ARRAYS 1

Good Evening

Today's content

01. Zero Queries I
02. Zero Queries II
03. Rain water Trapping
04. Max Subarr sum (Kadane's Algo)

01. Initially all elements of an arr[] are 0 & you are given Q queries. Every query contains idx & value. Increment elements from i^th idx to last idx by value. Return final state of arr[].

arr[] = { 0 0 0 0 0 0 0 }
0 1 2 3 4 5 6

Queries

idx	val
3	4
1	3
4	-2

+4 +4 +4 +4
+3 3 3 3 3 3
-2 -2 --2

0 3 3 7 5 5 5

Brute force → For every query, iterate from idx to n-1 & update the values of arr.

TC: O(Q * N)

SC: O(1)

Idea 2

a_0	a_1	a_2	a_3	a_4

Prefix =

a_0 a_0 a_0 a_0 a_0
\+ \+ \+ \+
a_1 a_1 a_1 a_1
\+ \+ \+
a_2 a_2 a_2
\+ \+
a_3 a_3
\+
a_4

Queries

idx	val
3	4
1	3
4	-2
1	-3

arr[] = { 0 +3 0 +4 -2 0 0 }
0 1 2 3 4 5 6

pr[] = { 0 3 3 7 5 5 5 }

Idea → Just add the values at given indexes & then create psum array.

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

    arr[idx] += val;
```

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

// Create psum array using arr.

02 Initially all elements of an arr[] are 0 & you are given Q queries. Every query contains [s, e, val]. Increment elements from s to e by value. Return final state of arr[].

arr[] = { 0 0 0 0 0 0 0 0 0 0 }
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

Queries

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

arr[] = { 0 4 -3 3 0 0 0 0 0 0 }
0 1 2 3 4 5 6 7 8 9

-3 -3 -3 -3 -3 -3 -3 -3

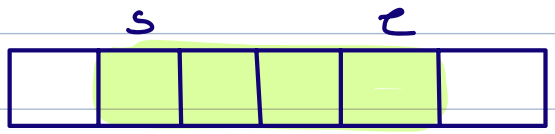
Queries

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

s	val
3	+3
2	-3
1	4

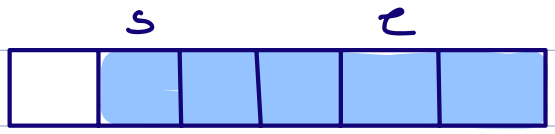
counterpart

e	val
7	-3
8	+3
10	-4



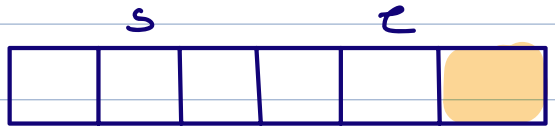
s e val

arr[s n-1] += val



arr[s] += val

arr[e+1 n-1] -= val



arr[e+1] -= val

```
sumqueries (int [][] Queries, int [] ar)
{
    for (i=0; i<Q; i++)
    {
        s = Queries[i][0]
        e = Queries[i][1]
        val = Queries[i][2]
        arr[s] += val;
        if (e+1 < n) arr[e+1] -= val;
    }
    // convert arr into psum array
}
```

TC: O(Q+N)

SC: O(1)

Q Given an arr[]. Create two arrays lmax[], rmax[] & return it.

lmax[i] = max of all elements from 0 to i

rmax[i] = max of all elements from i to N-1

arr = { 1 -6 3 8 4 5 2 }
0 1 2 3 4 5 6

lmax[] = { 1 1 3 8 8 8 8 }

rmax[] = { 8 8 8 8 5 5 2 }

