Q Max subarray sum

Eg: $-3, 2, 4, -1, 3, -4, 3$ ans = 8

Brute: Check for all subarrays

TC: $O(n^2)$

Kadane's Algorithm.

Case 1 All elem ≥ 0

3	2	1	6

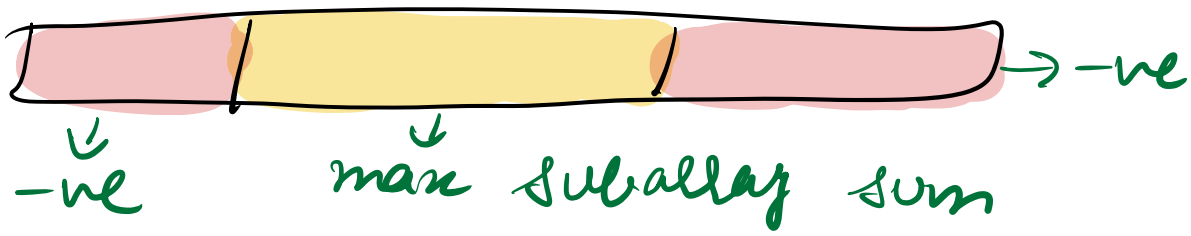
12

Case 2 All elem < 0

-8	-4	-2	-10

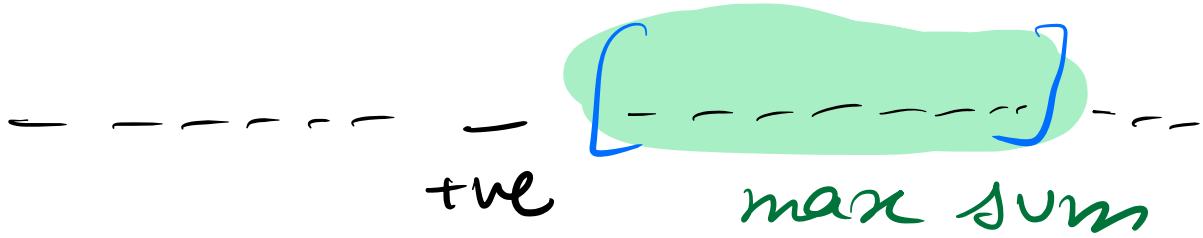
-2

Case 3

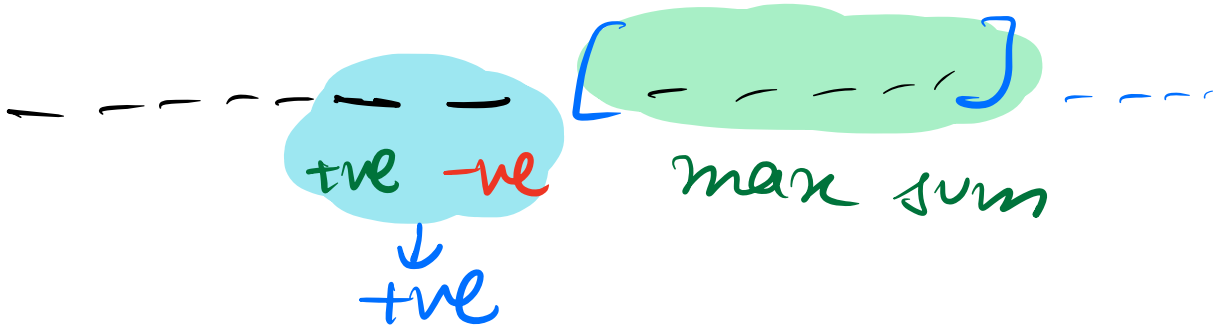


+ve	-ve	+ve	-ve
+10	-5	+7	-50

Case 4



Case 5



If sum > 0, ⇒ take

ar	5	6	7	-3	2	-10	-12	80
sum=0	5	11	18	15	17	7	~~-5~~ 0	80
ans = INT_MIN	5	11	18	18	18	18	18	80

	-2	3	4	-1	5	-10	7
0		3	7	6	11	1	8

Code

```
sum = 0
ans = INT_MIN
for ( i=0 ; i<n ; i++ ) {
    sum = sum + a[i]
    ans = max( ans, sum )
    if ( sum < 0 )
        sum = 0
}
return ans
```

