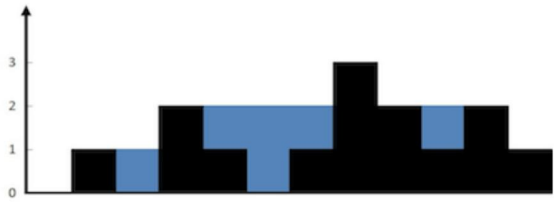
42. Trapping Rain Water

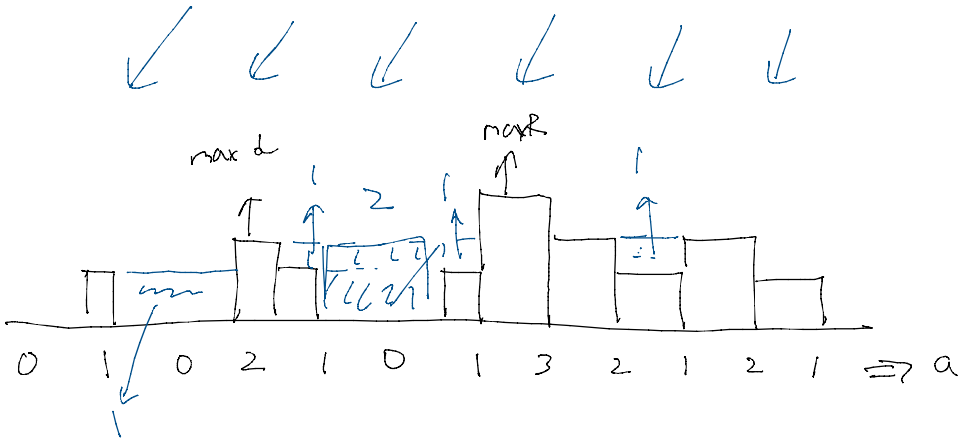
06 April 2022 08:36 AM

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.

Example 1:




```
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.
```



1 + 1 + 2 + 1 + 1 = 6

Brute force :

→ for every index find out the unit of water stored

at index 2 ⇒ 1 amount of water stored

" " 4 ⇒ 1 " " " "

min (left (i), right (i)) − a[i]

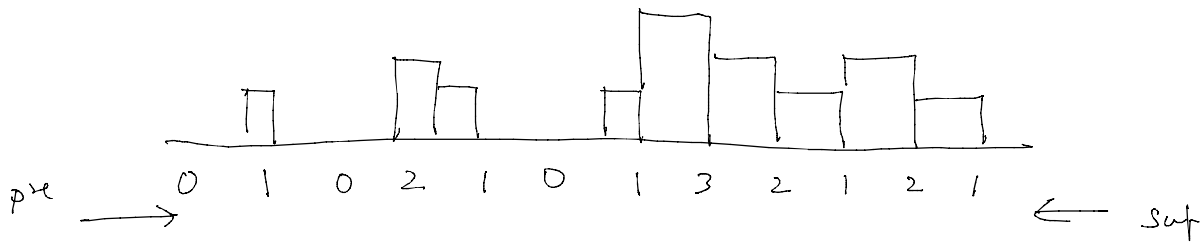
$$\min\left(\text{left}(i), \text{right}(i)\right) - a[i]$$

at 3rd index $\min(2, 3) - 2$

$\Rightarrow \quad 2 - 2 = 0$

T.C $\Rightarrow O(n^2)$ S.C $\Rightarrow O(1)$

2) Better Solution : Prefix Sum and Suffix May



				L								
pre	0	1	1	2	2	2	2	3	3	3	3	3
				R								
suf	3	3	3	3	3	3	3	3	2	2	2	1
	0	1	2	3	4	5	6	7	8	9	10	11

from back you start

$$\min\left(\text{left}(i), \text{right}(i)\right| - a[i]$$

Easily you get left max and right max from pre and suf

for 3rd index

$\min(2, 3) - 2$

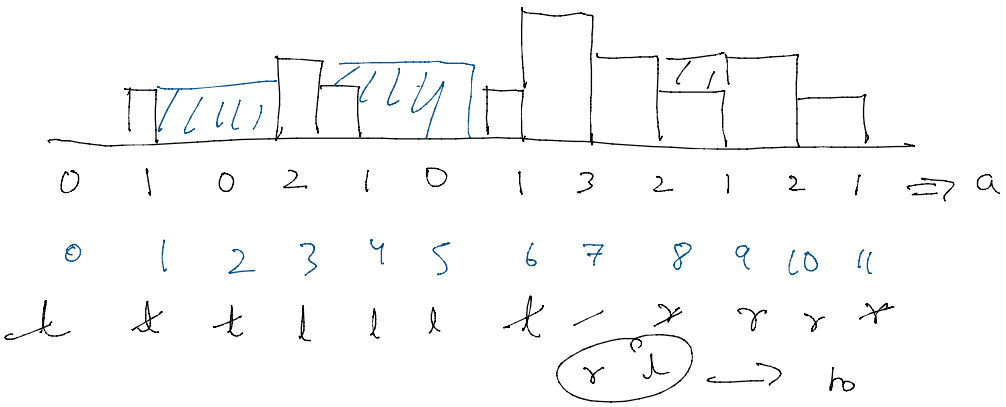
$= 0$

T.C $\Rightarrow O(n) + O(n) + O(n)$

$\Rightarrow O(3n) \simeq O(n)$

S.C $\Rightarrow O(2n)$

(3) Optimal Solution : Two pointer approach



$l = 0$

$r = n - 1$

total_water = 0 1 2 4 5

left max = 0 1 2 3

right max = 0 1 2

⇒ ans

```
if (a[l] <= a[r])
    if (a[l] >= left_max)  left_max = a[i]
    else  total_water += (left_max - a[i])
    l++
else
    if (a[r] >= right_max)
        right_max = a[r]
    else  total_water += (right_max - a[i]
```

(a[] < a[l])

T.C ⇒ O(n)

S ⇒ O(1)

```
            // Increment left, we'll now look at the next point.
            left++;
        // If the height at the left is NOT greater than height at the right, we cannot fill from left to
            right without over-
        // flowing; however, we do know that we could potentially fill from right to left, if there is nothing
            in between.
        } else {
            // Similarly to above, we see that we've found a height greater than the max, and cannot fill it
                whatsoever, but
            // everything before is optimally filled
            if (height[right] >= maxRight) {
                // We can say we've found a new maximum and move on.
                maxRight = height[right];
            // If we haven't found a greater elevation, we can fill the current elevation with maxRight -
                height[right]
            // water.
            } else {
                totalWater += maxRight - height[right];
            }
            // Decrement left, we'll look at the next point.
            right--;
        }
    }
    // Return the sum we've been adding to.
    return totalWater;
}
```