```
int [] lmax = new int [n];

lmax[0] = ar[0];

for (i=1; i<n; i++){
    lmax[i] = max(ar[i], lmax[i-1]);
}
```

rmax[] → {TODO}

TC: O(N)

SC: O(1)

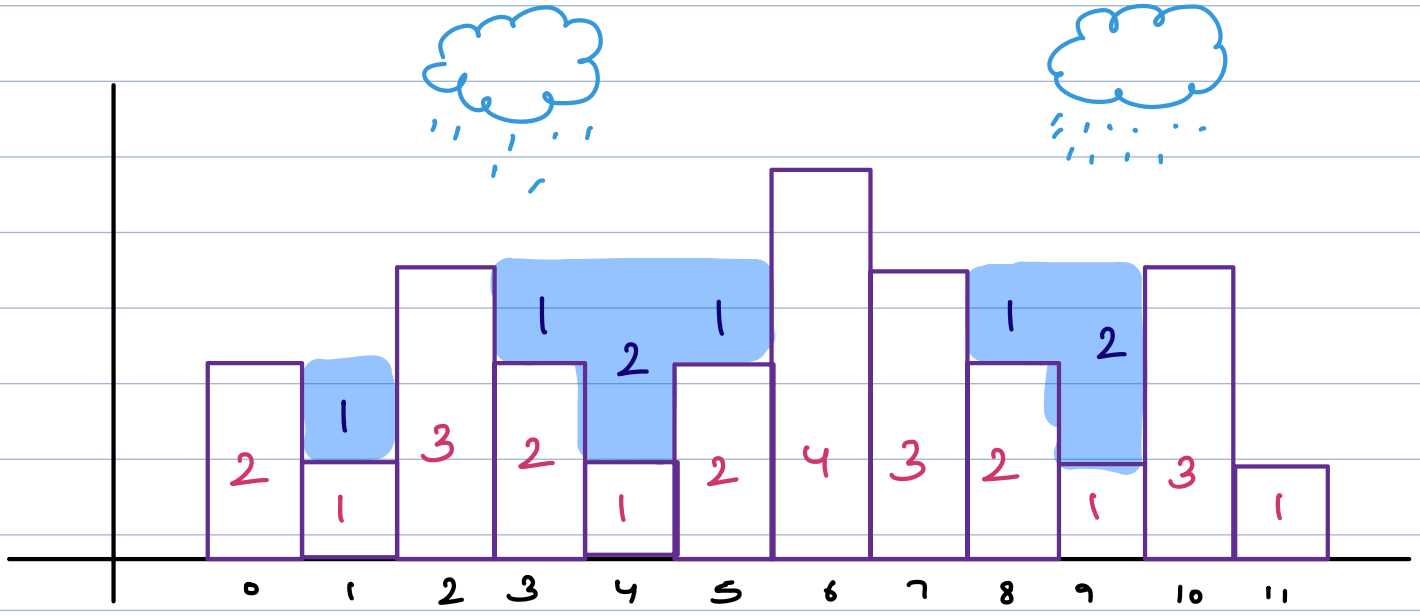
SC → Extra space that our algorithm takes

10:05pm – 10:15 pm

Q Rain water Trapped

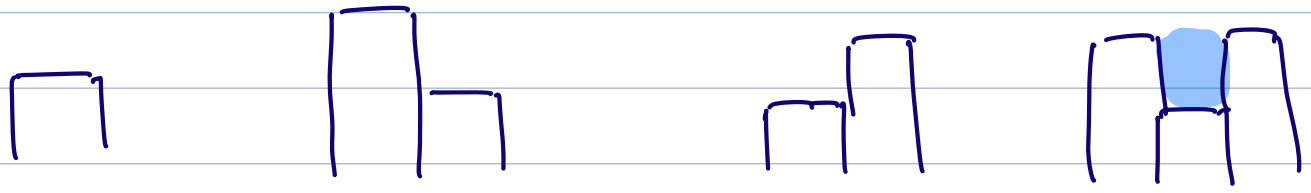
Given n array elements, where ar[i] represents height of the building. Return amount of water trapped on all buildings

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



