Today's Content

- prefix sum ✓
- problems based on prefix sum ✓
- LeftMax[], RightMax[] (TODO) ✓
- Water accumulated
- Max Subarray Problems

10

	0	1	2	3	4	5	6	7	8	9
ar[10] =	3	2	-1	5	6	8	2	3	2	6

Q : 3

1 4 : 12

3 6 : 21

1 7 : 25

Idea:

→ Calculate pf[i]

pf[i] = Sum of all elements [0 i]

pf[0] = ar[0]

for (i = 1 ; i < N ; i++) {

pf[i] = pf[i-1] + ar[i]

}

Else i = 0

TC: O(N)

→ Calculate sum for each Query

while (Q--) {

// Say input i, j

// We need to get sum of all [i-j]

(0 j) = (0, i-1) + (i, j)

pf[j] = pf[i-1] + sum(i-j)

if (i != 0)

sum(i-j) = pf[j] - pf[i-1]

else

sum(0-j) = pf[j]

}

TC: O(Q)

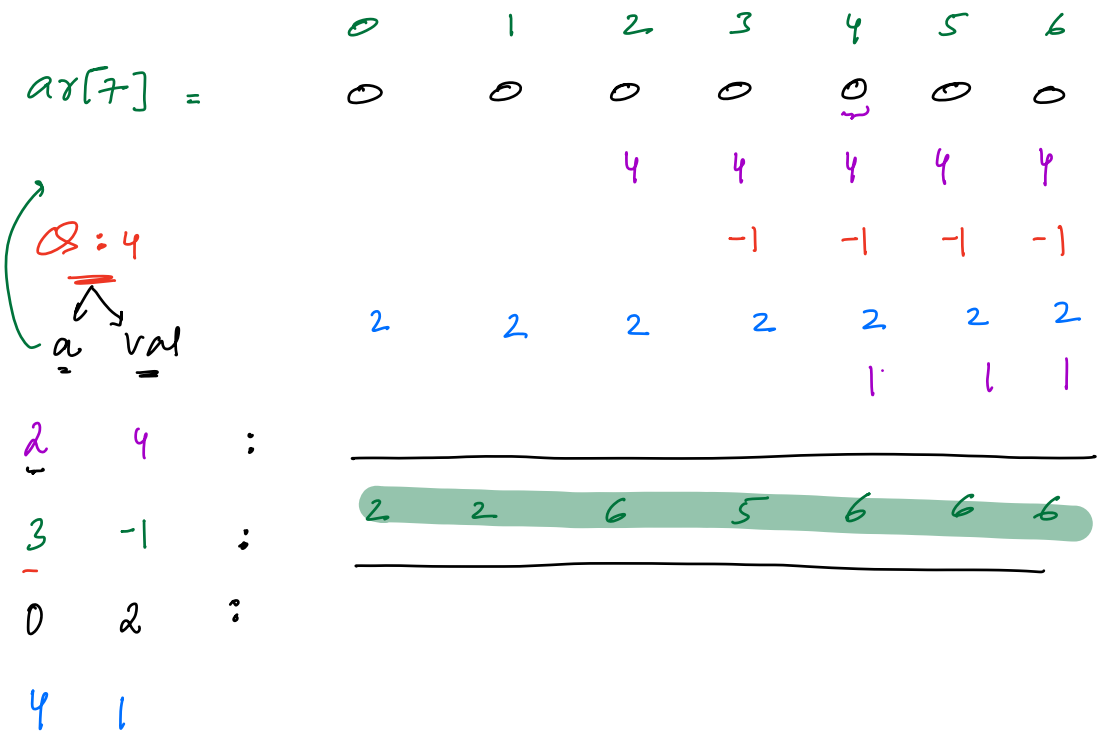
overall TC:

→ O(N + Q)

SC: → S O(N): new pf[]

↳ O(1): Changing input array.

