SCALER

One Dimensional Array

TABLE OF CONTENTS

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

$1 \le N \le 10^5$

Example 1 : arr[ ] → [-3 , 2 , 4 , -1 , 3 , -4 , 3] ⇒ 8
0 1 2 3 4 5 6

Example 2 : arr[ ] → [4 , 5 , 2 , 1 , 6]
0 1 2 3 4

full array

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

biggest elem

Idea -1

try for all subarrays & get max sum

$O(N^2)$ using carry forward.

Observations

Case 1

When all elements are positive

2	4	5	1	3

⇒ take full array

Case 2

When all elements are negative

-7	-3	-11	-8	-15

⇒ take only 1 elem. Biggest elem

Case 3

	+ve	

Kadane's Algorithm

3 4 -1 -5 -3 2

3 7 6 1 -2 2

0

-3 -2 -5

ans = INT_MIN -3 -2

sum = 0 -3 0 -2 0 -5 0

*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

= 5 11 18 15 17 7 ~~-5~~ 8 20 16 23 21

0

-2 3 4 -1 5 -10 7

ans = ~~INT_Min~~ ~~-2~~ ~~3~~ ~~7~~ 11

sum = ~~0~~ ~~-2~~ ~~0~~ ~~3~~ ~~7~~ ~~6~~ ~~11~~ 7 8

</> Code

```
ans = INT_MIN        Integer.Min_Value,
                     sys.minsize
sum = 0
for( i:0 → n-1) {
    sum += arr[i]
    if( sum > ans )
        ans = sum
    if ( sum < 0)
        sum = 0
}
print (ans)
```

TC : O(N)

SC : O(1)

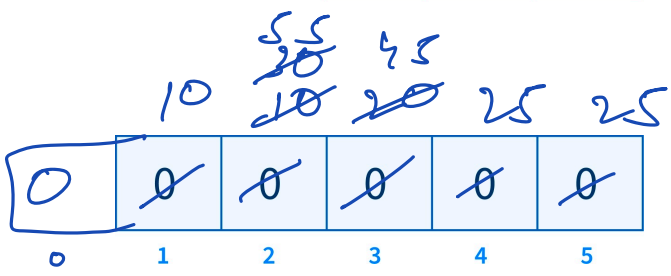
Continuous Sum Query

< *Question* > : There are A beggars sitting in a row outside a temple. Each beggar has an empty pot initially. There are N devotees. Each devotee gives some fixed amount to beggars from indices l to r.

Given a 2-D Array B[N][3], where B[i][0] represents l [left index]
B[i][i] represents r [right index]
B[i][2] represents amount

Example : A = 5 , B[N][3] →

1	2	10
2	3	20
2	5	25



ans ⇒ 0, 10, 55, 45, 25, 25

BF Idea

Idea -1

< / > *Code*

< *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]

