"Life is not a problem to be solved"

→ Skill Evaluation Contest

→ Mock interview

Q1) given an integer array ar[n], find the max subarray sum out of all subarrays.

eg:- ar → { −2 | 3 4 −1 5 | −10 7 }
0 | 1 2 3 4 | 5 6
→ sum = 11.

ar → { −3 | 4 6 8 | −10 2 7 }
→ sum = 18

ar → { 4 5 2 1 6 }
sum = 18

ar → { −4, −3, −6, −9, −2 }
sum = −2.

Brute Force :-
→ Look at all subarrays → O(n²)
→ find sum of each → O(n) (independently)
n² * n
↓
O(n³) T.C.

(i, j) → + ar[j+1]
(i, j+1)

```
fn maxSum(arr[], n) {
    max = -∞
    for(i → 0 to n-1) {
        sum = 0
        for(j → i to n-1) {
            sum += arr[j]
            if(sum > max) {
                max = sum
            }
        }
    }
    return max
}
```

$O(n^2)$ T.C.
$O(1)$ S.C.

Cases :

1) All Positive {3, 4, 7, 1, 2}

sum(arr)

2) All Negative {-3, -4, -7, -1, -2}

max(arr)

3) If +ve are placed in betn. {- - - - (++++) - - - -}

take all +ves

4) {+++ - - -} or {- - - - - +++++}

take all +ves

5) {(++++) - - - - (++++)}

{4 5 -3 -9 1 1}

{4 5 -9 -80 12 11}

{4 15 -8 -10 12 8}

{-2, 3, 4, -1, 5, -10, 7}

max sum ending here	-2	3	7	6	11	1	8

max_sum ending at i

- → ar[i]
- → ar[i] + max_sum ending at (i-1)

{-3, -4, 5, -1, 2, -4, 6, -1}

-3	-4	5	4	6	2	8	7
0	0						

8.

[Kadane's Algorithm]

max_sum[i] = max sum of subarray ending at i

$$= \max\Big(arr[i],\ arr[i] + arr[i-1],\ arr[i] + arr[i-1] + arr[i-2],\ \vdots\ arr[i] + arr[i-1] + \cdots + arr[0] \Big)$$

$$= arr[i] + \max\Big(0,\ arr[i-1],\ arr[i-1] + arr[i-2],\ \vdots\ arr[i-1] + arr[i-2] + \cdots + arr[0] \Big)$$

$$max\_sum[i] = arr[i] + \max(0, max\_sum[i-1])$$

$$\left[ans = \max_{i} \left(max\_sum[i] \right) \right]$$

```
fn max_sum(ar[], n){
    max = -∞, curr = 0
    fn(i → 0 to n-1){
        curr += ar[i]
        if(curr > max)
            max = curr

        if(curr < 0)
            curr = 0

    }
    return max
}
```

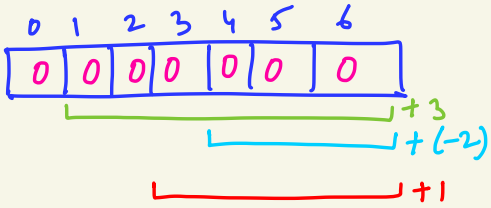
O(n) T.C.
O(1) S.C.

Q2) Given arr[n], where each element is 0, return final array after performing multiple queries.

query(i, x) → Add x to all nos. from i to n-1.

