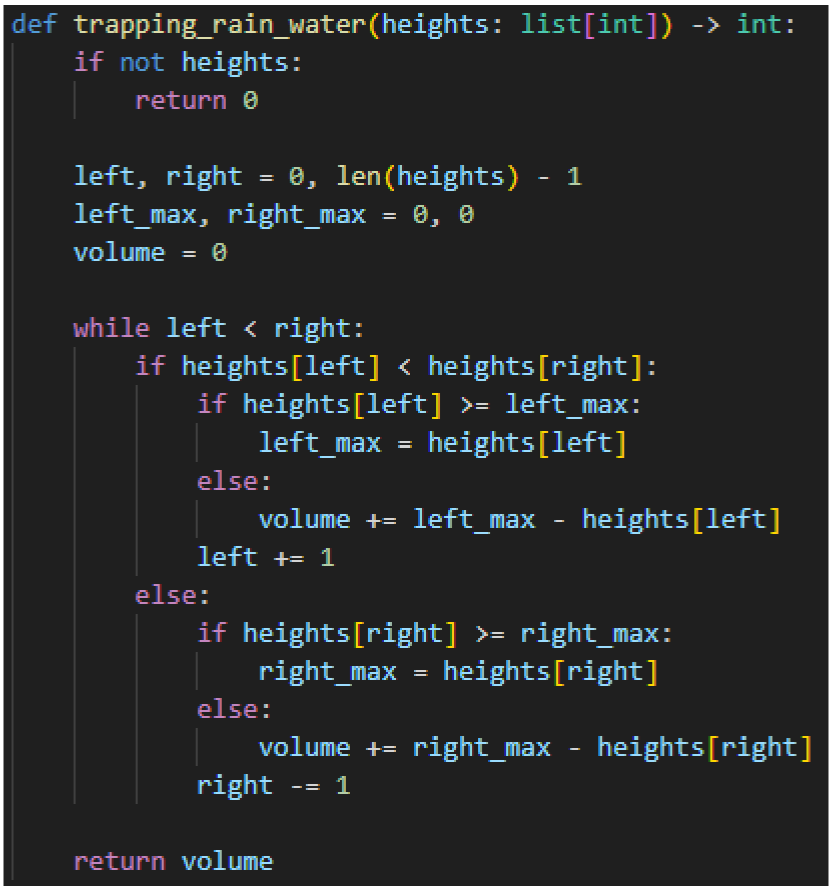
Problem Statement:

Given an integer array, heights, where heights[i] represents the height of the ith bar, determine the maximum amount of water that can be contained between these bars. The water will spill over the shorter of two bars. The solution must efficiently compute this maximum volume.

Solution 1



```python
def trapping_rain_water(heights: list[int]) -> int:
    if not heights:
        return 0

    left, right = 0, len(heights) - 1
    left_max, right_max = 0, 0
    volume = 0

    while left < right:
        if heights[left] < heights[right]:
            if heights[left] >= left_max:
                left_max = heights[left]
            else:
                volume += left_max - heights[left]
            left += 1
        else:
            if heights[right] >= right_max:
                right_max = heights[right]
            else:
                volume += right_max - heights[right]
            right -= 1

    return volume
```

