42. Trapping Rain Water

Problem Statement

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

The above elevation map 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.

Thanks Marcos for contributing this image!

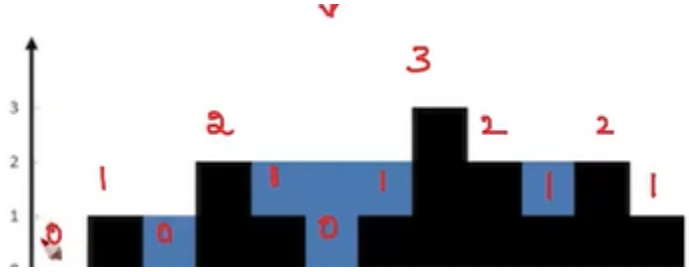
Example:

```
Input: [0,1,0,2,1,0,1,3,2,1,2,1]
Output: 6
```

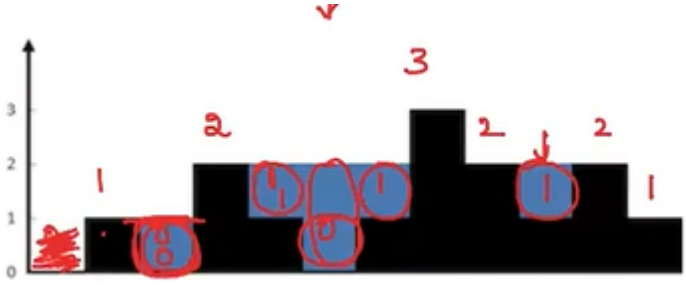
2. Lets understand this with an example

```
Input: [0,1,0,2,1,0,1,3,2,1,2,1]
Output: 6
```

3. So here if we represent our input array we get



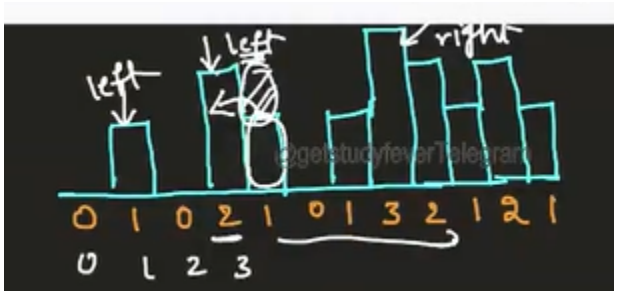
4. Now all the blue colored blocks store the rain water…if we calculate the amount of water it stored …then we get 6units of water =



Approach 1:

1. Here what we are doing is for every element…we calculate leftMaxHeight and rightMaxHeight .
2. Now we'll take min(leftMaxHieight, rightMaxHeight)..it gives the value ..which can be used to get diff between the min() and current element
3. Code:
4.
```
n = len(height)
result = 0
for i in range(1, n-1):
    #find max element on left side
    left = height[i]
    for j in range(i):
        left = max(left, height[j])

    #find max element on right side
    right = height[i]
    for j in range(i+1, n):
        right = max(right, height[j])
    result = result + min(left, right) - height[i]
return result
```
T.C ⇒ O(n²)

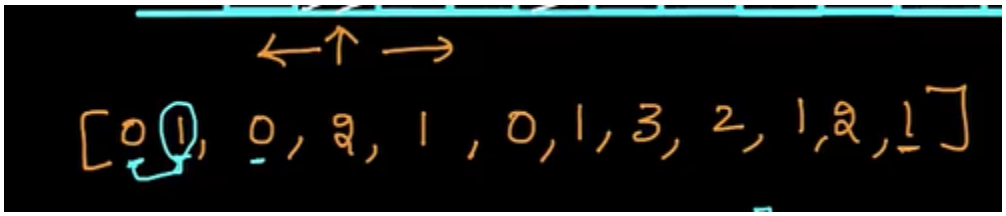
lets dry run this code
5. 

