Trapping Rain Water Algorithm Documentation (Generated by ChatGPT)

Problem Statement

Given an array `height` representing an elevation map where the width of each bar is 1, compute how much water can be trapped after raining.

Approach

This solution uses the **two-pointer approach** to efficiently calculate the trapped water by iterating through the elevation map from both ends.

Implementation Details

Function Signature

```
from typing import List
class Solution:
    def trap(self, height: List[int]) -> int:
```

Parameters

- `height (List[int])`: A list of non-negative integers representing the height of bars in the elevation map.

Return Value

- `int`: The total amount of water trapped between the bars.

Algorithm Explanation

1. **Edge Case Handling**: If the `height` list is empty, return `0` as no water can be trapped.
2. **Initialize Pointers**:
    - `left` and `right` pointers at the beginning and end of the list.
    - `left_max` and `right_max` to store the highest bars encountered from the left and right sides.

    - `res` to accumulate the trapped water.
3. **Two-Pointer Traversal**:
    - If `height[left] < height[right]`, process the left side:
        - Update `left_max`.
        - Calculate trapped water at `left` and add to `res`.
        - Move `left` pointer rightward.
    - Otherwise, process the right side:
        - Update `right_max`.
        - Calculate trapped water at `right` and add to `res`.
        - Move `right` pointer leftward.
4. **Termination**: The loop runs until `left` and `right` pointers meet.
5. **Return `res`**: The total trapped water.

Complexity Analysis

- **Time Complexity**: `O(n)`, as we traverse the height array once.
- **Space Complexity**: `O(1)`, as only a few extra variables are used.

Example Walkthrough

Example 1

Input:

```
height = [0,1,0,2,1,0,1,3,2,1,2,1]
```

Visualization:

```
Elevation Map:        _
                    _| |
    _   | |   | |  | |
  _| |_ | |_| | |__| |
```

Water Trapped: `6`

Output:

```
Solution().trap([0,1,0,2,1,0,1,3,2,1,2,1])  # Returns 6
```

Example 2

Input:

```
height = [4,2,0,3,2,5]
```

Output:

```
Solution().trap([4,2,0,3,2,5])  # Returns 9
```

Example 3

Input:

```
height = [4,2,3]
```

Output:

```
Solution().trap([4,2,3])  # Returns 1
```

Summary

- Uses a **two-pointer technique** to optimize water trapping calculations.
- Maintains a **left_max** and **right_max** to determine trapped water at each step.
- Runs efficiently in **O(n) time complexity** with **O(1) space complexity**.

This approach ensures an optimal and scalable solution for the **Trapping Rain Water** problem.