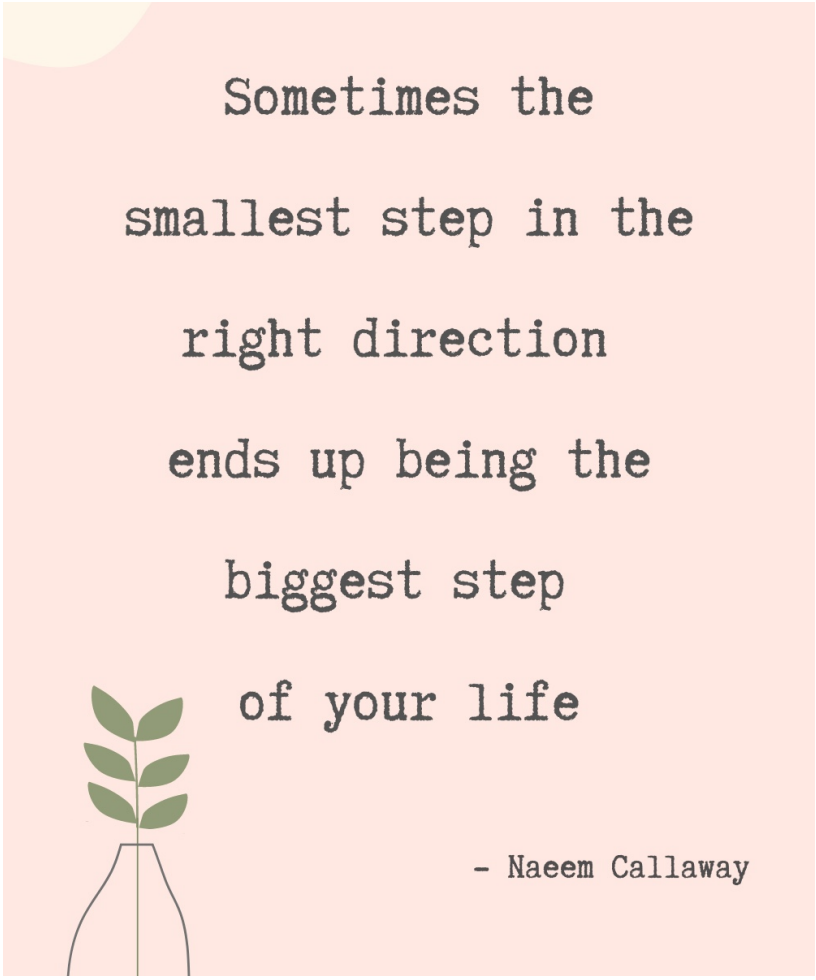
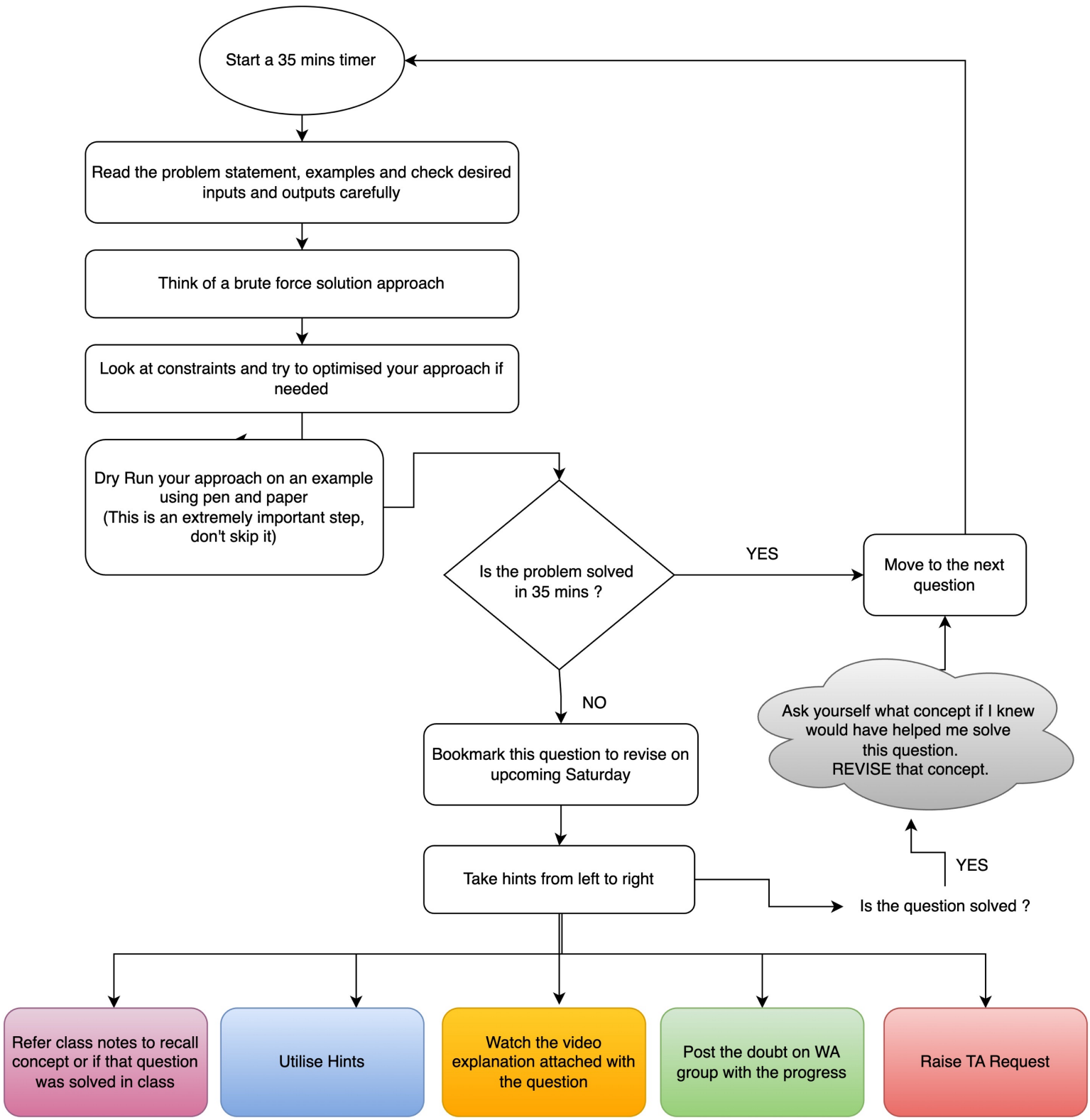


