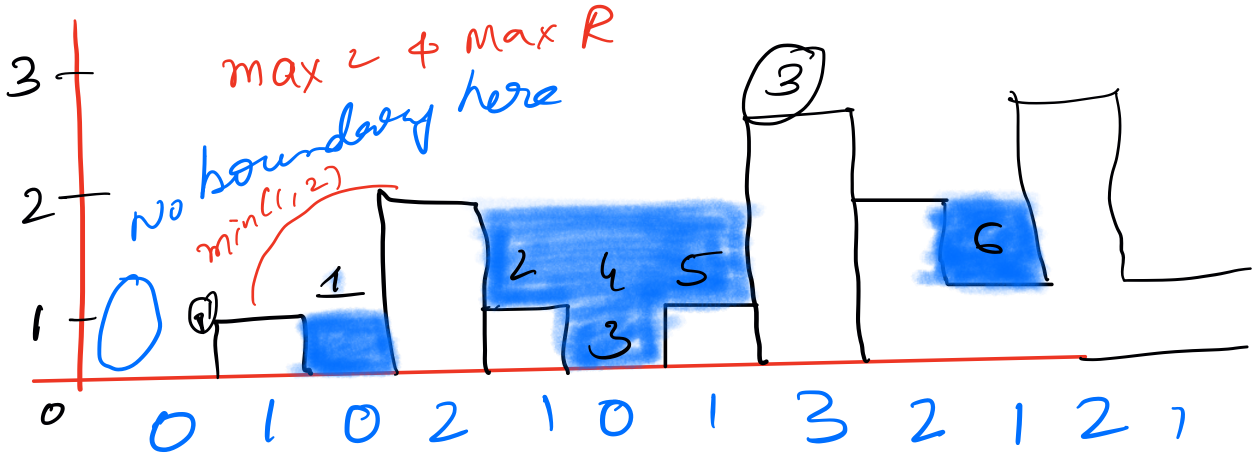
maxarea = max(maxarea
, area)

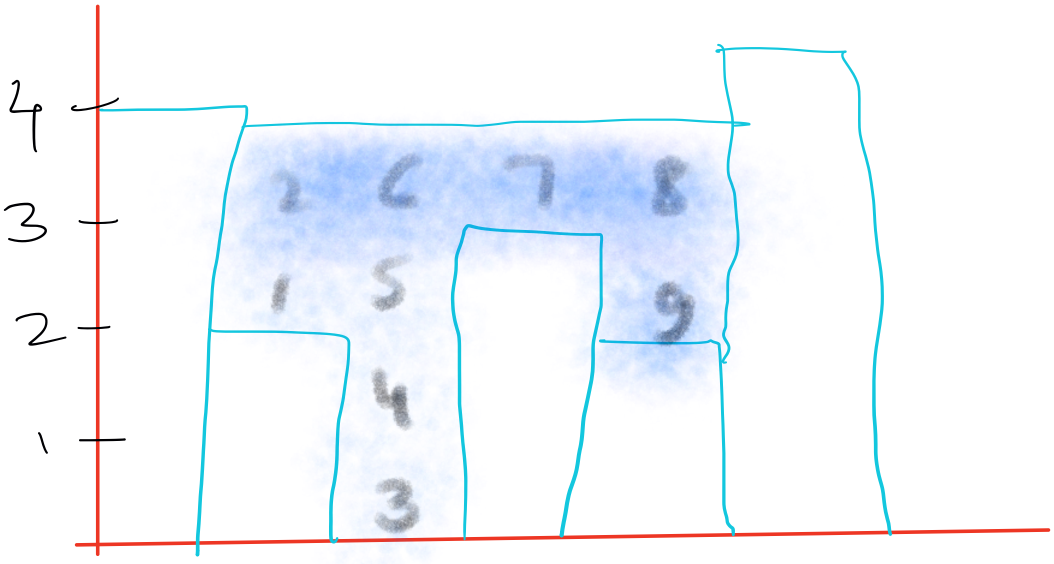
return maxarea.