28 Given N array elements = 0



Idea:

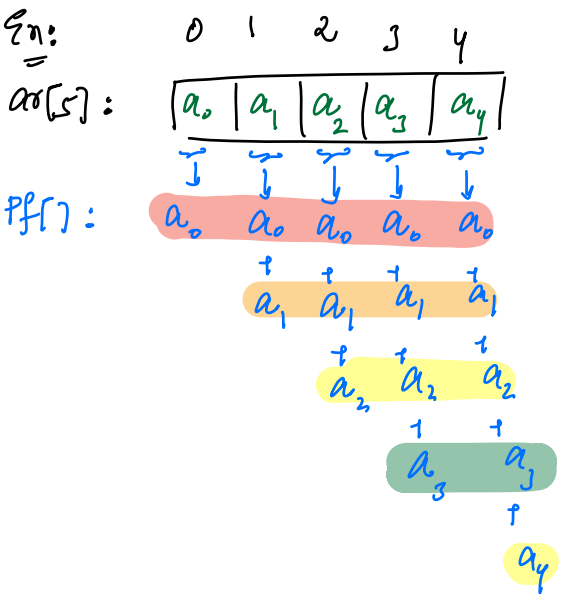
For every query Iterate & update array.

```
while( Q-- ){
    // Given index a & val V
    for( i = a ; i < N ; i = i+1 ){
        ar[i] = ar[i] + V
    }
}
```

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

Idea:

Eg:



Idea:

ar[7] =

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

8 queries

ind	val
2	4
3	-1
0	2
4	1

// apply pf on above :

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

Idea 2:

→ For every query directly update array

→ Once all updates are done apply pf[]

Pseudo code

```
while(Q--){
    // given index a & value v
    ar[a] += v
}
// Apply pf[] on given array.
// Update same array itself
// ar[0] → no change
for(i=1; i<N; i++){
    ar[i] = ar[i] + ar[i-1]
}
```

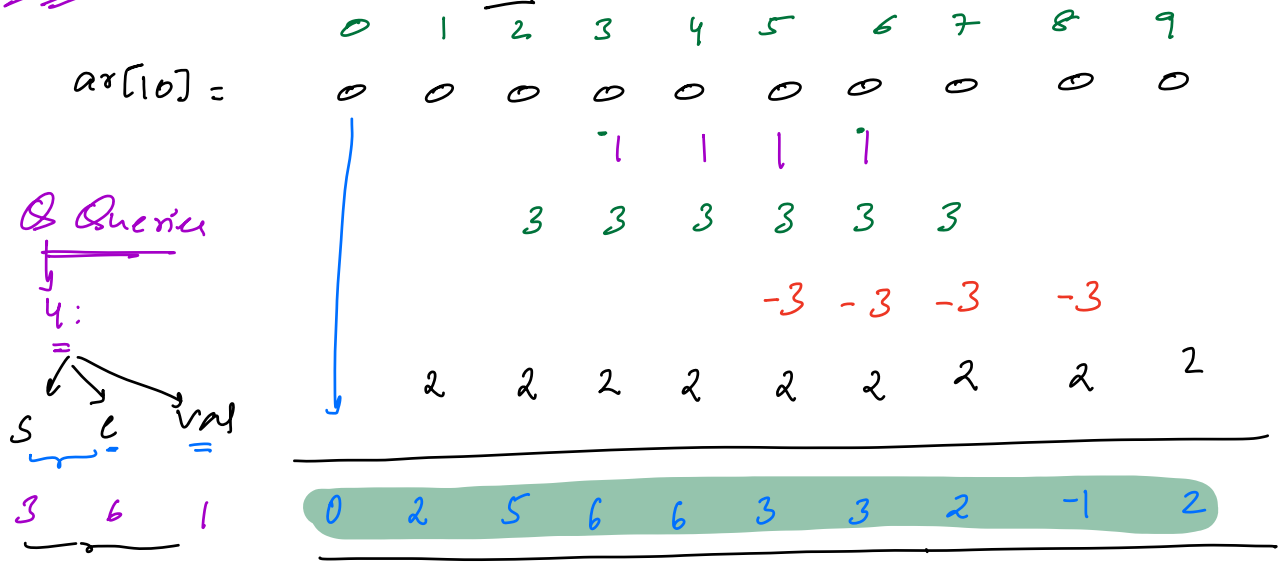
{ Because 2 queries can have same index }