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

	0	1	2	3	4	5	6	7	8	9
Q)	3	2	-1	5	6	8	2	3	2	6

Queries: 3

[1, 4]

[3, 6]

[1, 7]

Idea: prefix sum

pf[i]: ?

pf[0] = ?

```
for (i=1; i<n; i++) {
    pf[i] = pf[i-1] + a[i]
}
```

Answer for each query

```
for (i=0; i<Q; i++) {
    read (s, e)   // start & end
// sum[i:j] = ?
  if (s == 0)
    ans = pf[e]
  else
    ans = pf[e] - pf[s-1]
```

TC:

SC:

Q. Given N array elements = 0

For every query of the form index, val add val to all indexes [index : n-1]

Q = 4

idx	val	0	1	2	3	4	5	6
		0	0	0	0	0	0	0
2	4	0	0	4	4	4	4	4
3	-1	0	0	4	3	3	3	3
0	2	2	2	6	5	5	5	5
4	1	2	2	6	5	6	6	6

$\Rightarrow$

Brute: Use nested loops to add for each query. TC: $O(QN)$

Idea ar[s]

0	1	2	3	4
a_0	a_1	a_2	a_3	a_4
a_0	$a_0 + a_1$	$a_0 + a_1 + a_2$	$a_0 + a_1 + a_2 + a_3$	$a_0 + a_1 + a_2 + a_3 + a_4$

$Q = 4$

idx	val	0	1	2	3	4	5	6
		0	0	0	0	0	0	0
2	4	0	0	4	0	0	0	0
3	-1	0	0	4	-1	0	0	0
0	2	2	0	4	-1	0	0	0
4	1	2	0	4	-1	1	0	0
	pf	2	2	6	5	6	6	6

- For every query directly update array
- Now take prefix sum of array.

Code

```
for (i=0; i<Q; i++) {
    read (idx, val)
    ar[idx] += val
}
```

1 3 6

```
// Now take pref sum
for (i=1; i<n; i++) {
    ar[i] += ar[i-1]
}
```

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

Q3 Given N array elements = 0

For every query of the form s, e, val

add val to all indexes [s : e]

	0	1	2	3	4	5	6	7	8
Eg:	0	0	0	0	0	0	0	0	0

3, 6, 1

			1	1	1	1		
→			1	1	1	1	1	1
							-1	-1

	0	1	2	3	4	5	6	7	8
1, 5, 6		6	6	6	6	6			
→		6	6	6	6	6	6	6	6
							-6	-6	-6

Idea [s, e, val] is same as

1) [s : n-1] add val

2) [e+1 : n-1] add -val

```
for ( i=0 ; i<Q ; i++) {
    read (s, e, val)
    ar[s] += val
    if ( e != n-1)
        ar[e+1] -= val
}
```

Step 2 : Take prefix sum

1	4	3
0	5	-1
2	2	4
4	6	3

0	1	2	3	4	5	6	7
-1	3	4	-4	3	-3	1	-3
-1	2	6	2	5	2	3	0

TC: SC: Same as above

0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0
	+3				-3		
0	3	3	3	3	0	0	0

$[1, 4, 3] \Rightarrow [1, 7, 3] + [5, 7, -3]$

	3	2	1	5	1	3
lmax	3	3	3	5	5	5
rmax	5	5	5	5	3	3

$bf(i) = (bf(i-1), a(i))$ max

Previously studied

Leftmax & Rightmin

(from carry-fwd)

Requirements:

