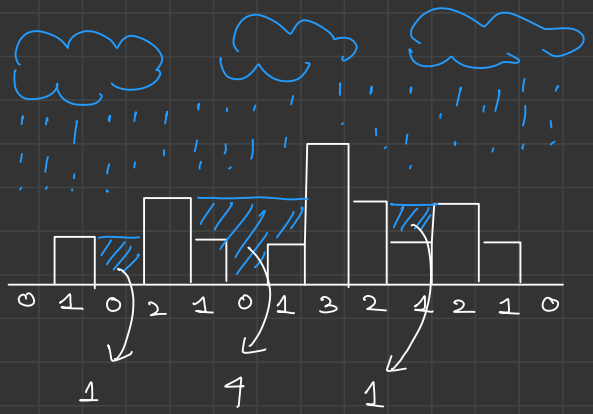
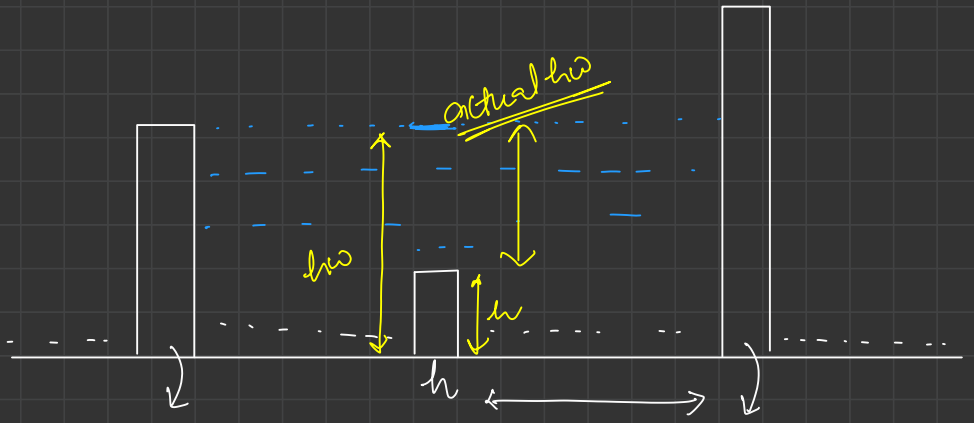
Trapping Rain Water



{ total water stored = 6 units }

How many unit of water a building can store above itself?

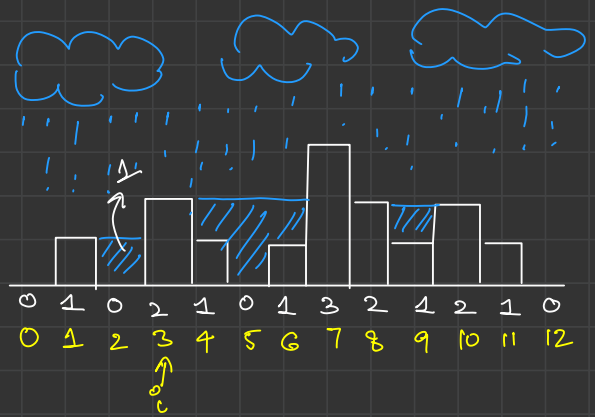


max^m height building on left side

$$hw = \min(LB, RB)$$

max height building on right side

$$\text{actual } hw = hw - h$$



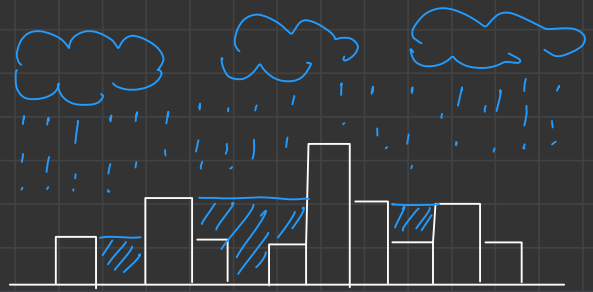
Tc : O(N²)
SC : O(1)

Total =

Brute force

i = 0	RB = 3	LB = −∞	how = −∞		
i = 1	RB = 3	LB = 0	how = 0		
i = 2	RB = 3	LB = 1	how = 1	h = 0	ahw = 1 water = 1
i = 3	RB = 3	LB = 1	how = 1	h = 2	

Can we do in O(N)?