TC: O(Q + N)

SC: O(1)

Beggars Outside Temple

google : Given N array elements = 0



Q Queries

4:

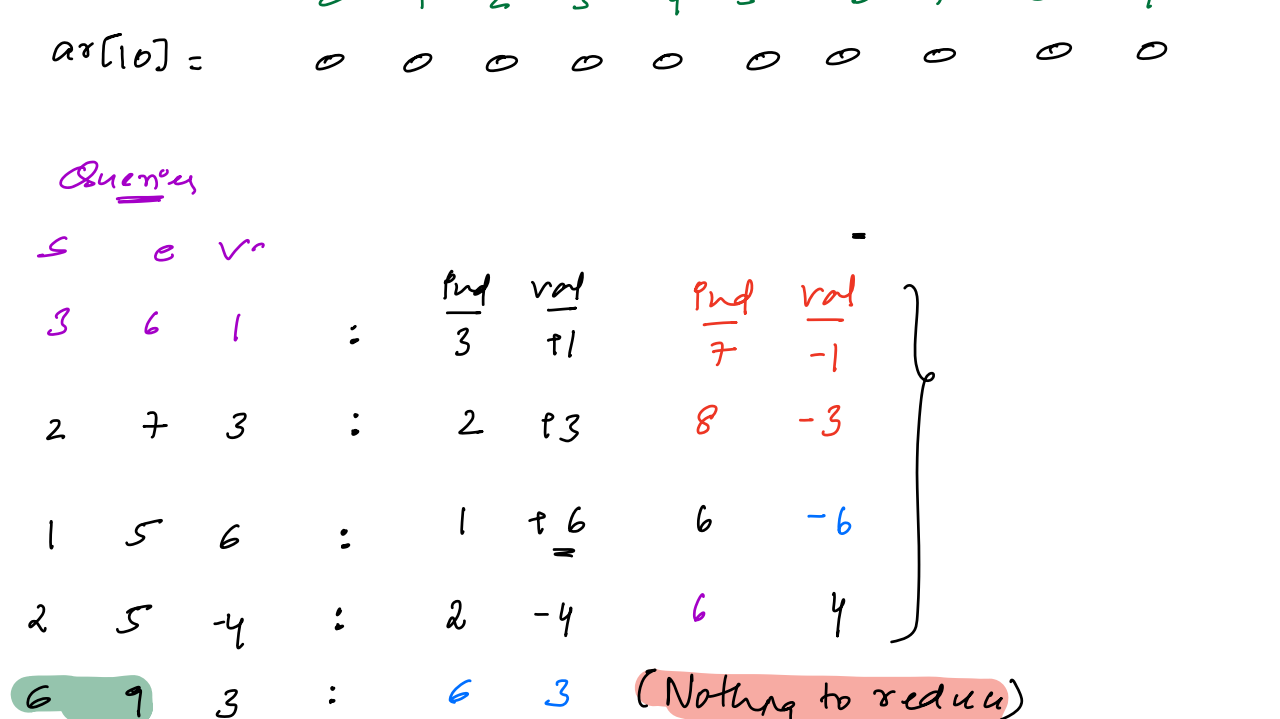
S e val

3 6 1

2 7 3

5 8 -3

1 9 2



Queries

S	e	Val		pnd	val	pnd	val
3	6	1	:	3	+1	7	-1
2	7	3	:	2	+3	8	-3
1	5	6	:	1	+6	6	-6
2	5	-4	:	2	-4	6	4
6	9	3	:	6	3	(Nothing to reduce)	

