Algorithms Analysis

Assume:

- n is the number of words in dictionary
- k is the number of words need to be corrected
- a is average length of English words

TASK 3

For task 3, my approach is to build a hash table on dictionary, it cost Θ(n), and then generate all candidate words which has 1 edit distance to each word in document, it cost c = Θ(k*(26*(a+1) + 26*a + a)). Each look up in hash table cost Θ (1), therefore, it cost Θ(c*k). The overall cost of this algorithm is Θ(n+k^2). When k is small, the cost is just Θ(n).

Alternative Approach:

Linear search through the dictionary and compare each word in document with each word in dictionary. The cost of this approach is Θ(k*n). When k is small, the cost is just Θ(n). However, the reason that I chose the above approach is that when k becomes larger and larger, the cost of linear scan increases dramatically.

TASK 4

For this task, my approach is also **building a hash table on dictionary** first, it costs Θ(n), and then iterate through the document linked list. For each word document, I initially check if it is in the hash table, this cost Θ(1) for each word. If it is not in the table, then generate all possible candidate words for 1 edit distance and find if each of them in dictionary. At this stage, the cost is approximately Θ((26^(a+1)*k)). If the correct word still not found here, it will iterate

the through the dictionary, to find the first found two edit distances or three edit distances word. This costs $\Theta(n)$ for each word. If we assume every word reach the final point of the approach, when $n>>k$, the overall cost is $\Theta(k*n)$, when $n<<k$, the overall cost is $\Theta((26^{(a+1)*k))}$. The drawback of this approach is when the dictionary is large and most of the word does not exist in dictionary, which including the edit distances=1,2,3 version of words, this approach will take a lot of time.

Alternative Approach:

Instead of iterating through the whole dictionary when the word with edit distance 1 are not found. I primarily create one more hash table at the beginning. This hash table only contains all possible word which can come up with only deletion with edit distance of 2 and 3. The hash table use those word and dictionary word itself as key, and two things as keys, which are the pointer to the bucket which store the dictionary word in another table and the edit distance from its original word. If collision occur, if they have same edit distance, discard the last one, if they have different edit distance, use separate chaining to deal with. In this way, the approach only need to generate the candidate of the document word which has 2 and 3 edit distance with only deletion. It dramatically reduces the candidate word number, and each step is hash table lookup, which is super quick. The reason that I didn't pick this approach is that considering the time constrain, it difficult to finish this well before the due date. Another reason is that I cannot prove this mathematically that it covers all the possible words, but it works well empirically based on the experience of using another language to implement this method.