42 :- Trapping of rain water.

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



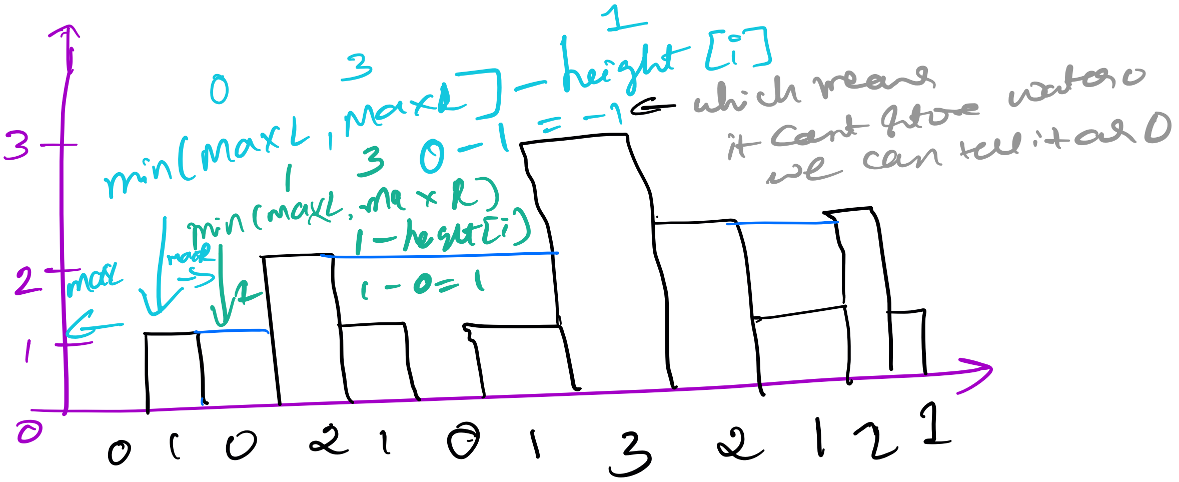
Input = height = [0,1,0,2,1,0,1,3,2,1,2,1]



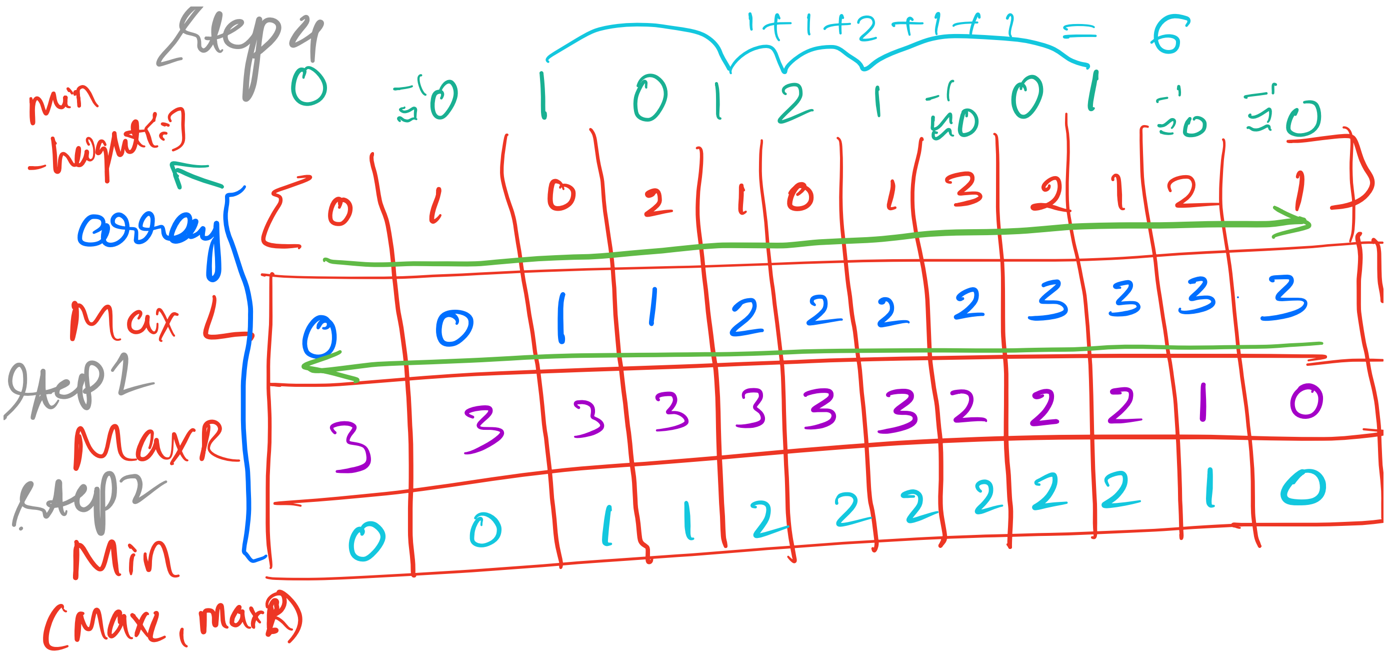
Input: [4,2,0,3,2,5]