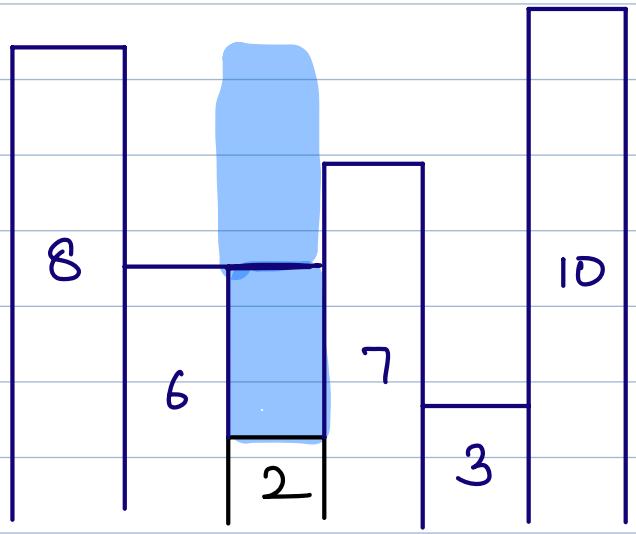
Ans = 8

Condition to accumulate the water

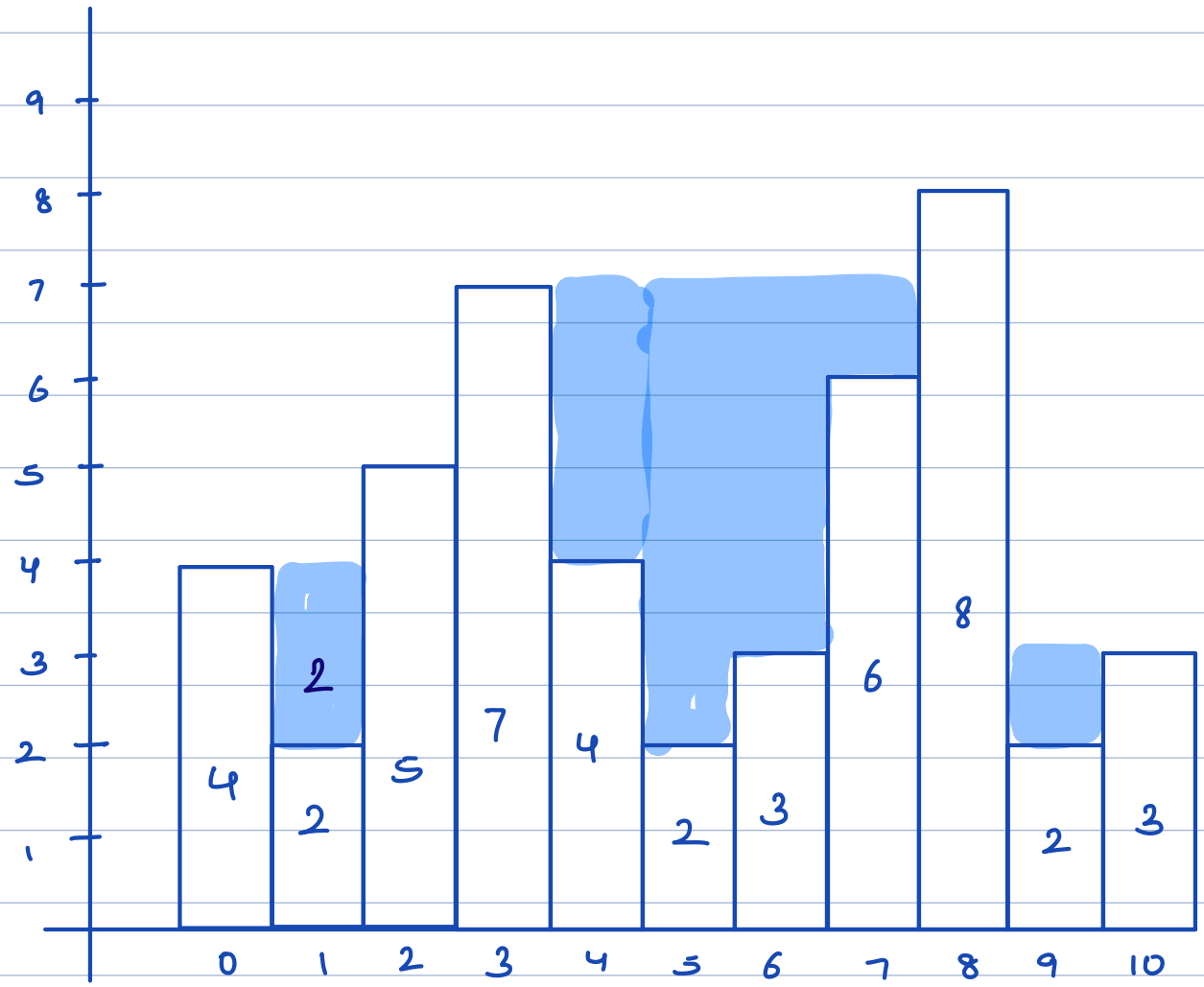


Idea → Calculate the water accumulated at each building & add it.

Observation



water accumulate = min(leftmax, rmax) − ht of building



lmax[] = 4 4 5 7 7 7 7 7 8 8 8

rmax[] = 8 8 8 8 8 8 8 8 8 3 3

min = 4 4 5 7 7 7 7 7 8 3 3

ht = 4 2 5 7 4 2 3 6 8 2 3

water = 0+2+0+0+3+5+4+1+0+1+0 = 16 water

01. Create lmax[] & rmax[] array