One Dimensional Array

TABLE OF CONTENTS

How to approach a problem ?



< *Question* > : Given arr[N]. Find the maximum subarray sum out of all subarrays.

$1 \leq N \leq 10^5$

Example 1 : arr[ ] → [-3 , 2 , 4 , -1 , 3 , -4 , 3]

0 1 2 3 4 5 6

8

Example 2 : arr[ ] → [4 , 5 , 2 , 1 , 6]

0 1 2 3 4

18

Example 3 : arr[ ] → [-4 , -3 , -6 , -9 , -2]

→ -2

Idea -1 Brute Force

→ (1) Find all the subarrays. Sum
(2) Compare them and take the largest.

T.C: $O(N^3)$

→ Carry forward T.C. $O(N^2)$

Observations

Case 1

When all elements are positive

2	4	5	1	3

⇒ Sum of all the elements.

Case 2

When all elements are negative

-7	-3	-11	-8	-15

⇒ largest number

Case 3

	+ve	

⇒ 1 2 3 4 -2 5 3.

1 2 3 4 -20 5 3

at any given point if sum becomes 0 or negative. leave and start fresh.

Kadane's Algorithm

Minimum sum
→ if all -ve add zero
→ if all +ve smallest number.
→ if -ve and +ve

*Dry-Run

arr[ ] → [5 , 6 , 7 , -3 , 2 , -10 , -12 , 8 , 12 , -4 , 7 , -2]

		0	1	2	3	4	5	6	7	8	9	10	11
0	cur_sum ⇒	5	11	18	15	17	7	0	8	20	16	23	21
0	max_sum ⇒	5	11	18	18	18	18	18	18	20	20	23	25

ans = 25

A=[ ] [-2, 3, 4, -1, 5, -10, 7]

0	run_sum	0	3	7	6	11	1	8
-inf	max_sum	-2	3	7	7	11	11	11

ans = 11

</> Code

```
function maximumSubarray ( arr[ ], n )
{
    max_sum = -inf   // -∞ -1
    run_sum = 0
    for ( i=0; i<n; i++ )
    {
        run_sum += arr[i];
        if ( run_sum > max_sum )
        {
            max_sum = run_sum;
        }
        if ( run_sum < 0 )
        {
            run_sum = 0
        }
    }
    return max_sum;
}
```

[-2, -1, -5]

T.C: O(N)
S.C: O(1)

10:15 pm

Continuous Sum Query

< *Question* > : Initially all elements of an arr[N] are 0. Then you are given Q queries. Every query contains i-idx and value. Increment elements from ith.idx to last idx by value. Return final state of arr[ ].

