42. Trapping Rainwater

<https://leetcode.com/problems/trapping-rain-water/>

1. **Listen**

**Problem Statement:**

Given n non-negative integers representing an elevation map where the width of each bar is 1, compute how much water it can trap after raining.

**Input:**

int[] height: **n** non-negative integers representing an elevation map where the width of each bar is 1.

height[i] is a non-negative integer representing a height and a width of 1.

**Goal:**

Compute how much water the elevation map can trap after raining.

**Return:**

Return the number of 1x1 squares we can trap after it rains.

1. **Examples**

**Example 1:**

A picture containing text, clipart

Description automatically generated

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

**Output:** 6

**Explanation:** The above elevation map (black section) is represented by array [0,1,0,2,1,0,1,3,2,1,2,1]. In this case, 6 units of rain water (blue section) are being trapped.

**Constraints:**

* n == height.length
* 1 <= n <= 2 \* 104
* 0 <= height[i] <= 105

**Test Cases:**

* No rainwater is able to be trapped.

1. **Brute Force**

**Solution 1: ------------------------------------------------------------------------------------------------------**

**Time = O(N)**

**Space = O(N)**

**Intuition**

* Let’s take a look at an example.

A picture containing text, clipart

Description automatically generated



* Let’s say we want to be able to trap water at say index 4.
* Now, how would we go about doing this?
* First of all, when doing this algorithm, we want to take it one index at a time.
* Perhaps you’re saying, maybe there’s some way to calculate it bottom-up? Well, there’s no easy way to do that
* The best way would be to calculate the area of each individual cell one by one.

**Find the Maximum Height in the Left and Right subarrays relative to index i.**

* So, just by looking at the graph, how would you get the individual area of index 4?
* We would obviously want to fill up as much area in the current index as we can. Therefore, we need to find the Maximum height from the left and right sides of the array relative to i.
* So, for index 4, we would look to the left and right.
  + What’s the tallest elevation to the left? 2.
  + What’s the tallest elevation to the right? 3.

**Get the Minimum between the left max height and the right max height.**

* Now, we can’t fill rain up to height 3 at index 4, because then the water would just overflow to the left over the height 2 elevation. So, we take the minimum between the two heights.
* But, we know that the elevation of the larger height between the two heights is at least as tall as the minimum height. I.e., we know that height 3 is at least as tall as height 2. So we can comfortably fill up our area to height 2 without overflowing.
* In pseudocode, it looks like this.

int leftMaxHeight = max height from left-side of the array up to i.

int rightMaxHeight = max height from i to the end of the right-side left side of the array.

int minHeight = Math.min(leftMaxHeight, rightMaxHeight).

**Subtract elevation at index i from minHeight**

* Now, we also have to account for the elevation of the current index. Index 4 has an elevation of 1. If it didn’t have an elevation, like index 5, we could simply just say the area is the minimum between the left and right heights. But since there is an elevation here, we need to subtract it from the minimum between the left and right heights.
* In pseudocode, it looks like this.

int currArea = minHeight – height[i];

if (currArea > 0) totalArea++;

* Now, there are times where we won’t be able to fill in an area, like index 3.
* Index 3 has an elevation of 2. The max left height is 1, and the max right height is 3.
* After taking the minimum between the two, we get 1.
* After subtracting the current elevation, which is 2, from 1, we can see that we would end up with a negative area.
* If we encounter any area like this, we simply ignore it, because we aren’t able to fill in any rain water.

**Algorithm**

Text

Description automatically generated with low confidence

* For every single, position, in order to know how much water we can trap at index i, we need to know, what’s the max left and right height of every single position?
* We can do two preemptive passes over the heights array and store these values into left and right height arrays in linear time and space.
  + We make one pass going left to right finding the maxLeft height for each index.
  + We make one pass going right to left finding the maxRight height for each index.

Calendar

Description automatically generated with low confidence

Now remember, we need to calculate the minimum between the left and right heights for each index in order to determine how much water we can trap at each position.

We can iterate over both arrays and fill in a minimum height array in linear space and time.

A picture containing text

Description automatically generated

Remember, we can find the amount of rainwater a position can trap if we know the minimum between the max left and right heights and subtract this from the current position’s elevation.

Now that we know the minimum between the left and right heights for each position, we can simply iterate over the heights array and subtract the min(L, R) value from the current index value.

We can keep track of and return the total area we can fill.

1. **Optimize**

**Solution 2: ------------------------------------------------------------------------------------------------------**

**Time = O(N)**

**Space = O(1)**

Intuition

Out intuition follows the same suit as the first solution. However, we can optimize the space complexity by using the two pointer technique.

1. **Walkthrough**
2. **Implement**
3. **Test**