Index 0 1 2 ... s s+1 .. e e+1 -- N-1

v v -- v v --- v

-v ... -v

// generalize

Q: s e v : [s N-1] add (v) [e+1, N-1] add (-v)

// Pseudo Code

```
while (Q--) {
    // Given s, e, val
    ar[s] += val          // [s, N-1] add v
    (Edge Case)
    if (e+1 < N) {
        ar[e+1] -= val    // [e+1, N-1] add (-v)
    }
}
```

// apply pf() on array

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

	0	1	2	3	4	5	6	7
// ar[] :	0	0	0	0	0	0	0	0
		+3				-3		
		3	3	3	3	0	0	0

1 4 : 3 } → [1 + : 3] , [5 + : -3]

4Q

	0	1	2	3	4	5
ar[6] = {	1	-6	3	2	8	7 }
Pfx[6] = {	1	1	3	3	8	8 }

Pfx[i] = max of all Elements from [0 i] (Prefix)

↳ TODO : TC : O(N)

5Q

	0	1	2	3	4	5	6
ar[7] = {	3	10	6	7	0	2	-1 }
Sfx[7] = {	10	10	7	7	2	2	-1 }

N = 7

Sfx[i] = max of all Elements from [i N-1] (Suffix)

↳ TODO : TC : O(N)

Sfx[i] = sum of all [i, N-1]

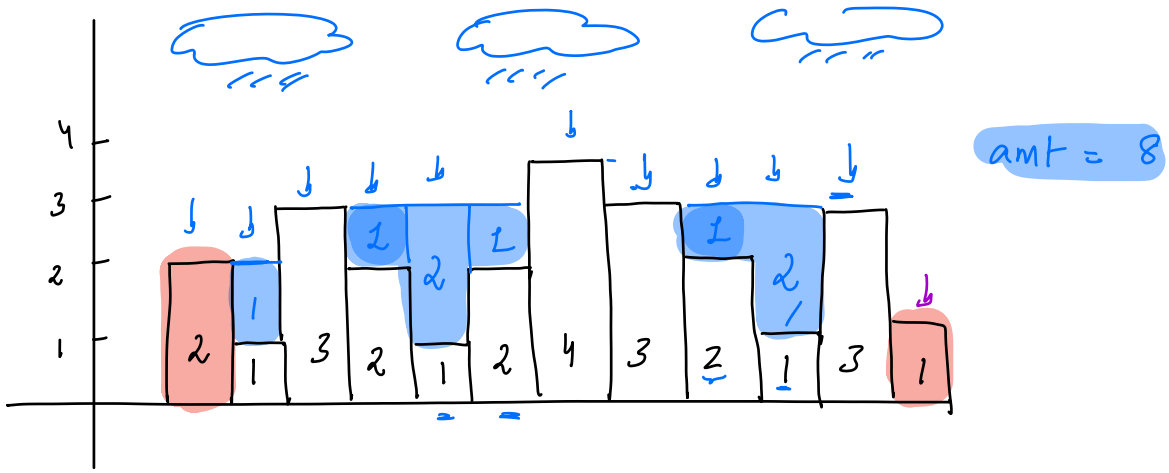
10:40 break

6) Rain water trapped ?

Given N array elements, where ar[i] represents height of the building

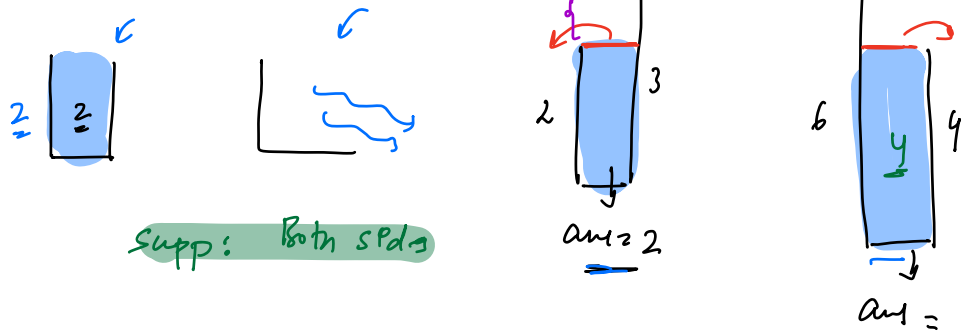
Return amount of water trapped on all buildings