output= 9

Explanation: [0,1,0,2,1,0,1,3,2,1,2,1]



(step 5)



min-height[i]	0	-1 ≤0	1	0	1	2	1	-1 ≤0	0	1	-1 ≤0	-1 ≤0
array	[0	1	0	2	1	0	1	3	2	1	2	1]
Max L	0	0	1	1	2	2	2	2	3	3	3	3
MaxR	3	3	3	3	3	3	3	2	2	2	1	0
Min (MaxL, maxR)	0	0	1	1	2	2	2	2	2	2	1	0

So:→ from above method we need to determine the maximum left value of a numeber for Each index without considering that index. initial value will be 0. when i=0 while calculating for all value of i.

here the time complexity will be O(n) and space complexity is O(n) so we can optimize the code using two pointer solution and we can save space complexity.

here is solution: L, R = 0, len(s)-1

res = 0

max[L] ↓ max[R] ↓

min (height[L], [height[R])

↓

0 - heigh[i] = 0

L → L → L → L

[0, 1, 0, 2, 1, 0, 1, 3, 2, 1, 2, 1]

R ← R

min (maxL, maxR)

min (1, 1)

1 - ht[3] => 1-2

=> -1 ≯ 0

next iteration

maxL = 2 & maxR = 1

R -= 1 & maxR = 2

↓

min (max[L], max[R])

min (1, 1)

=> 0 - 1 => -1 ≯ 0

we can iterate

L or R

we can iterate L for next step.

L += 1 for next step.

& maxL = 1

min (maxL, maxR)

min (1, 1)

height[R]