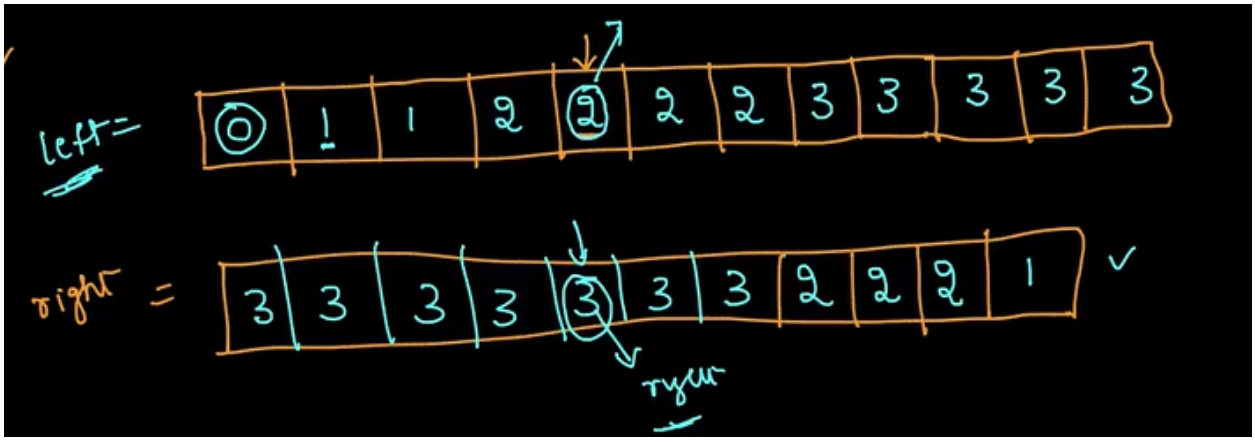
6. Consider our example…we'll start from index 1
7. For index = 1 ….left = 1….and right = 3
```
left = height[i]
for j in range(i):
    left = max(left, height[j])

#find max element on right side
right = height[i]
for j in range(i+1, n):
    right = max(right, height[j])
```
8. Now the result would be = 0+min(1,3)-1 = 0
9. For index = 2 ….left = 1 and right = 3….now the result would be = 0+min(1,3)-0 ⇒ 1
10. For index = 3(2) ….left = 2…and right = 3….now the result = 1+min(2,3)-2 = 1
11. Similarly for index 4 ..result = 1 + min(2,3)-1 ⇒ 2
12. Index 5..result = 2+min(2,3)-0 ⇒ 4
13. SImilarly if we calculate for all the indexes…then we'd get result = 6
14. The time complexity of this approach is O(n^2) and in real time we'd get TLE error

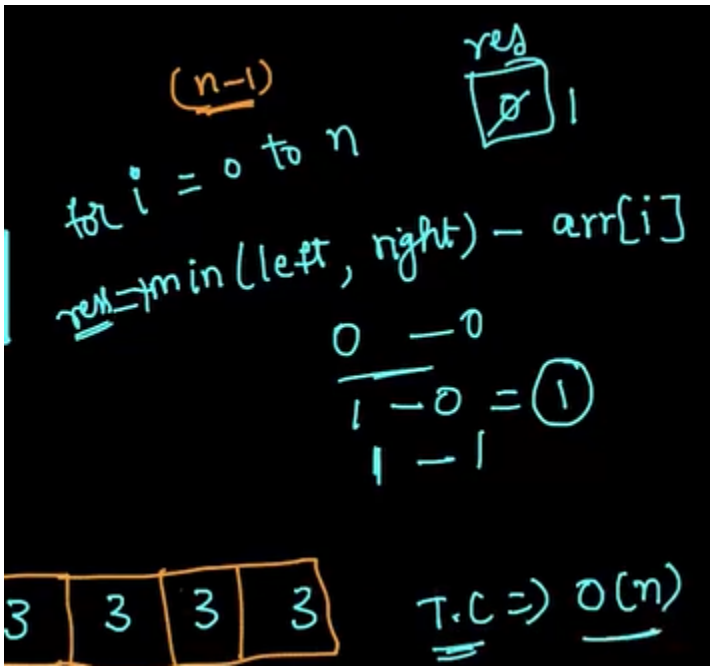
2nd Approach

1. Here we will take 2 extra arrays
2. Given array





3. In the array left…we'll fill the values with maxUntilFound…see the array and understand
4. Similarly for array right..we'll iterate the array from back..and fill the value with maxUntilFound
5. It gives leftMaxHeight and rightMaxHeight for the given element…then we can iterate these both arrays at once…and can compute the result