LB = ~~XXXX~~3

RB = ~~XXXX~~3

lmax[] = -∞, 0, 1, 1, 2, 2, 2, 2, 3, 3, 3, 3, 3 ⟶ TC: O(N)

rmax[] = 3, 3, 3, 3, 3, 3, 3, 2, 2, 2, 1, 0, -∞ ⟶ TC: O(N)

```
for (int i=0; i<n; i++)
{
    low = Math.min(lmax[i], rmax[i]);
    if (low > height[i])
    {
        int addw = low - height[i];
        totalWater += addw*1;
    }
}
```

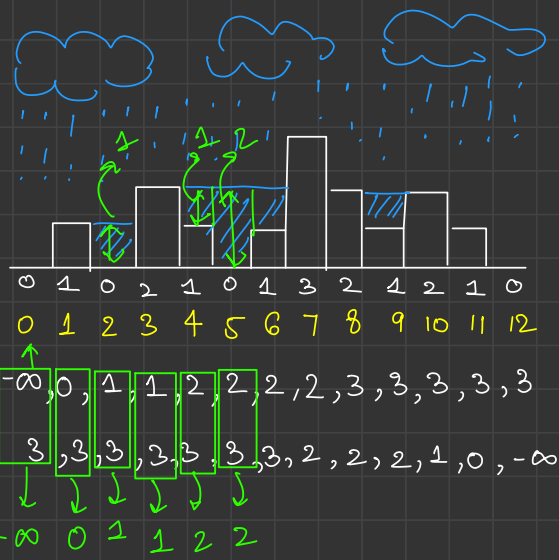
TC: O(N)
SC: O(N)

```java
// step 3: calc. water above each building
int totalWater = 0;
for (int i = 0; i < n; i++) {
    int hw = Math.min(lmax[i], rmax[i]);
    if (arr[i] < hw) {
        int ahw = hw - arr[i];
        totalWater += (ahw * 1);
    }
}

System.out.println(totalWater);
```

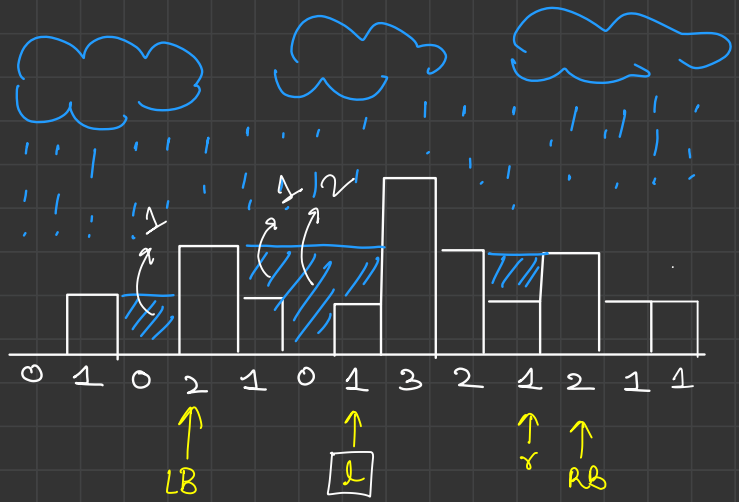
2 - 0 = 2

width



Tc : O(N)
SC : O(N)

TC : O(N) SC : O(1)



LB <= RB

```
hw = LB;
if (hw > arr[l])
{ totalWater += (hw - arr[l]) * 1;
}
LB = Math.max(LB, arr[l]);
l++;
```