$1 \le N \le 10^5$

$1 \le Q \le 10^5$

arr[ ] → [0 0 0 0 0 0 0]
0 1 2 3 4 5 6

Queries → 3

idx	val
3	4
1	3
4	-2

[0, 0, 0, 0, 0, 0, 0]
0 1 2 3 4 5 6

+4 +4 +4 +4.

+3 +3 +3 +3 +3 +3

-2 -2 -2

result = [0, 3, 3, 7, 5, 5, 5]

BF Idea

A : [0 0 0 0 0]
0 1 2 3 4

1, 3 → +3 +3 +3 +3

0, 2 → +2 +2 +2 +2

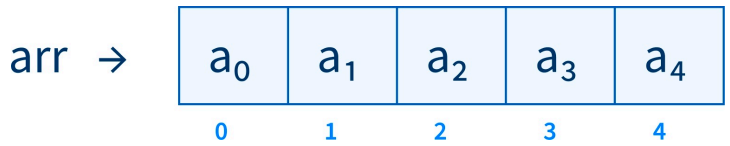
4, 1 → +1

[2, 5, 5, 5, 6].

1) iterate over every query. → Q
   ↳ add the value from ith to nth index. n

TC = O(Q * n).

Observation



1, 3 +3

2, -1 -1

4, 2 2

1, 5

Q = [[1, 3], [2, -1], [1, 5],]

T.C: O(N+Q)

arr =

0	1	2	3	4
0	8	-1	0	0

psum.

0	1	2	3	4
0	8	7	7	7

idx value.

Q[ ][ ]

arr = {0}

</> Code

```
for ( i = 0 ; i < Q ; i++ )
{
    idx = Q[i][0]
    val = Q[i][1]
    arr[idx] += val;
}
```

// processing query

T.C: O(N+Q)
S.C: O(1).

// Calculate prefix sum

```
for ( i → 1 to N )
{
    arr[i] += arr[i-1];
}
```

< *Question* > : Initially all elements of an arr[N] are 0. Given Q queries.

Every query contains [s, e, val]. Increment elements from s to e by val.

Return the final state of arr[ ].

arr[10] → [0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0]

0 1 2 3 4 5 6 7 8 9

$1 \le N \le 10^5$

$1 \le Q \le 10^5$

Queries → 3

s	e	val
3	6	3
2	7	-3
1	9	4

0 1 2 3 4 5 6 7 8 9

+3 +3 +3 +3

−3 −3 −3 −3 −3 −3

4 4 4 4 4 4 4 4 4.

0 4 1 4 4 4 4 1 4 4.

0 1 2 3 4 5 6 7 8 9

[0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

+3 −3

−3 +3

4

→ [0, 4, −3, 3, 0, 0, 0, −3, 3, 0]

↳ prefix sum ↗.

Quiz :

arr[8] → [0 , 0 , 0 , 0 , 0 , 0 , 0 , 0]
0 1 2 3 4 5 6 7

Queries → 4

si	ei	val
1	4	3
0	5	-1
2	2	4
4	6	3

0 1 2 3 4 5 6 7
[0, 0, 0, 0, 0, 0, 0, 0]

3 -3

-1 1

4 -4

3 -3.

= [-1, 3, 4, -4, 3, -3, 1, -3]

↓ prefix sum

H.W

Write optimized code for this.