idx: 0 1 2 3 4 5 6

3 4 4 -2

0 3 7 11 9 9 9

Queries → 3

idx	val
3	4
1	3
4	-2
2	4

0 3 3 7 5 5 5

0 0 0 0 0

0 1 2 3 4

3 3 3 3

2 2 2 2 2

1

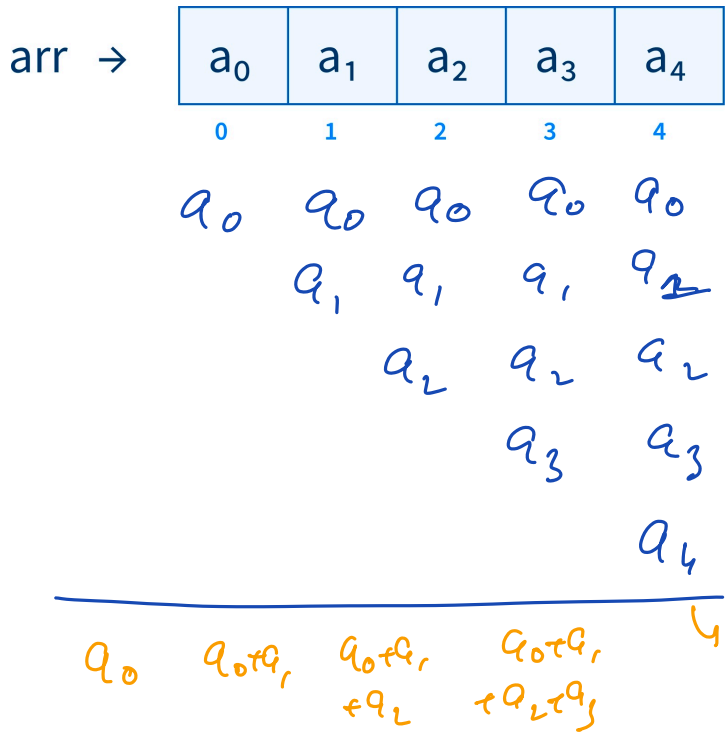
2 5 5 5 6

BF Idea

for each query, iterate from start idx till the end & add the given value

TC: O(QN)

Observation



idx val

< / > Code

```
// initially i have all which has all 0's
for( i : 0 → Q-1 ){
|     arr[ idx[i] ] += val[i]
}
for( i : 1 → n-1 ){                    TC : O(N+Q)
|     all[i] += all[i-1]
}
```

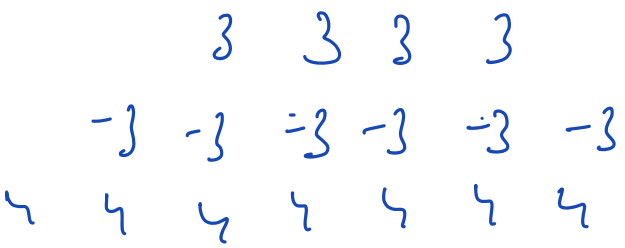
< *Question* > : Initially all elements of an arr[N] are 0. Given Q queries.

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

Quiz :

arr[8] → [0 , 0 , 0 , 0 , 0 , 0 , 0 , 0]

0 1 2 3 4 5 6 7

3 3 3 3 3 3 3

-3 -3 -3

Queries → 4

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

1 add 3

5 add -3

s, e, val

↳ s, val

↳ e+1, -val

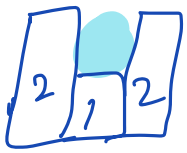
BF Idea

Iterate for all queries

TC: O(QN)

< / > Code

```
for( i: 0 → Q-1 ){
        arr[start(i)] += val[i]
        if( end[i]+1 ≤ n-1 )
            arr[end[i]+1] -= val[i]
}
for( i: 1 → n-1 ){                    TC: O(N+Q)
    arr[i] += arr[i-1]
}
```

