M*N). The worst-case space complexity is also O(M*N).

Task 2: Scrabble words finder

Step	Time complexity (worst-case)
The query string and the sorted list of (word, key) pairs from the previous question are passed to a function getScrabbleWords(). Using counting sort, the letters in the query string are sorted in alphabetical order.	O(k) where k is the length of the query string.
Using binary search, an instance of the sorted query string, 'foundIndex', is located in the sorted (word, key) list.	O(klogN); for each of the logN steps in binary search, the comparison between two strings takes O(k).

The word at foundIndex is appended to an output list. All words matching the target query string that occur BEFORE foundIndex are added to the output list, by iterating backwards. All words matching the target query string that occur AFTER foundIndex are added to the output list, by iterating forwards.	O(W), where W is the number of words in the output.
The output list is sorted alphabetically using radix sort and printed.	O(k*W)

If we assume that k*W < k*logN (for a large N and small k and W), the time complexity of this algorithm is O(klogN + W).

Task 3: Query with wildcard

Step	**Time complexity (worst-case)**
1. The query string and the sorted list of (word, key) pairs from the previous question are passed to a function getWildcardWords().	O(1)
2. A new string is created consisting of the query string, plus the first letter of the alphabet. The letters in this new query are sorted alphabetically using counting sort.	O(k) where k is the length of the query string.
3. Using binary search, an instance of this new sorted query string, 'foundIndex', is located in the sorted (word, key) list.	O(klogN); for each of the logN steps in binary search, the comparison between two strings takes O(k).
4. The word at foundIndex is appended to an output list. All words matching the target query string that occur BEFORE foundIndex are added to the output list, by iterating backwards. All words matching the target query string that occur AFTER foundIndex are added to the output list, by iterating forwards.	O(W), where W is the number of words in the output.
5. Steps 2 - 4 are repeated 26 times, each time with the search target consisting of the original query plus a different letter of the alphabet.	O(1) as 26 is a constant.
6. The output list is sorted alphabetically using radix sort and printed.	O(k*W)

If we assume that k*W < k*logN (for a large N and small k and W), the time complexity of this algorithm is O(klogN + W).