BF Idea

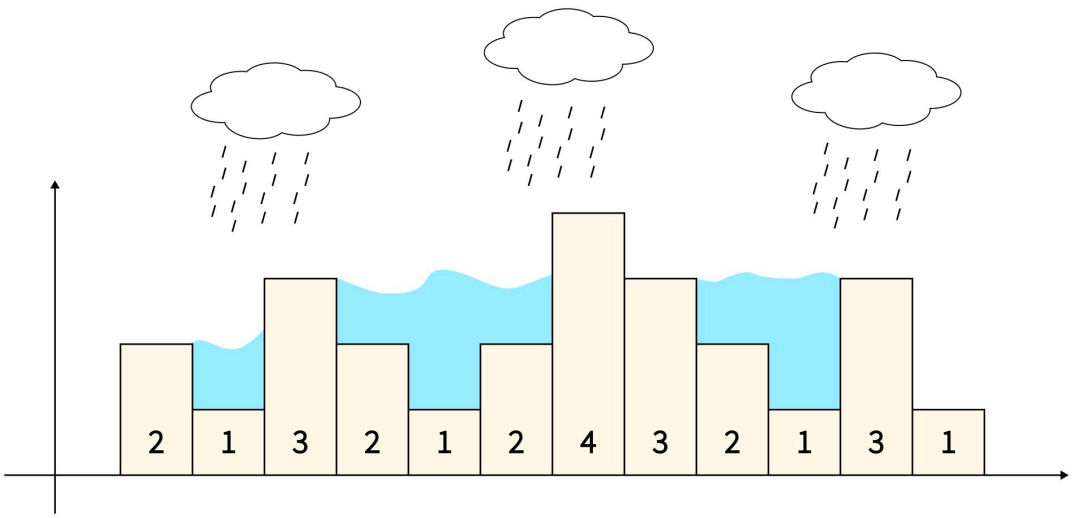
< / > *Code*

< *Question* > : Given arr[N], where arr[i] → height of building.

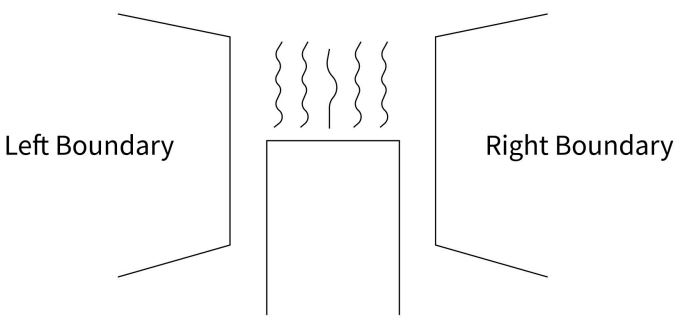
Return amount of water trapped on all the buildings.

arr [2 1 3 2 1 2 4 3 2 1 3 1]

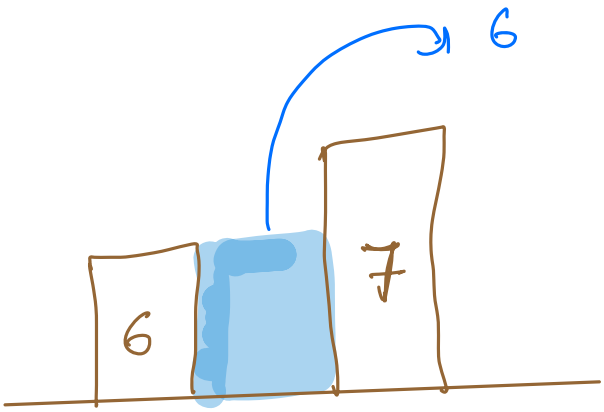
$1 \leq N \leq 10^5$

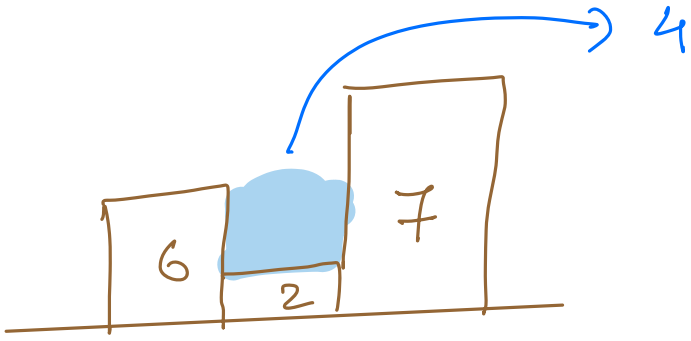


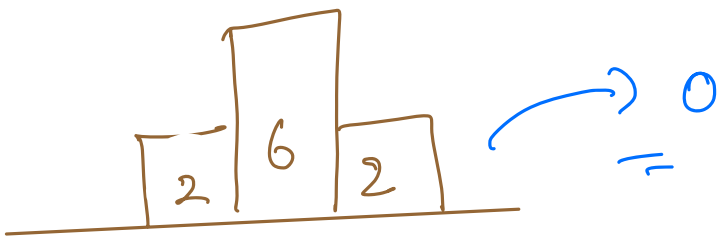
Idea -1 Find the amount of water trapped on every building.