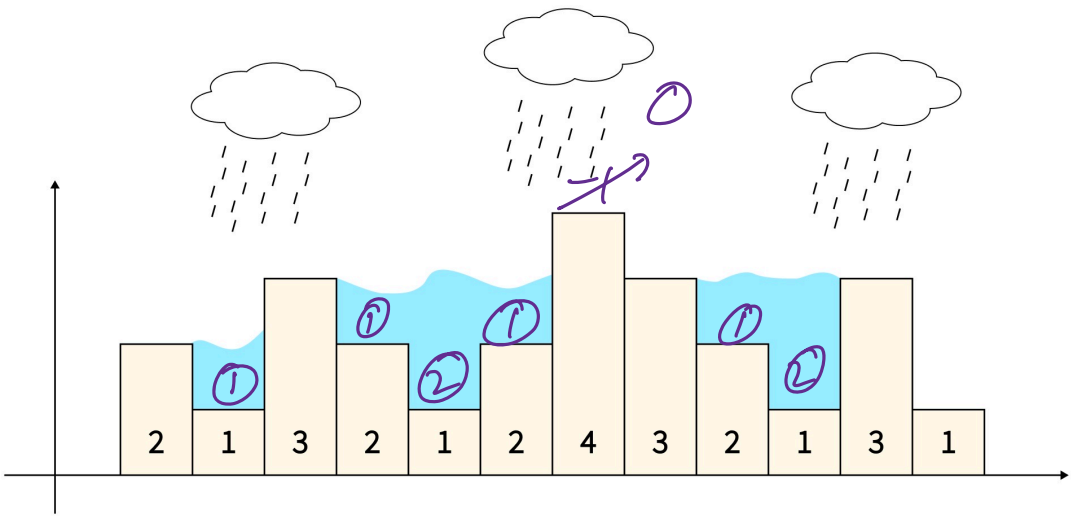


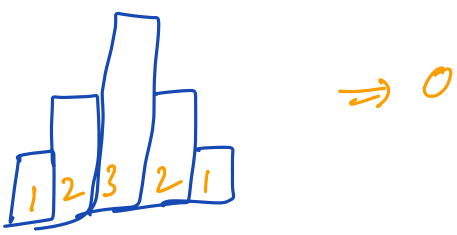
< *Question* > : Given arr[N], where arr[i] → height of building.

Return amount of water trapped on all the buildings.

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

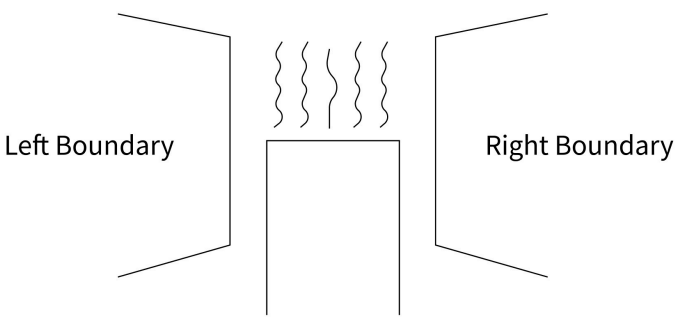
$1 \leq N \leq 10^5$





⇒ 0

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



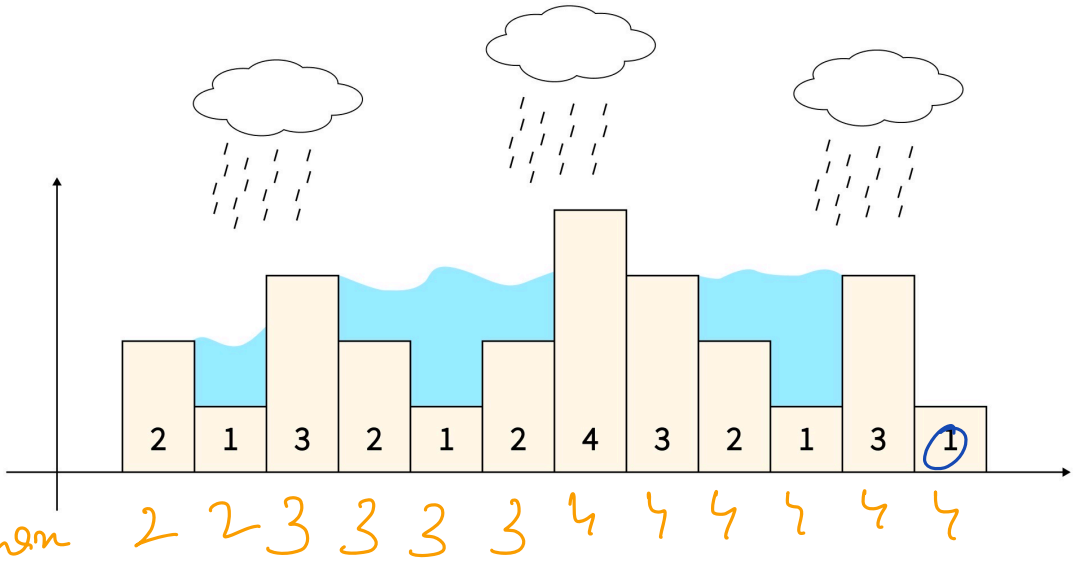
Observation

1) Left support = biggest building on left
2) right support = biggest building on right
3) net support = min(left_support, right support)
4) if net_support > h[i]
   water += net_support - h[i]
5) No water on ends

