Time Complexities of Levenshtein Distance Algorithm:

The provided code implements the Levenshtein Distance algorithm to count the number of insertion, deletion and substitution operations needed to transform one string into the other. Let's break down the time complexities for different cases:

**Best Case:**

* The best-case scenario for the Levenshtein distance algorithm occurs when the two input strings are identical.
* In this case, the Levenshtein distance is 0, indicating that no edits are required to transform one string into the other.
* Since no edits are needed, the algorithm can terminate early without performing any additional computation beyond the initial string comparison.
* The best-case time complexity for the Levenshtein distance algorithm is O(N), where N is the length of the shorter input string.
* This occurs because the algorithm only needs to traverse the length of the shorter string once to determine that no edits are necessary.

**Worst Case:**

* The worst-case scenario for the Levenshtein distance algorithm occurs when the two strings being compared are completely different, i.e., they have no common substrings.
* In this case, the algorithm has to consider every character in the first string against every character in the second string.
* So, it results in a time complexity of O (M \* N), where 'M' is the length of the first string and 'N' is the length of the second string.
* If two strings of length m and n are completely different, Levenshtein distance algorithm would need to consider every character pair to compute the distance.

**Exceptional Case:**

* An exceptional case arises when dealing with different lengths of strings, particularly when one string is empty.
* In this case, if one string is empty, the Levenshtein distance would be equal to the length of the non-empty string.
* This is because to transform an empty string into a non-empty string, you would need to perform as many insertions as there are characters in the non-empty string.
* In such cases, the algorithm may need to perform character comparisons for non-empty string, leading to a time complexity similar to the best case.
* In summary, the Levenshtein distance algorithm has a best-case time complexity of O(N), a worst-case time complexity of O(N×M), and an exceptional case time complexity that can approach the best case in scenarios with different lengths of strings.
* It's important to note that while the Levenshtein distance algorithm is efficient for moderate-length strings, there are optimizations and variations of the algorithm that can further improve its performance in certain scenarios, such as reducing the space complexity or handling specific types of edits more efficiently.