Documentation: Minimum Path Sum in a Triangle

Problem Description

Given a triangle array, the task is to find the minimum path sum from the top to the bottom. For each step, you may move to an adjacent number in the row directly below. More formally, if you are on index i on the current row, you may move to either index i or index i + 1 on the next row.

Example 1:

Input: triangle = [[2],[3,4],[6,5,7],[4,1,8,3]]

Output: 11

Explanation: *The triangle looks like:*

```
   2
  3 4
 6 5 7
4 1 8 3
```

The minimum path sum from top to bottom is 2 + 3 + 5 + 1 = 11.

Example 2:

Input: triangle = [[-10]]

Output: -10

Constraints

- 1 <= triangle.length <= 200
- triangle[0].length == 1
- triangle[i].length == triangle[i 1].length + 1
- -10^4 <= triangle[i][j] <= 10^4

Approach:

1. Iterate from the bottom to the top: Start from the second last row of the triangle and move upwards to the top.
2. Update each element: For each element in the current row, update its value to the minimum path sum by adding the minimum of the adjacent numbers from the row directly below.
3. Final result: The top element of the triangle will contain the minimum path sum from top to bottom.

Time Complexity:

- The solution iterates through each element of the triangle exactly once, resulting in a time complexity of O(n), where n is the total number of elements in the triangle.

Space Complexity:

- The solution uses O(1) extra space (in-place modification of the input triangle array). For the follow-up question requiring only O(n) extra space, dynamic programming with a 1D array could be used, but this solution modifies the input triangle directly.

Example Usage:

- The given solution can be tested with the following inputs:
- triangle = [[2], [3, 4], [6, 5, 7], [4, 1, 8, 3]] should return 11.
- triangle = [[-10]] should return -10.