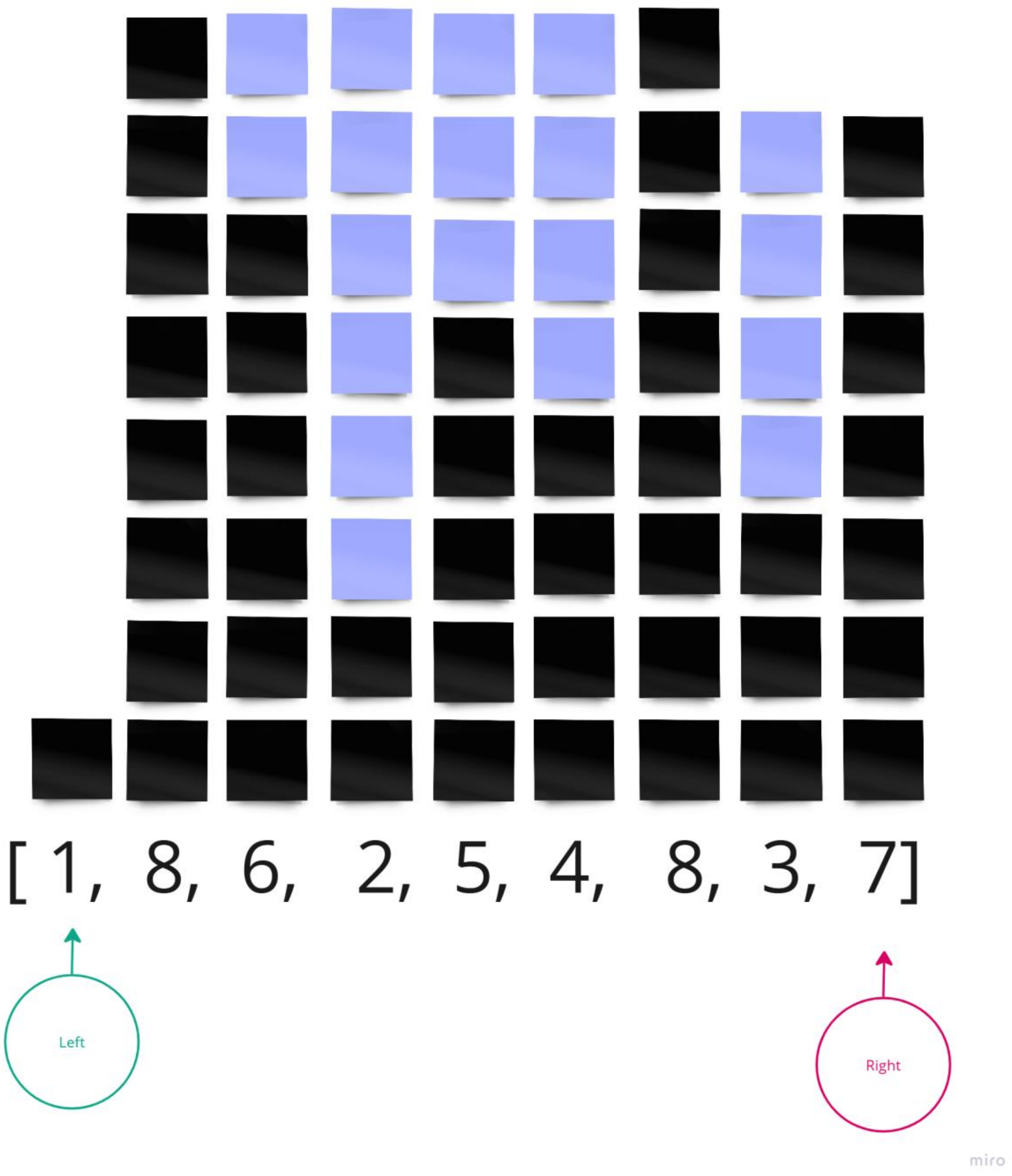
Step-by-Step Breakdown

1. **Input:**
    - heights: list of integers representing the heights of the bars [1,8,6,2,5,4,8,3,7].
2. **Intermittent step 1:**

   Edge Case Check:
    - If heights is empty, return 0 since no container can be formed.
    - python
3. **Intermittent step 2:**

Initialize Pointers and Variables:

- Set left to 0 (the start of the list) and right to len(heights) - 1 (the end of the list).
- Initialize two variables left_max and right_max to track the maximum height seen so far from the left and right, respectively.
- Initialize volume to 0 to accumulate the total trapped water.



Iterative Two-Pointer Process:

While left < right, compare heights[left] and heights[right]:

- If heights[left] < heights[right]:
- If heights[left] is greater than or equal to left_max, update left_max with heights[left].

