Documentation

Intuition

The Wiggle Subsequence problem involves identifying the longest subsequence of numbers where differences between consecutive elements alternate between positive and negative. This pattern resembles a zigzag, and understanding the problem requires focusing on turning points in the sequence. By observing the array, any increase followed by a decrease, or vice versa, represents a valid "wiggle." This observation drives the approach to solve the problem efficiently.

Approach

The solution uses a greedy algorithm to track changes in the direction of differences. Two variables, up and down, are initialized to keep track of the length of the longest wiggle subsequence that ends with an increasing or decreasing difference, respectively. The idea is to traverse the array and update these values whenever the sequence "wiggles." If the current element is larger than the previous one, it forms an upward wiggle and up is updated. Similarly, a smaller element updates down.

Edge Cases

Special cases arise when the input sequence has only one element or when all elements are equal. In both situations, the entire sequence is trivially a wiggle sequence, and the output is the length of the sequence, which is 1. Handling such cases ensures the solution is robust and accounts for all possible inputs within the constraints.

Algorithm

The algorithm begins by initializing up and down to 1, reflecting that a single element is a valid wiggle. The array is then traversed from the second element onward. *For each element, the algorithm compares it with the previous one:*

- If it is larger, it signifies an upward wiggle and up is incremented based on the value of down.
- If it is smaller, it indicates a downward wiggle and down is incremented using up. The final result is the maximum of up and down, representing the longest wiggle subsequence.

Complexity Analysis

The time complexity of the algorithm is O(n), where n is the size of the array. This efficiency is achieved by iterating through the array only once, comparing consecutive elements. The space complexity is O(1) since the solution uses only a constant amount of extra memory, namely the variables up, down, and a few temporary variables during the loop.

Real-World Applications

The problem has applications in analyzing data patterns, such as stock market trends or signal processing, where identifying oscillations is important. Understanding the behavior of sequences and finding efficient solutions to pattern-related problems is crucial in such fields. The greedy algorithm approach ensures the solution remains optimal and efficient, making it suitable for real-time data analysis.

Conclusion

This problem highlights the power of greedy algorithms in solving pattern-based challenges. By focusing on local changes (turning points), the solution avoids the need for complex dynamic programming or exhaustive search. The elegant combination of intuition and efficient traversal makes this approach effective and easy to implement and understand.