1 - 0 => 1

res += 1

Since both maxL & maxR are Equal

we can iterate

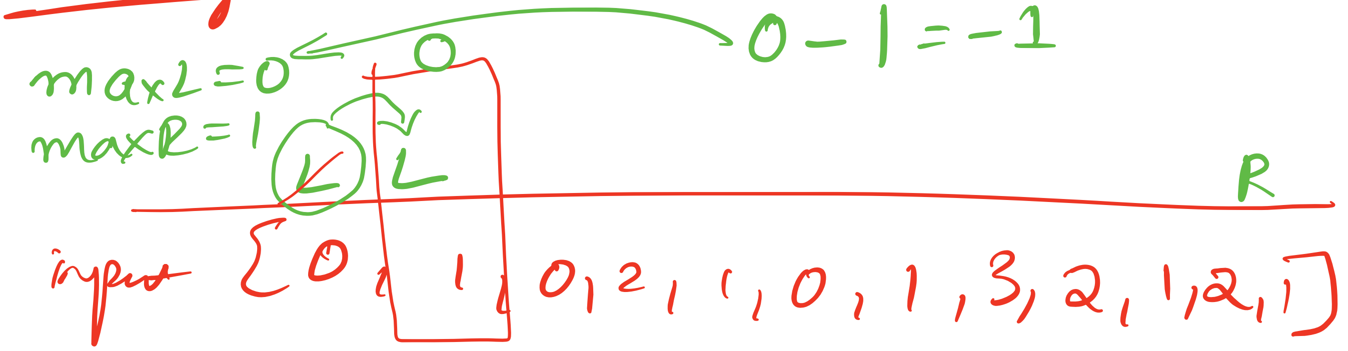
& height[i] < maxL

L++

& maxL = 1

Continued to Ed.

Same fig :