Leftmax & Rightmax

0 : i i, n-1

Q Rain water trapped

Given array of size N, ar[i] represents height of ith building

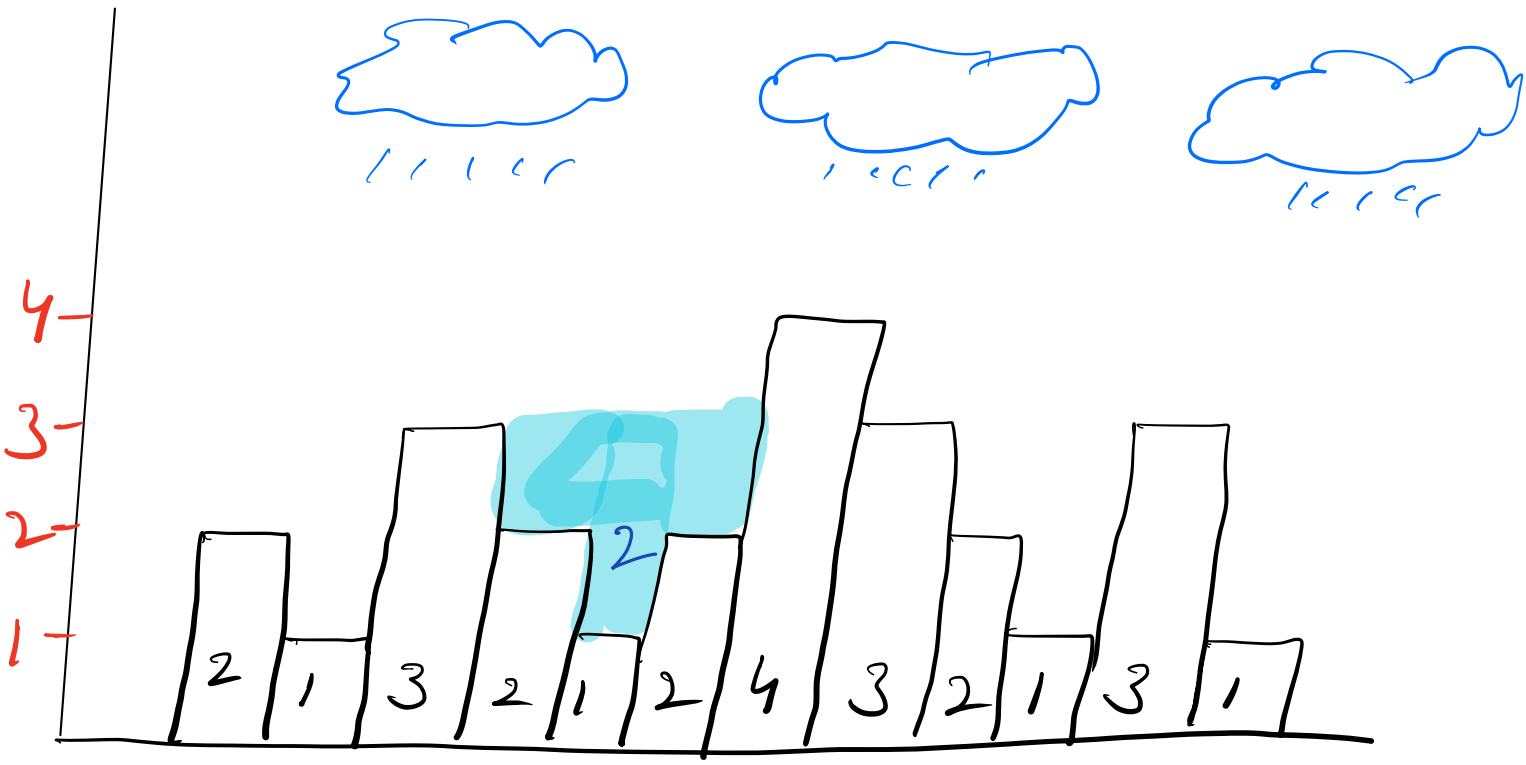
Assume that it rains (A LOT)

Return amount of water trapped.