Ex: ar[] = { 2, 1, 3, 2, 1, 2, 4, 3, 2, 1, 3, 1 }



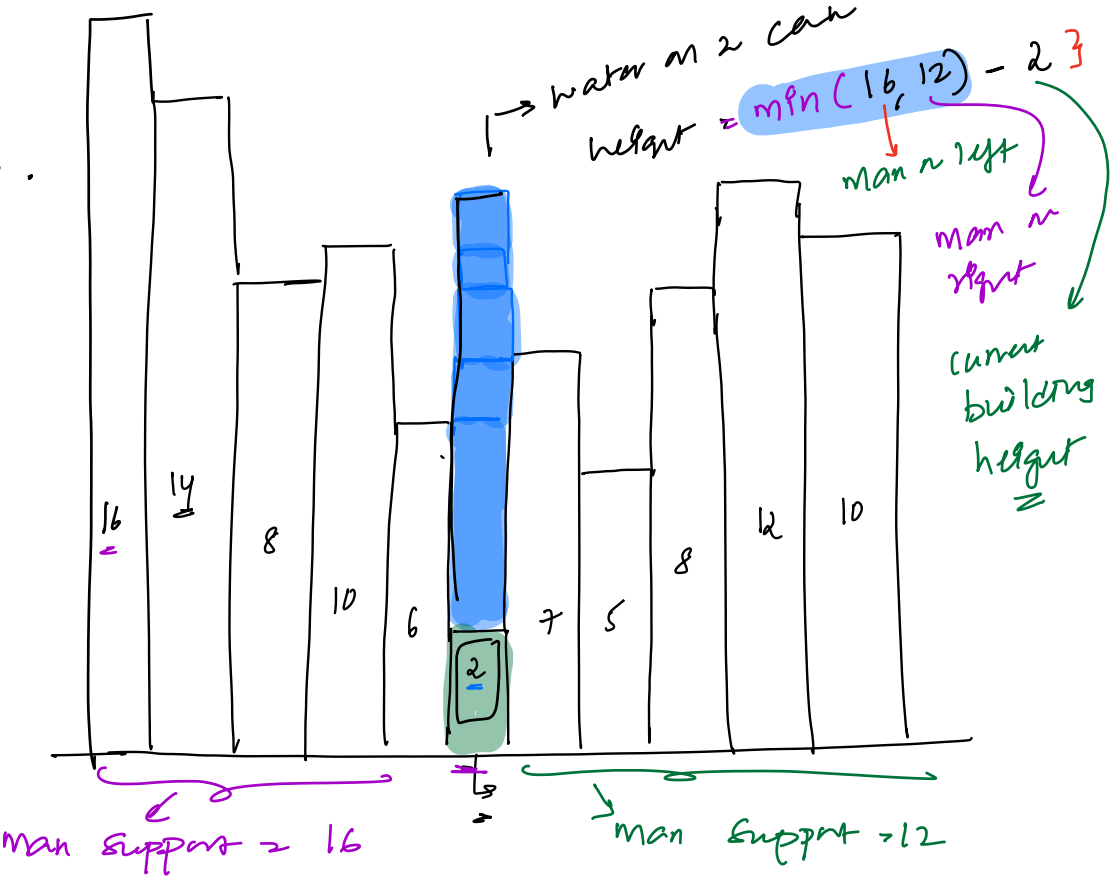
Idea: Calculate sum of water accumulated on each building

Ex:

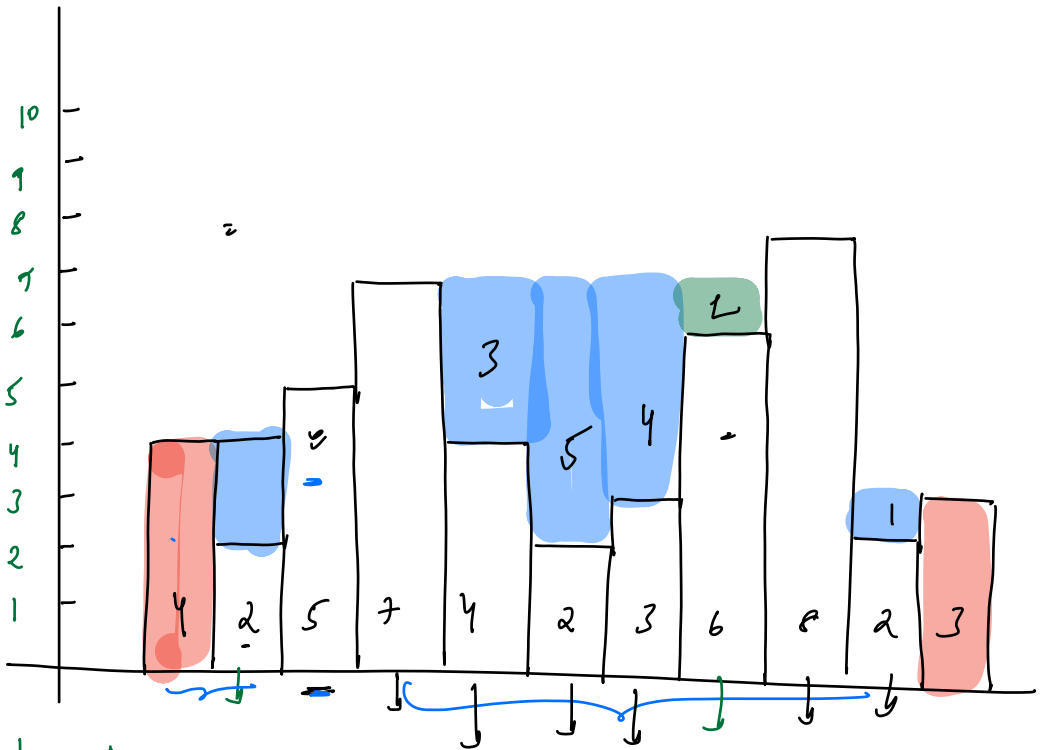


Supp: Both sides

Ex.


water on 2 can
height = min(16, 12) - 2
max n left
max n right
current building height
16
14
8
10
6
2
7
5
8
12
10
max support = 16
max support = 12

Ex:



L :	4	4	5	7	7	7	7	7	8
R :	8	8	8	8	8	8	8	3	3
H: min(L,R) :	4	4	5	7	7	7	7	3	3
V: H - arr[i] :	2	-1	-2	3	5	4	1	-5	1

= 1.6

no water

// Pseudo Code

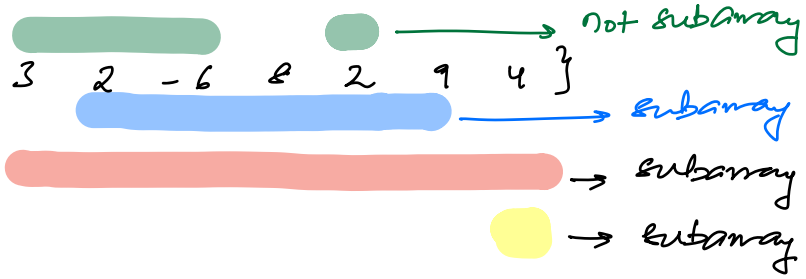
TC : O(N) SC : O(N)

```
ans = 0;                    {Because corner building no water}
i = 1; i < (N-1); i++) {
    // Calculate amt of water
       accumulate on i^th building
    { Lman = {man of all [0, (i-1)] → Pfx[i-1]   } optimize
      Rman = {man of all [i+1, N-1] → Sfx[i+1]   }
    H level = min(Lman, Rman)
    W = max(H level - ar[i], 0)
    ans = ans + W
}
```