BF idea

⇒ for each building,
iterate on left for left_sup
iterate on right for right_sup

< / > Code



lmax 2 2 3 3 3 3 4 4 4 4 4 4

4 4 4 4 4 4 4 3 3 3 3 1

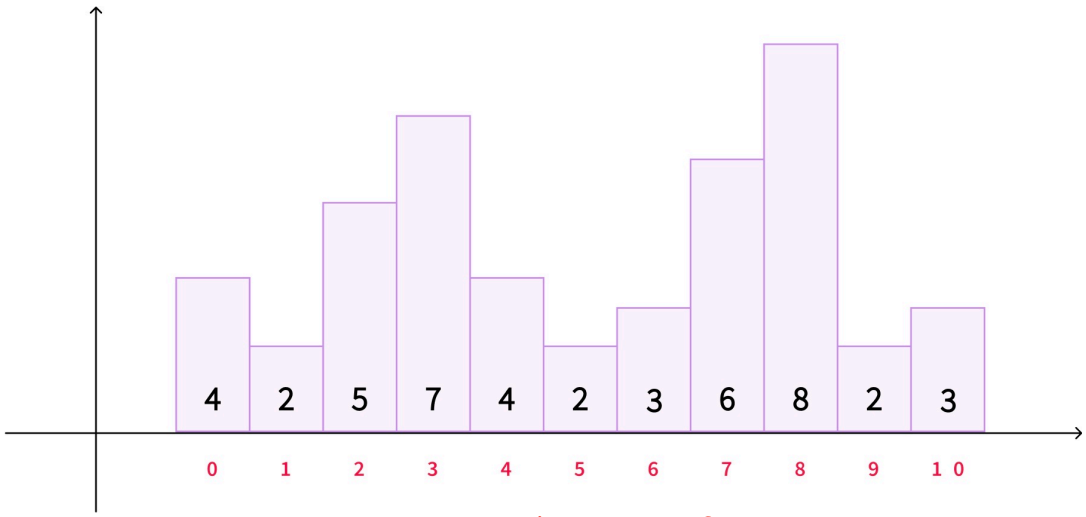
lmax ⇒ max till index (i)

```
pf max[0] = arr[0]
for( i: 1 → n-1) {
|   pf max[i] = max(pf max[i-1], arr[i])
}
```

rmax

```
rmax[n-1] = arr[n-1]
for( i: n-2 ⟶ 0 ){
|   rmax[i] = max(rmax[i+1], arr[i])
}
```

*Dry-Run



0 1 2 3 4 5 6 7 8 9 10

lmax[] → [4 4 5 7 7 7 7 7 8 8 8]

rmax[] → [8 8 8 8 8 8 8 8 8 3 3]

< / > Code

```
lmax[0] = arr[0]
for( i: 1 → n-1) {
|    lmax[i] = max( lmax[i-1], arr[i])
}
```

rmax

```
rmax[n-1] =  arr[n-1]
for( i: n-2 ⟶ 0 ){
|   rmax[i] = max (rmax[i+1], arr[i])
}
```

```
water = 0
```

```
for ( i: 1 ⟶ n-2 ) {
        left_support = lmax[i-1]
        right_support = rmax[i+1]
        net_support = min(left_support,
                          right_support)
        if( net_support > h[i] )
            water += net_support - h[i]
}
```

TC: $O(N)$

SC: $O(N)$