6. This computing of left and right arrays is a concept of DP (memoization)

7. code:

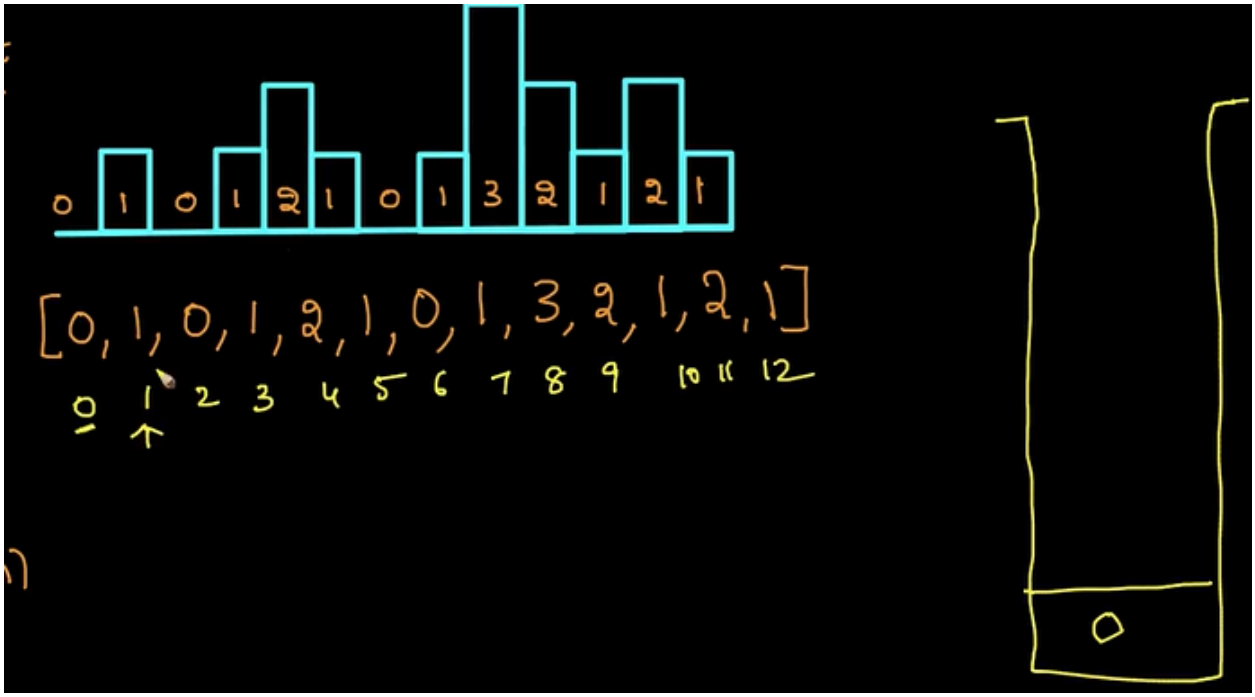
```
class Solution(object):
    def trap(self, height):
        """
        :type height: List[int]
        :rtype: int
        """
        n = len(height)
        left = [0]*n
        right = [0]*n
        result = 0
        left[0] = height[0]
        for i in range(1, n):
            left[i] = max(left[i-1], height[i])
        right[n-1] = height[n-1]
        for i in range(n-2, -1, -1):
            right[i] = max(right[i+1], height[i])
        for i in range(n):
            result += min(left[i], right[i])-height[i]
        return result
```

Approach

3rd Approach

1. In this approach we'll be using a stack

2. First we'll push index 0 to the stack…



3. Next we'll move to 1…and we compare it with the top of the stack