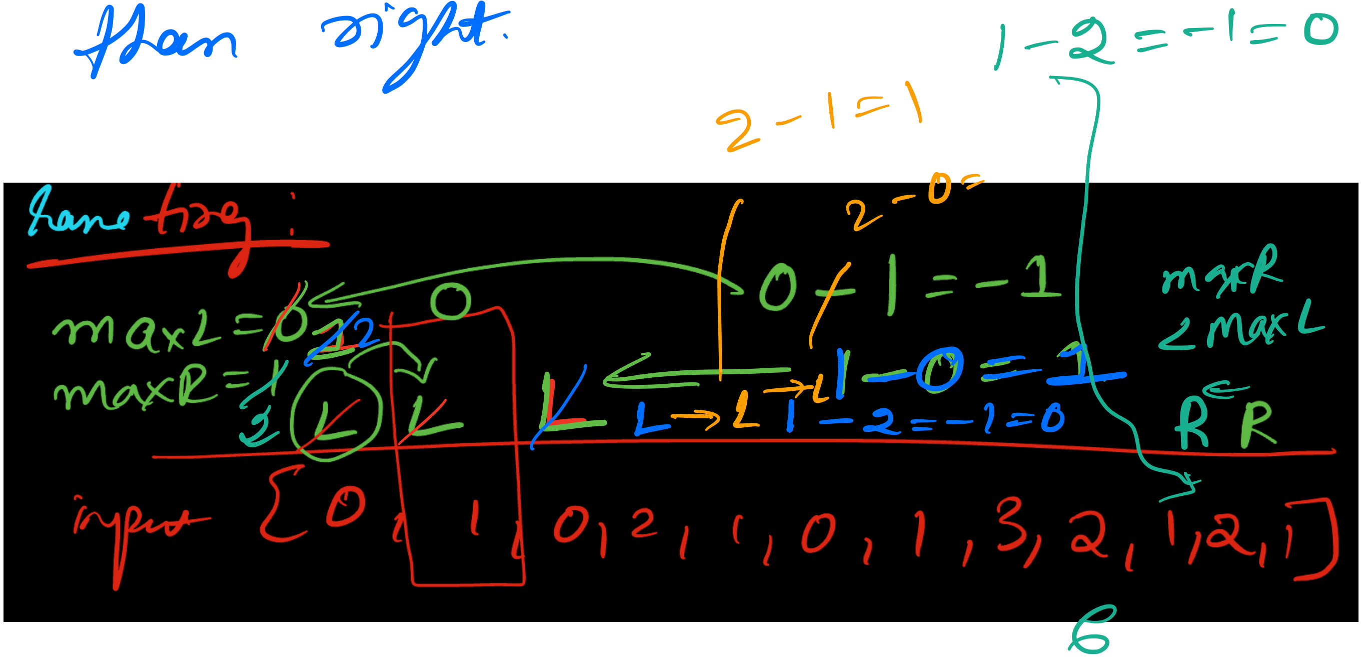


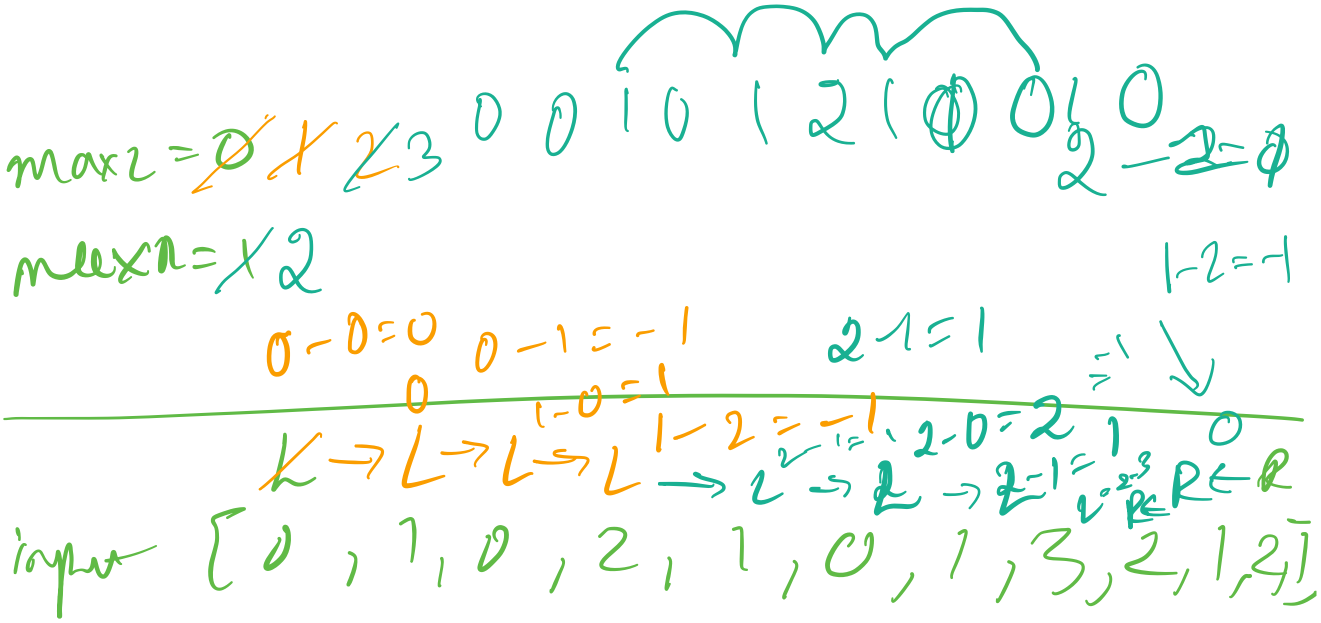
we want min of maxL, maxR.

we Exactly dont no what maxR is where 3 is a true max right. but why we dont need that value then.

remember we want minimum of max left & max Right value.

We already Know that the left is pretty small that is 0 we will shift left value to next position because left was smaller than right.




0 0 1 0 1 2 1 0 0 1 0
2 - 2 = 0
maxL = ~~0~~ ~~1~~ ~~2~~ 3
maxR = ~~1~~ 2
0-0=0
0-1=-1
0
1-0=1
2-1=1
1-2=-1
1-2=-1
2-0=2
2-1=1
0
L → L → L → L → L → 2 → 2 → 2 R ← R
input [0, 1, 0, 2, 1, 0, 1, 3, 2, 1, 2, 1]

1 (0, 1) => 1 1, 0

1 - 1 1 - 0 => 1

2

$2 - 2 = 0$

$1, 2 \Rightarrow 2$

$2 - 2 = 0$

```
class Solution(object):
    def trap(self, height):

        res = 0
        l, r = 0, len(height) - 1
        leftMax, rightMax = height[l], height[r]
        if not height:        # if the height is empty.
            return 0.
        while l < r:
            if (leftMax <= RightMax)
            l += 1
            leftMax = max(LeftMax,
                          height[l])
            res += leftmax
                   - height[l]
        else:
            r -= 1
            rightMax = max(rightMax,
                           height[r])
            res += rightMax
                   - height[r]
return res.
```