$$lmax(i) = max(lmax(i-1), a(i))$$

$$rmax(i) = max(rmax(i+1), a(i))$$

Eg: { 2, 1, 3, 2, 1, 2, 4, 3, 2, 1, 3, 1 }



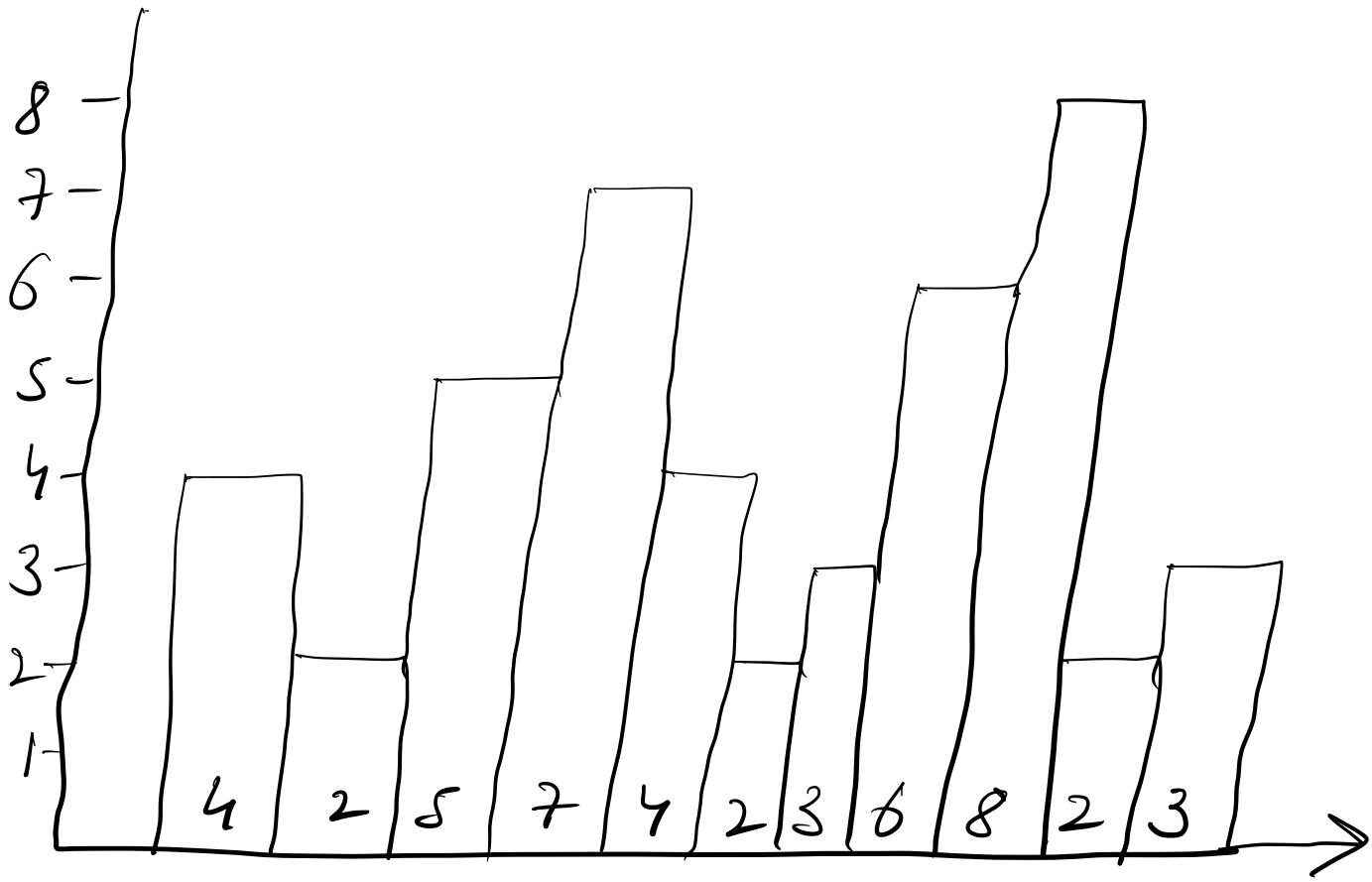
total amt =

idea Calc the amount of water trapped on top of each building

ans = min( )

left_support =

right_support =



L	lmax[i-1]	4	4	5	7	7	7	7	7	8	8	8
R	rmax[i+1]	8	8	8	8	8	8	8	8	8	3	3
sup	min(L,R)		4	4	5	7	7	7	7	3	3	
W			2	0	0	3	5	4	1	0	0	