```
water = 0 :
for ( i = 1 ; i < n-1 ; i++ ){
    int mini = minimum( lmax[i], rmax[i] )
        ht = ar[i] :
    water += (mini - ht );
}
return water:
```

TC : O(n)
SC : O(n)

O(1) → two pointers

Two pointer → In next class

Maximum subarray sum

Given N arr elements. Calculate the maximum subarr sum

arr[] = { -3 2 4 -1 3 -4 3 }
indices: 0 1 2 3 4 5 6

max subarr sum = 8

Brute force → Consider all the subarrays, for every subarray we need to iterate & find the sum & update max accordingly

TC : O(n³) SC : O(1)

Idea 2 → Create prefix arr[ ] & use for subarr sum

TC: $O(n^2)$
SC: $O(n)$

Idea 3 → Carry forward approach

```
max = -∞
for (i=0; i<n; i++)
    sum=0
    for (j=i; j<n; j++){
        sum += arr[j]
        max = Maximum(max, sum);
    }
}
```

TC: $O(n^2)$
SC: $O(1)$

Constraint → TC of $O(n)$

SC = $O(1)$

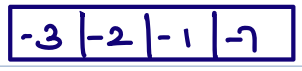
Idea 4 → Kadane's Algo

Case 1 = All elements are positive

3	2	1	7

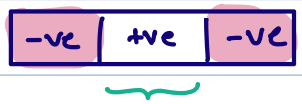
ans = sum of all elements

Case II → All elements are negative



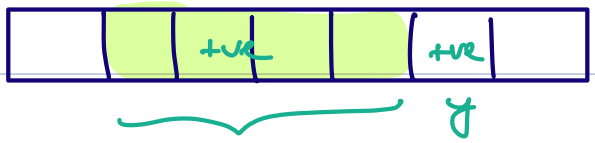
ans = max of array

Case III



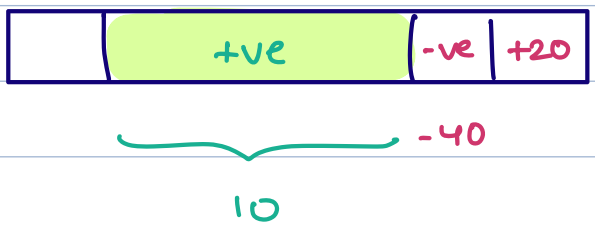
↳ consider the sum of all the +ve elements

Case IV



ans = +x + y → increase the subarr sum by taking +ve ele contribution

Case V



Obs → If eff sum > 0, then ? carry forward it in search of a better subarr sum

sum < 0 → finish the continuing subarr & start a new one.

01. arr = { 2 3 -4 6 }

	0	1	2	3
sum = 0	2	5	1	7
ans = -∞	2	5	5	7

indexes = s = 0, e = ∅, 3

02 arr = { 2 3 -6 7 }

sum = 0	2	5	~~1~~ 0	7
ans = -∞	2	5	5	7

03. arr = { 2 -6 3 7 }

s =

sum = 0	2	~~-4~~ 0	3	10
ans = -∞	2	2	3	10

04. arr = { -4 -3 -1 -7 }

sum = 0	~~-4~~ 0	~~-3~~ 0	~~-1~~ 0	-7
ans = -∞	-4	-3	-1	-1

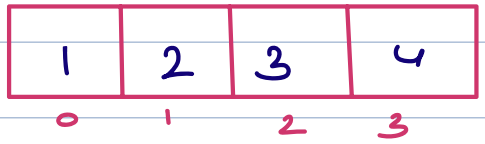
```
sum = 0        ans = -∞