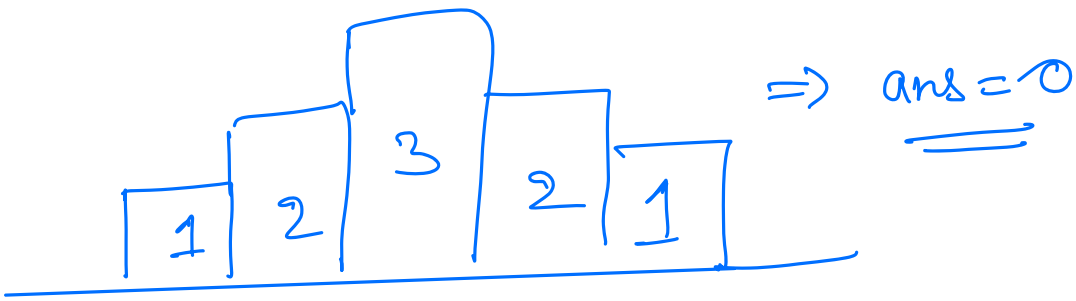
Observation





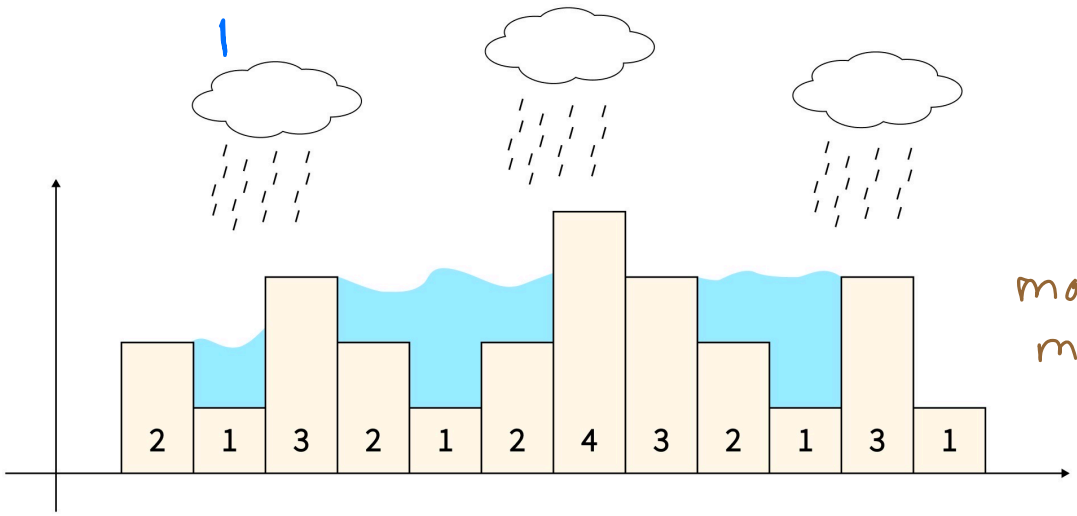


Quiz



< / > Code

Brute force



maxL = 3
maxR = 3
= 3 - 4
= -1

max height building towards left

max height building toward right.