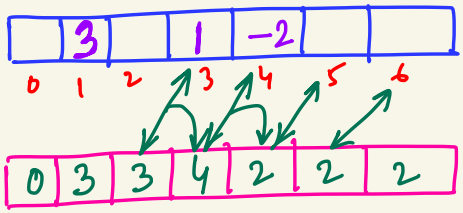
eg: n = 7

queries: (1, 3)
(4, -2)
(3, 1)



0 3 3 3 3 3 3
0 3 3 3 1 1 1
0 3 3 4 2 2 2

O(n*q) T.C.
O(1) S.C



```
for(i -> 0 to q-1){
    ar[Q[i][0]] += Q[i][1]
}
```
index — x

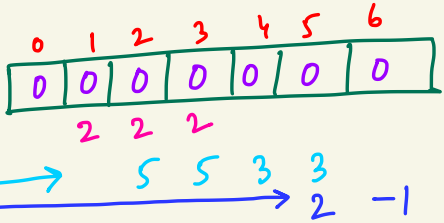
O(n+q) T.C.
O(1) S.C

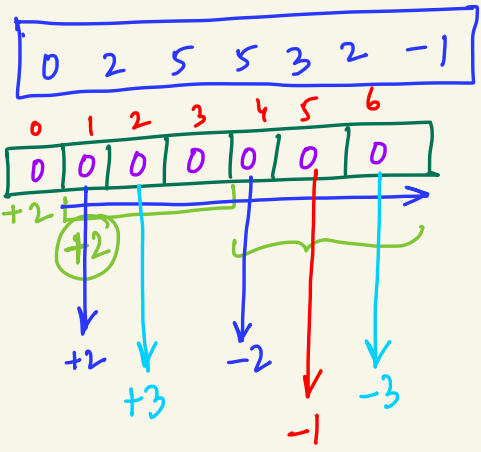
```
for(i -> 1 to n-1){
    ar[i] += ar[i-1]
}
return ar
```

Q 3) ar[n], queries

Query : (i, j, x) → Add x to all the elements from i to j.
i <= j.

n=7
(1,3,2)
(2,5,3)
(5,6,-1)





n=7
(1,3,2) → (1,6,2), (4,6,-2)
(2,5,3)
(5,6,-1)

0	2	5	5	3	2	-1

```
fn combineQueries(ar[], Q[][], n, q){
    fn(i→0 to q-1){
        l = Q[i][0]
        r = Q[i][1]
        x = Q[i][2]
        ar[l] += x
        if(r<n-1){
            ar[r+1] -= x
        }
    }
    fn(i→1 to n-1){
        ar[i] += ar[i-1]
    }
    return ar
}
```

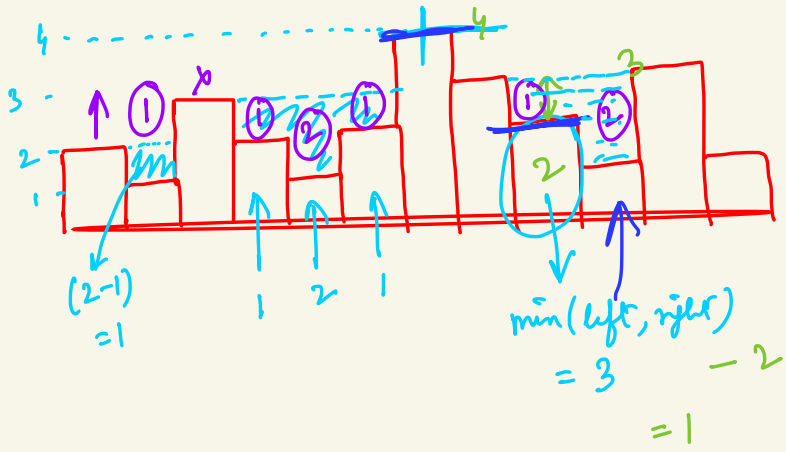
O(n+q) T.C.

O(1) S.C.