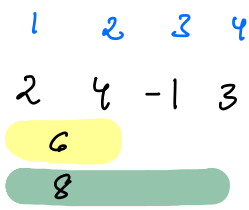
78) Given N Array Elements, Calculate Max Subarray Sum

Continous part of an array

Ex: ar[7] = { 3 2 -6 8 2 9 4 }



Ex: ar[7] = { -3 2 4 -1 3 -4 3 } → ans = 8



Approach 1 :

for every subarray

Iterate & get

sum : O(N^3)

Approach 2 :

get all subarray sums using pf[]

TC : O(N2)

SC : O(N)

get all subarray sums using carry forward

TC : O(N^2)

SC : O(1)

→ (Kadane's Algo)

Sc 1: All your elements >= 0 : Entire array is ans

3	2	1	6

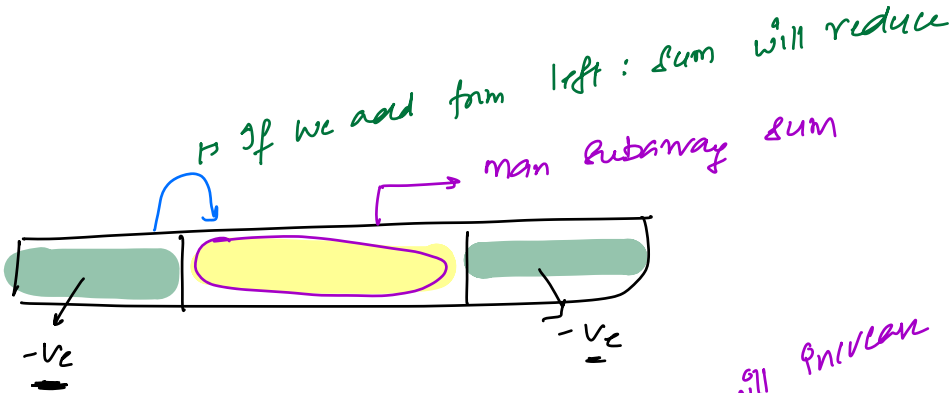