RB < LB

```
hw = RB;
if (hw > arr[r])
{ totalWater += (hw - arr[r]) * 1;
}
RB = Math.max(RB, arr[R]);
r--;
```

Sum of Subarray Minimums

Brute force : TC : $O(N^2)$, SC : $O(1)$

arr[] = { 3, 2, 4, 1, 5, 2 }

Total Subarray : $\frac{N*(N+1)}{2}$

Subarrays

(3)[3] (3,2)[2] (3,2,4)[2] (3,2,4,1)[1] (3,2,4,1,5)[1] (3,2,4,1,5,2)[1]

(2)[2] (2,4)[2] (2,4,1)[1] (2,4,1,5)[1] (2,4,1,5,2)[1]

(4)[4] (4,1)[1] (4,1,5)[1] (4,1,5,2)[1]

(1)[1] (1,5)[1] (1,5,2)[1]

(5)[5] (5,2)[2]

(2)[2]

Sum = 36

TC: O(N) SC: O(N)

	0	1	2	3	4	5
arr[] =	3	2	4	1	5	2

nseli = { -1, -1, 1, -1, 3, 3 }

nseri = { 1, 3, 3, 6, 5, 6 }

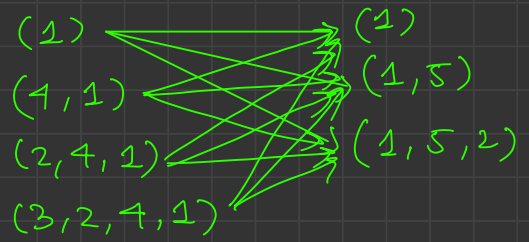
$$(i - nseli[i]) * (nseri[i] - i)$$

↓

No. of subarrays where i^{th} is min!