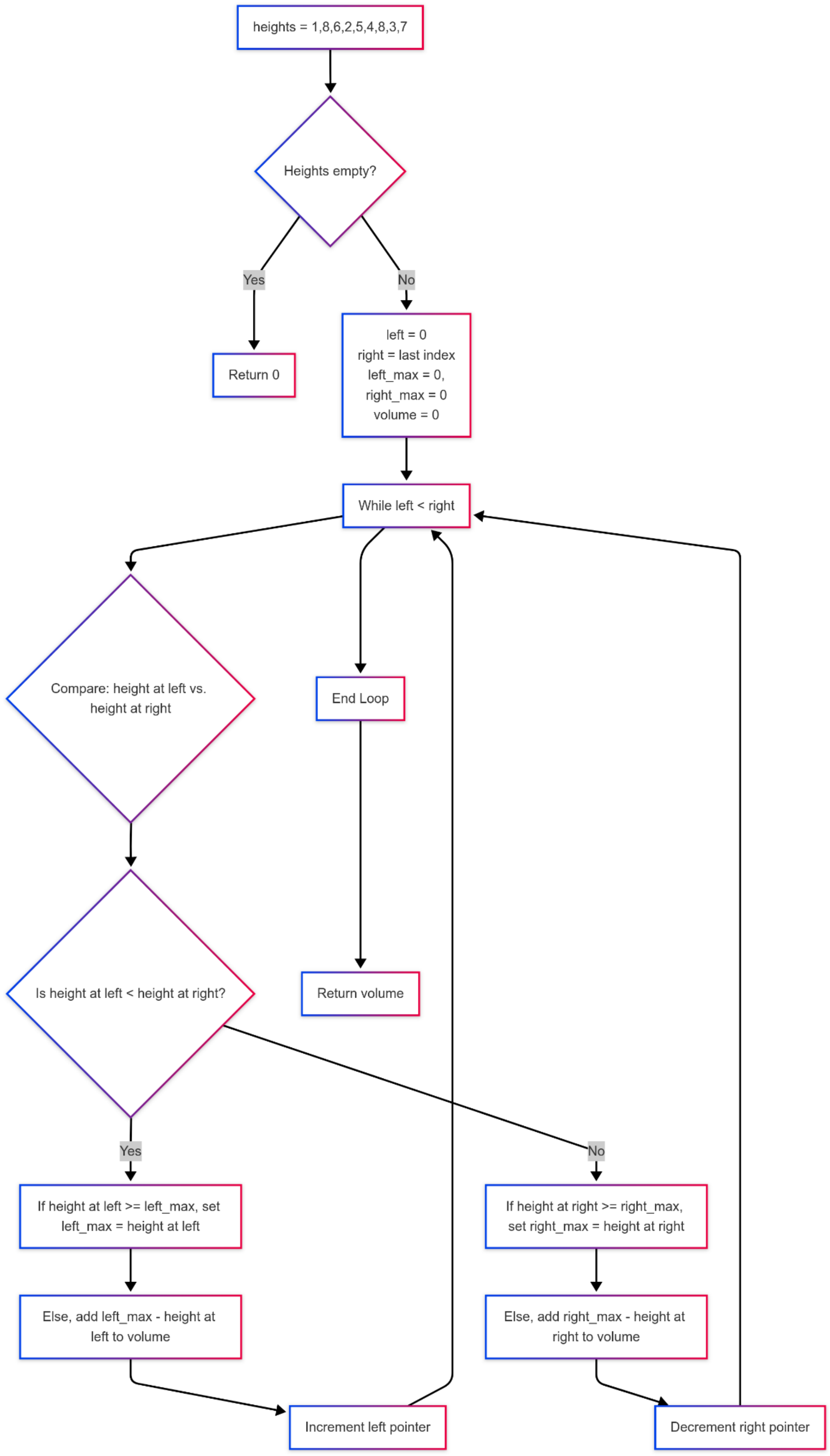
    - Otherwise, the current bar can hold water: add left_max - heights[left] to volume.
    - Increment the left pointer to examine the next bar.

   Else (heights[left] >= heights[right]):

    - If heights[right] is greater than or equal to right_max, update right_max with heights[right].
    - Otherwise, add right_max - heights[right] to volume.
    - Decrement the right pointer to examine the previous bar.

4. **Output:**
    - Return the computed volume which represents the maximum amount of water that can be trapped between the bars.
5. **Efficiency:**
    - Time Complexity: O(n) – The two-pointer approach traverses the array only once.
    - Space Complexity: O(1) – Only a fixed number of variables are used, irrespective of the input size.

Visual Flow Diagram


heights = 1,8,6,2,5,4,8,3,7
Heights empty?
Yes
No
Return 0
left = 0
right = last index
left_max = 0,
right_max = 0
volume = 0
While left < right
Compare: height at left vs. height at right
End Loop
Return volume
Is height at left < height at right?
Yes
No
If height at left >= left_max, set left_max = height at left
Else, add left_max - height at left to volume
If height at right >= right_max, set right_max = height at right
Else, add right_max - height at right to volume
Increment left pointer
Decrement right pointer