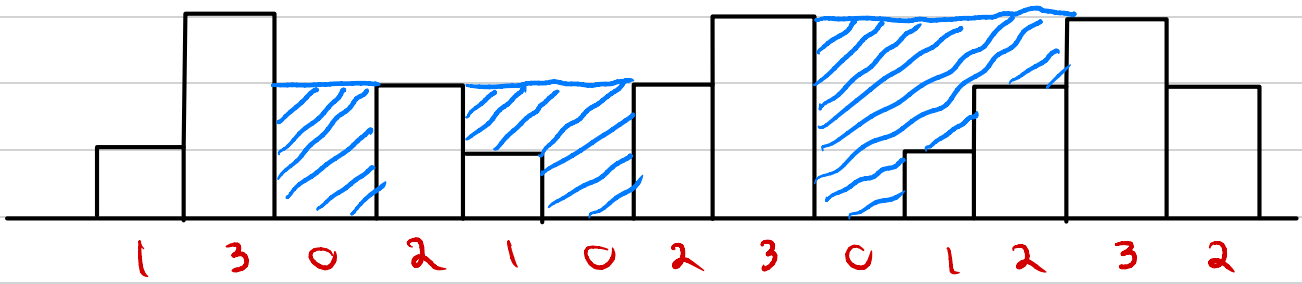
TRAPPING RAINWATER

We are given an array of integers where each integer represents the height of a building.

When it rains, water gets logged between buildings, our job is to calculate how much.



Water logged at i^{th} element can be given by

$$\min(leftMax, RightMax) - arr[i]$$

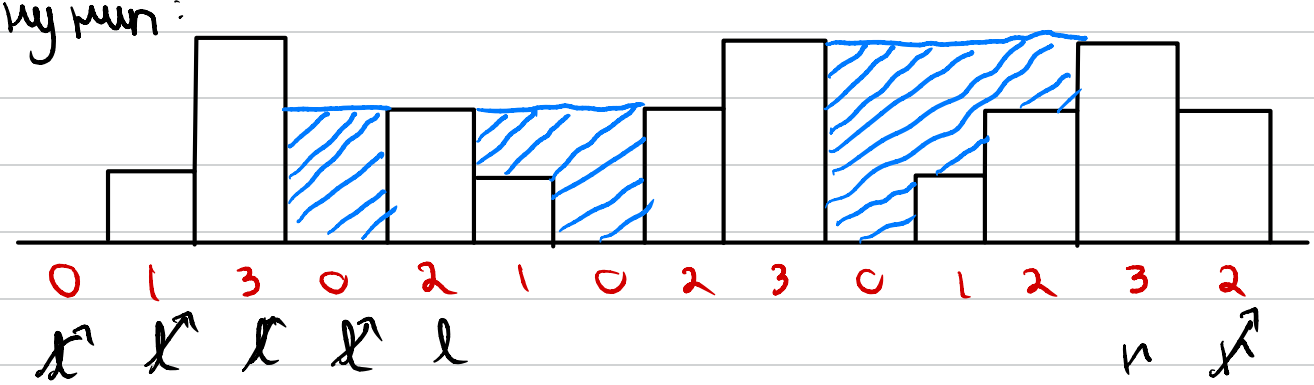
Largest Building to my left

Largest Element to my right

We can store leftMax and rightMax in array for each index using 2 arrays but that adds space complexity O(2N).

Optimal Solution does not involve use of both leftMax and Rightmax array.

Diagram:



lMax = 0, rMax = 2, total = 2

l is 0, r is 2
∴ Water can log
But lMax is 0, so it will flow back to streets.
So l shifts
l is 1, r is 2
∴ Water can log
But lMax is 0, so it will flow back to streets.
So l shifts and lMax changes to 1
l is 3, r is 2
∴ Water can log
But rMax is 0, so it will flow back to streets.
So l shifts and lMax changes to 1
l is 3, r is 2. Water can log but rMax is 0 so it will just flow. Hence r shifts and rMax changes to 2
Both l and r are 3. Move either. l shifts and lMax changes to 3.

l is 0, r is 3. Water will store as per rMax since its lesser. Total is now 2. l shifts again to 2. r is 3, hence water can log with rMax as it is lesser. 2 - 2 = 0 is added to total.

So On

Basically Algorithm is :

- lMax is 0, rMax is 0, total is 0, l is arr[0], r is arr[N-1]
- Whenever gap exists w/w l and r, water can log (only if lMax and rMax are not 0). Minimum of both of them is used.
- If l and r are equal, either of them can move. Otherwise, the smaller of them moves

Pseudocode :

```
trappingRainwater (arr, N) {
    int l = 0, r = N-1, total = 0, lMax = 0
        rMax = 0;
    while (l <= r) {
        if (arr[l] <= arr[r]) {
            if (lMax > arr[l]) {
                total += lMax - arr[l];
            } else {
                lMax = arr[l];
            }
            l += 1;
        } else {
```

```
            if (rMax > arr[r]) {
                total += rMax - arr[r];
            } else {
                rMax = arr[r];
            }
            r -= 1;
        }
    }
    return total;
}
```