$\Rightarrow$ 15

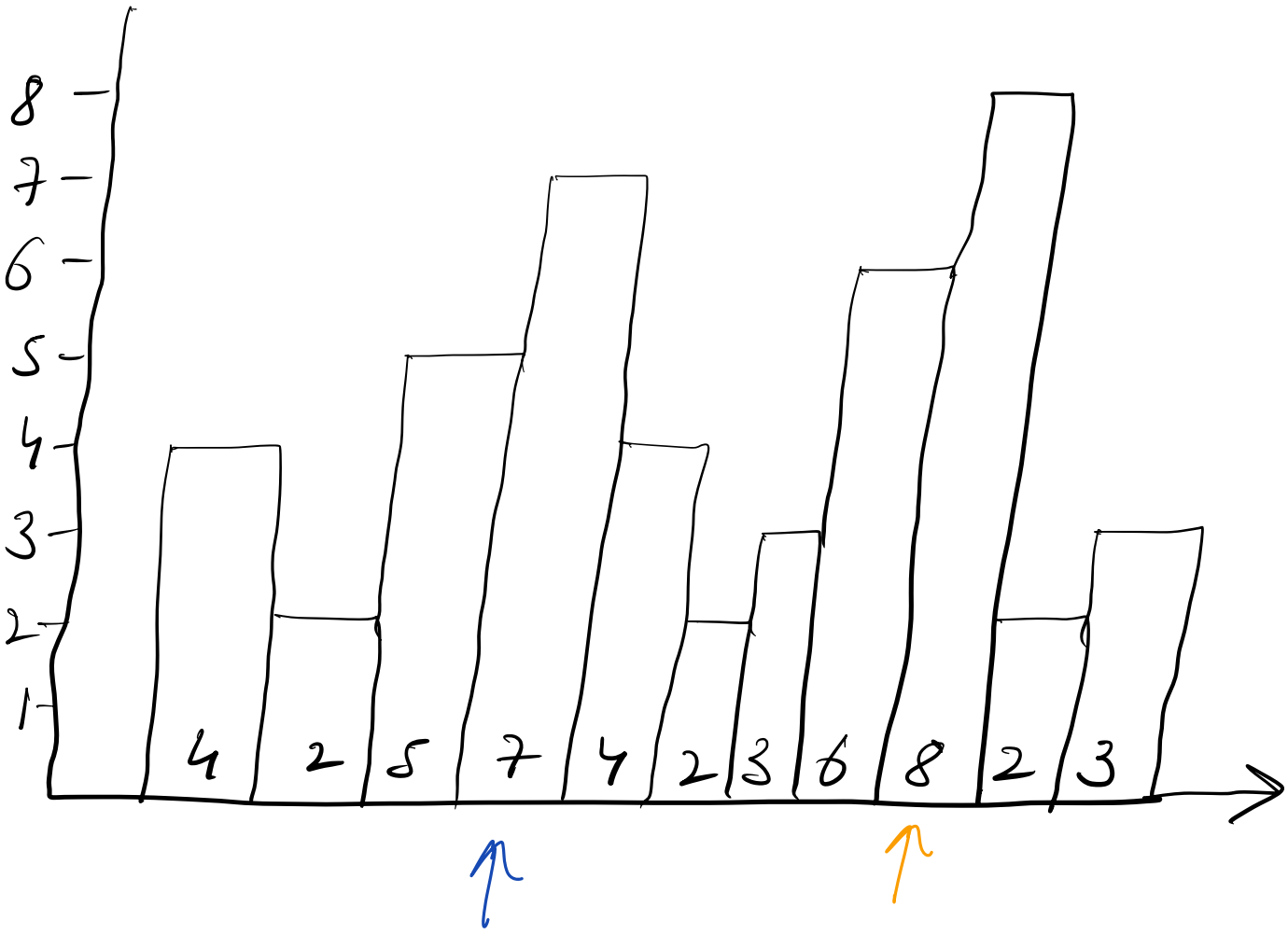
Code

because they will never have water

```
ans = 0
for (i=1 ; i<n-1 ; i++) {
    Lsup = lmax (i-1)
    Rsup = rmax (i+1)
    sup = min (Lsup, Rsup)
    W = sup - ar[i]
    if ( W>0 ) ans += W
}
```

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

How to do in $O(1)$ SC



move away from smaller height

```
i = 0                          j = n-1

ans = 0

lmax = A[0]                    rmax = A[n-1]

while (i<j) {
    if (lmax < rmax) {
        i++
        water = lmax - A[i]
        lmax = max(lmax, A[i])
    }
    else {
        j--
        water = rmax - A[j]
        rmax = max(rmax, A[j])
    }
    if (water > 0)
        ans += water
}

return ans
```