for (i = 0; i < n; i++)
{
    sum += ar[i]
    ans = maximum(ans, sum);
    if (sum < 0) {
        sum = 0
    }
}
return ans.
```

PSP = problem solving percentage

Prefix approach → Using carry forward technique & storing ans in an array.

Carry forward → Carrying the previous ans to generate the next ans

Doubts



sum of all subarray from 0 to last →

sum[0 0] = 1

sum [0 1] = 1 + 2 = 3

sum [0 2] = 3 + 3 = 6

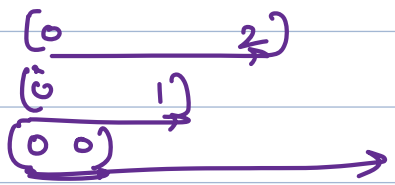
sum [0 3] = 6 + 4 = 10

sum[0 0]
[0 1]
[0 2]
[0 3]

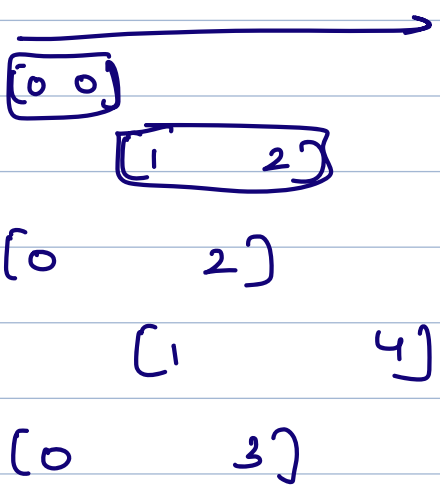
sum[0 2]
sum[1 3]

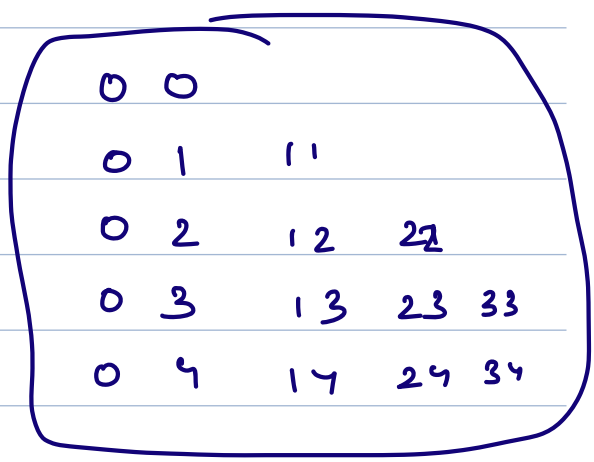
sum[0 3]
sum[2 3]

sum[0 0]



Sum





Sliding window = CF + Fixed size window

Rohan

How

Approach → Why?

Interviewer → Brute force Approach

Can F

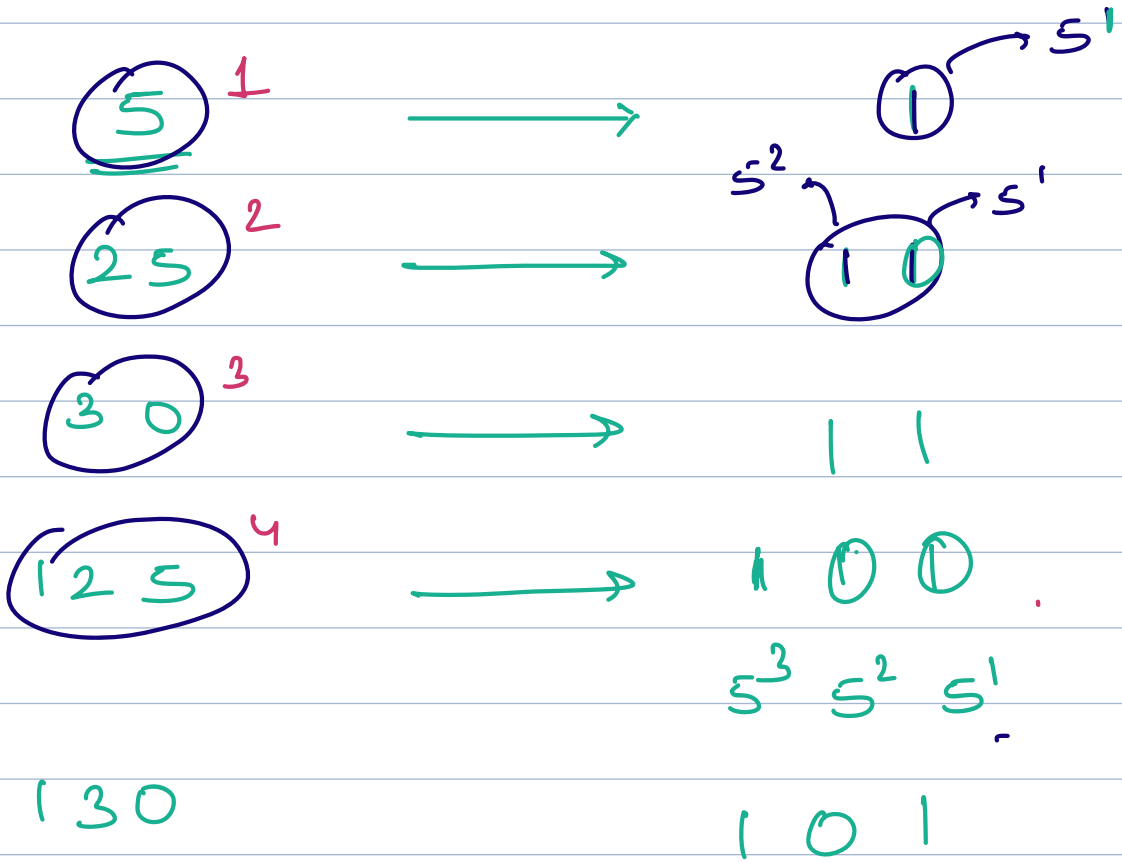
Prefix Approach

Search → HashMap = O(n)

Sort = O(n log n)

10^7

5	30
5	6 → 0
	1 → 1


5
1
25
2
30
3
125
4
1
5^1
5^2
5^1
1 0
1 1
1 0 0
5^3 5^2 5^1
130
1 0 1