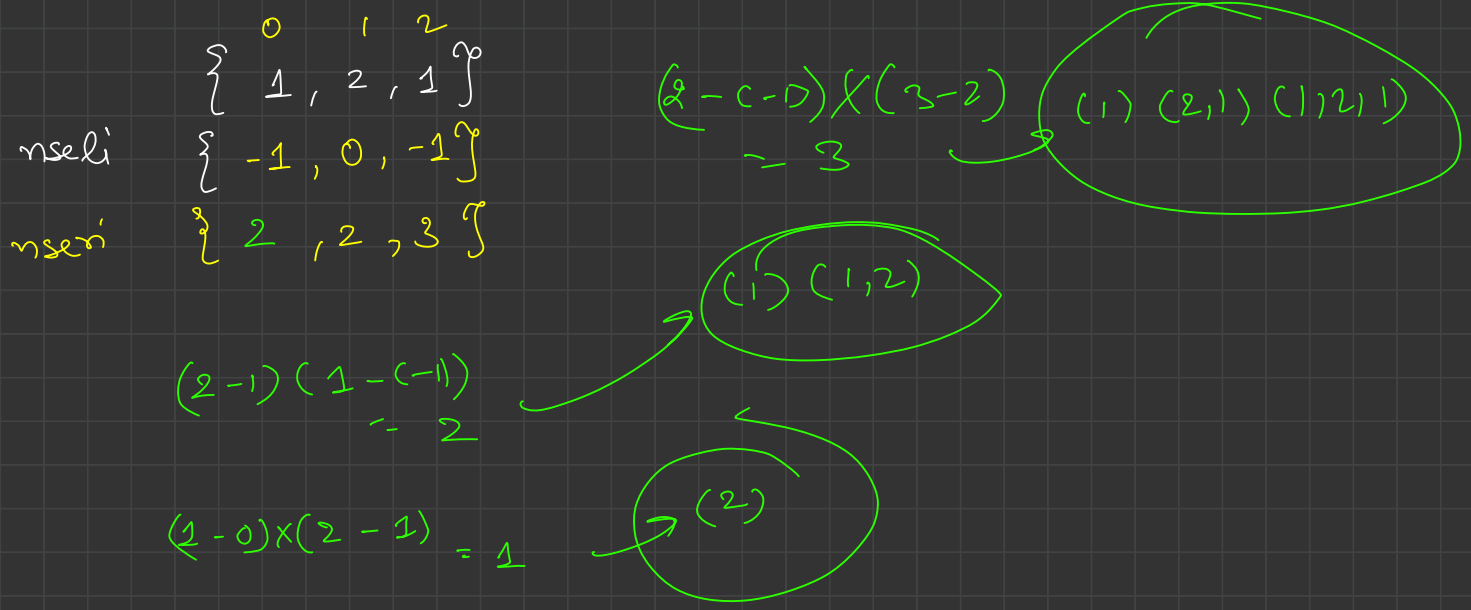
1 * 1

2 * 2 = 4



{ (1) (1,5) (1,5,2) (4,1) (4,1,5) (4,1,5,2) (2,4,1) (2,4,1,5) (2,4,1,5,2) (3,2,4,1) (3,2,4,1,5) (3,2,4,1,5,2)

```java
// step 3: calc number of subarray where arr[i] is min, and then calc there sum
long totalSum = 0;
for (int i = 0; i < n; i++) {
    long num = ((i - nseli[i]) * (nseri[i] - i)) % mod;
    long sum = (num * a[i]) % mod;
    totalSum = (totalSum + sum) % mod;
}
```



Minimum Stack

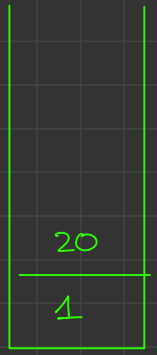
→ push

→ pop

→ getMin
   ↳ returns minEle. present in Stack.

eg:

st.push(1)
st.getMin() ~> 1
st.push(20)
st.getMin() ~> 1
st.push(-10)
st.getMin() ~> -10



st.pop()
st.getMin() ~> 1

st.push(1) → TC O(1)

st.getMin() ~ 1 → TC O(N)

st.push(20)

st.getMin() ~ 1

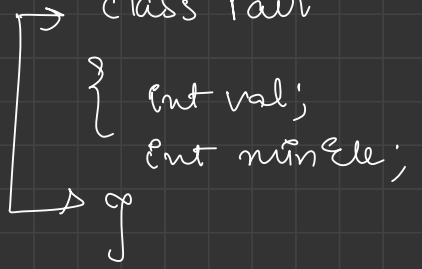
↳ O(1)? ∘

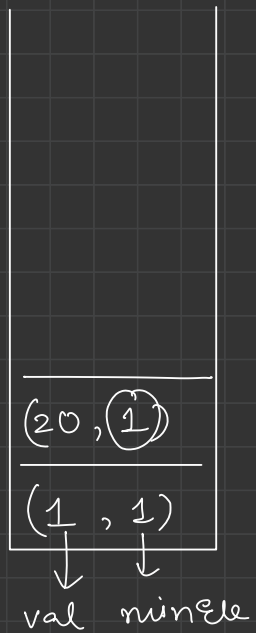
st.push(-10)

st.getMin() ~ -10

st.pop() → -10 TC O(N)

st.getMin() ~ 1





minEle = ~~1~~ -10

st.push(100)

st.getMin() → 100

st.push(20)

st.getMin() → 100

st.push(10)

st.getMin() → 10

st.push(5)

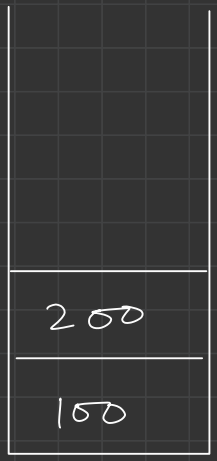
st.getMin() → 5

st.pop(); → 5

getMin() → 10

pop()

getMin()



minEle = ~~100~~ 10

x = val - minEle

prevMin = minEle - peek

= 10 - (-90)

= 100