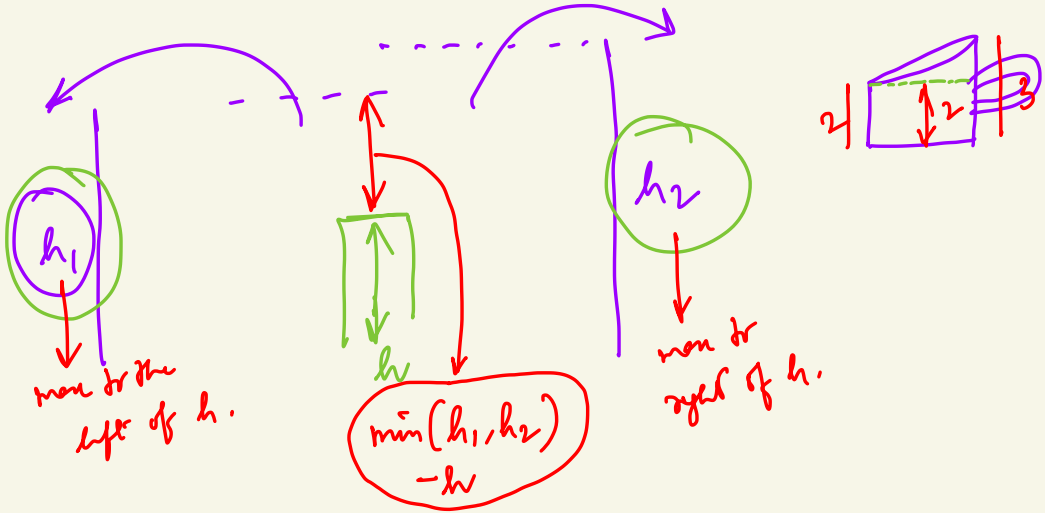
Q4) Trapping Rainwater Problem

Given n buildings with their heights, find the rain water trapped between the buildings.

ar → { 2, 1, 3, 2, 1, 2, 4, 3, 2, 1, 3, 1 }



Ans → 8.



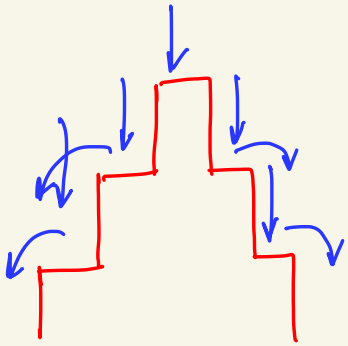
$$ans = \sum_i ans[i]$$

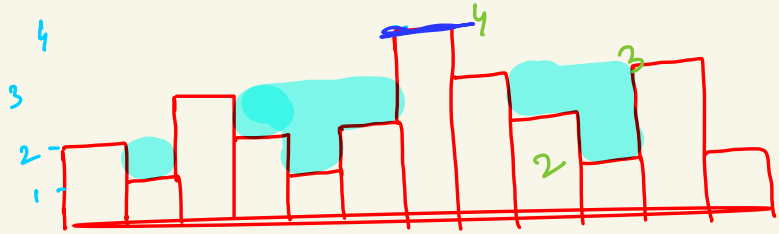
$ans[i]$ = ht. of water above building i

$$= \sum \left[\min(max\_left[i], max\_right[i]) - ar[i] \right]$$

$max\_left[i]$ → $\max_{j \in [0,i]} (ar[j])$

$max\_right[i]$ → $\max_{j \in [i,n-1]} (ar[j])$





Brute force :

$O(n^2)$ T.C.
$O(1)$ S.C.

for every i :

calculate max_left, max_right

(min(max_left, max_right) − ar[i])

Optimisation :

Precalculate all left_max and right_max

	4	2	5	7	4	2	3	6	8	2	3
lm →	4	4	5	7	7	7	7	7	8	8	8
rm →	8	8	8	8	8	8	8	8	8	3	3
ans →	0	2	0	0	3	5	4	1	0	1	0

16.

```
ans = 0
left_max = [0]*n
left_max[0] = ar[0]
for( i → 1 to n-1) {
    left_max[i] = max(ar[i], left_max[i-1])
}
right_max = [0]*n
right_max[n-1] = ar[n-1]
for( i → n-2 to 0) {
    right_max[i] = max(ar[i], right_max[i+1])
}
ans = 0
for(i → 0 to n-1){
    ans += min( left_max[i], right_max[i]) - ar[i]
}
return ans
```

$O(n)$ T.C.
$O(n)$ S.C.