Dynamic Programming – Project

Samuel Escalera Herrera

1. Improved findMisspelledWords method, no longer uses calculateSimilarity method now uses levenshtein directly, also uncouples the nested loop by making the words in text1 split before iterating and comparing them with text2.

2. The Levenshtein distance, which forms the basis of the DP algorithm, provides a robust way to measure the similarity between two strings by calculating the minimum number of operations (insertion, deletion or substitution) required to transform one string into the other.

3. The time complexity of the solution lies mainly in the implementation of the Levenshtein distance calculation within the calculateSimilarity method. The Levenshtein distance algorithm has a time complexity of O(m*n), where m and n are the lengths of the two input strings. Since the algorithm traverses each character of both strings once, the time complexity is directly proportional to the product of their lengths.

4. One solution I can think of is the Rabin-Krab algorithm, using hashing to compare the text substrings of the 2 txt files, however hashing could not help us to find misspellings or changes in word order. In conclusion, although there are other algorithms that are better at comparing text substrings, this algorithm could not help us to find spelling errors.