Title: Trapping Rain Water

Overview:

This documentation provides an explanation and implementation details for solving the "Trapping Rain Water" problem. The problem requires computing the amount of water trapped after raining given an elevation map represented by an array of non-negative integers.

Problem Statement:

Given an array of non-negative integers representing an elevation map where the width of each bar is 1, the task is to compute how much water can be trapped after raining.

Example:

Input: height = [0,1,0,2,1,0,1,3,2,1,2,1]

Output: 6

Explanation: The elevation map is represented by the array [0,1,0,2,1,0,1,3,2,1,2,1]. In this case, 6 units of rainwater are being trapped.

Explanation:

- The `trap` function takes an array `height` as input and returns the amount of trapped water.
- It initializes two pointers `left` and `right` at the beginning and end of the array respectively.
- `left_max` and `right_max` are initialized to the heights of the first and last bars.
- The function iterates through the array while `left` is less than `right`.
- If the height at `left` is less than the height at `right`, it checks whether the current height at `left` is greater than or equal to `left_max`. If it is, `left_max` is updated; otherwise, the trapped water is incremented by the difference between `left_max` and the current height at `left`, and `left` is incremented.
- If the height at `right` is greater than the height at `left`, it checks whether the current height at `right` is greater than or equal to `right_max`. If it is, `right_max` is updated; otherwise, the trapped water is incremented by the difference between `right_max` and the current height at `right`, and `right` is decremented.
- Finally, the total trapped water is returned.

Complexity Analysis:

- **Time complexity:** O(n) - where n is the number of elements in the input array.
- **Space complexity:** O(1) - constant space is used.