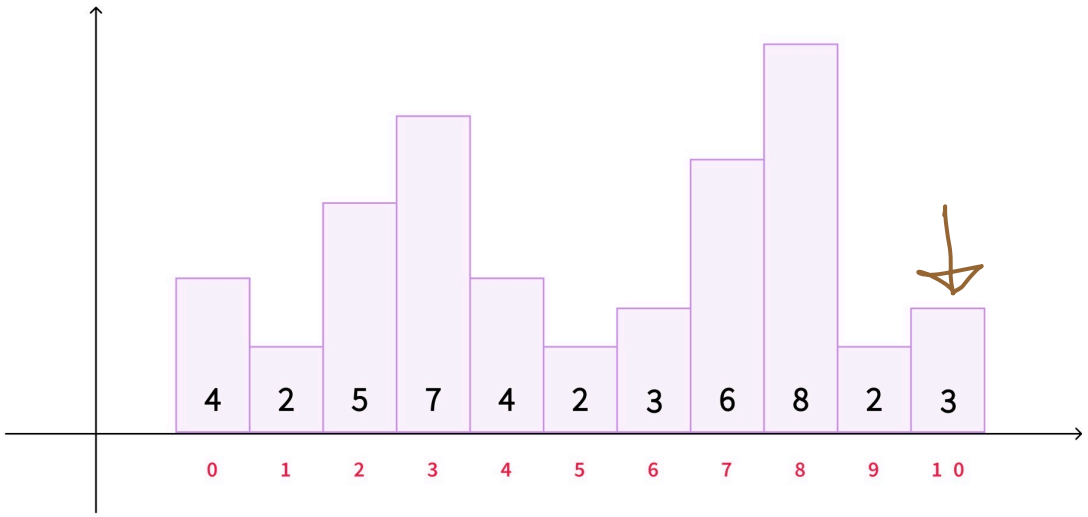
$\Rightarrow$ min (max(left) , max(right)) − A[i]

Brute force

```
ans = 0
for ( i = 1 to N-2)                    O(N)
{
    maxL = max ( 0, i-1 );             O(n)
    maxR = max ( i+1, n-1 );           O(n)
    water = min ( maxL, maxR ) - A[i]
    if (water > 0)  ans += water;
}
```

T.C: $O(N^2)$

***Dry-Run**



lmax[] → [4 4 5 7 7 7 7 7 8 8 8]

rmax[] → [8 8 8 8 8 8 8 8 3 3]

⇒ min(lmax[i], rmax[i]) − A[i]

T.C: O(N)

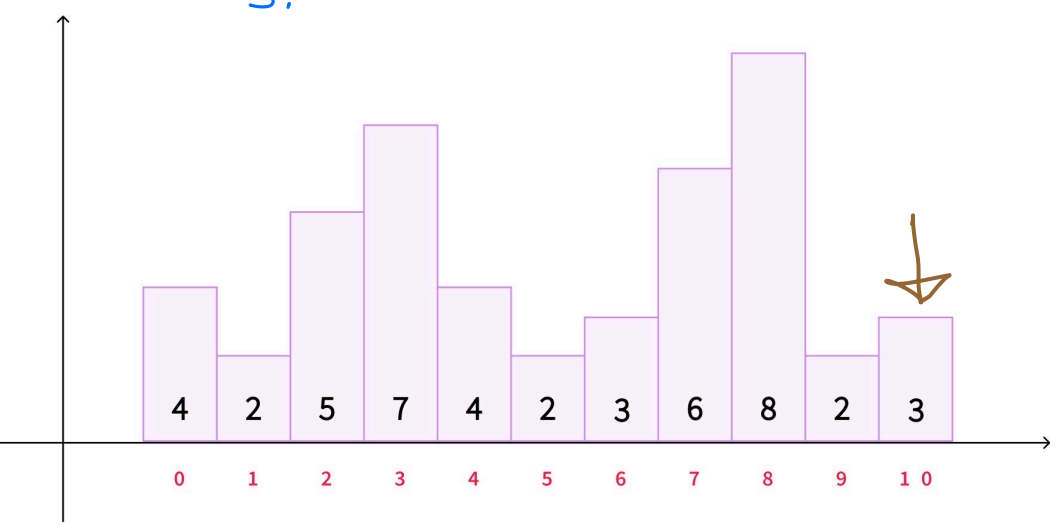
S.C: O(N)

< / > **Code**

Optmised approach

```
lmax = { 0 }
for ( i = 1 to N-1 )
    lmax[i] = max ( lmax[i-1], A[i-1] )
rmax = { 0 }
for ( i -> N-2 to 0 )
    rmax[i] = max ( rmax[i+1], A[i+1] )

for ( i = 1 to N-2 )  O(N)
{
    water = min ( lmax[i], rmax[i] ) - A[i]
    if (water > 0) ans += water;
}
```



lmax[] → [4 4 5 7 7 7 7 7 8 8 8]

rmax[] → [8 8 8 8 8 8 8 8 8 3 3]