→ 12

Sc 2: All your elements < 0 : Max of Entire array

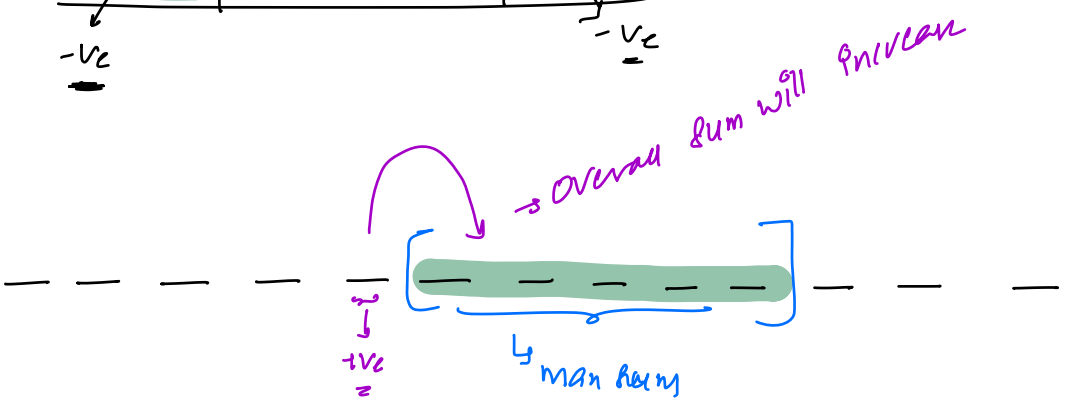
-8	-4	-2	-10

→ -2

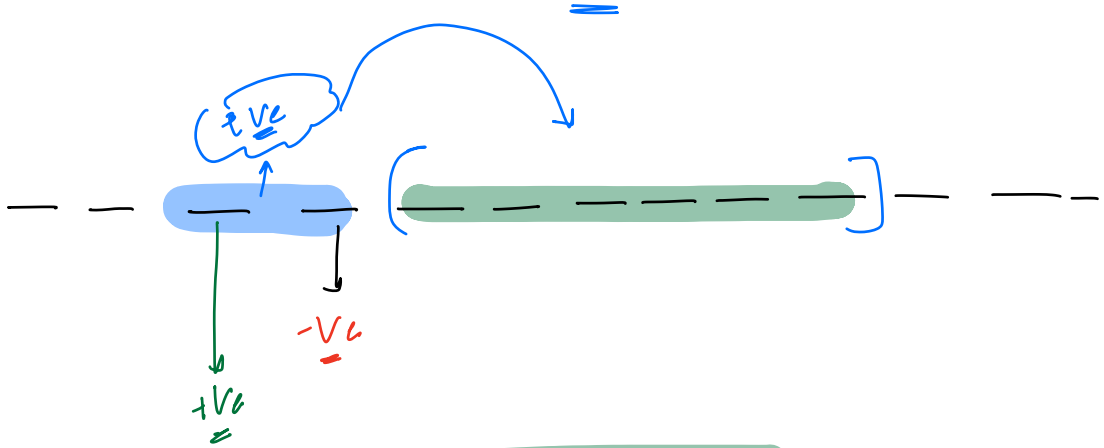
Sc 3:



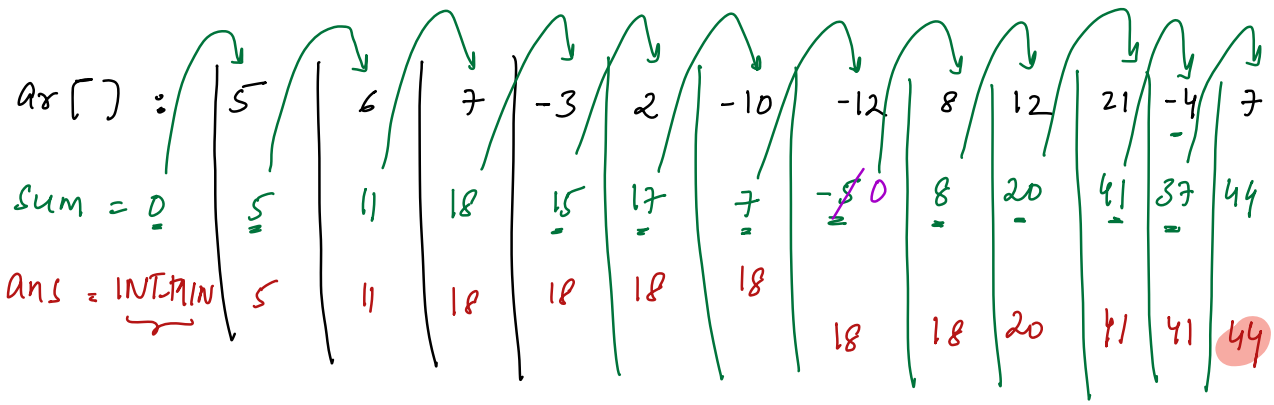
Sc 4:



Sc 5:



If sum > 0 only then we will carry sum




```
s = 0
a = (INT_MIN) ∩ ar[0]

for(i=0; i<N; i++){
    s = s + ar[i]
    a = max(a, s)
    if(s<0){
        s = 0
    }
}
return a;
```

TC = O(N)

SC = O(1)

Assignments : Discussed

Homework : Slight

↳ flip → (Max Subarray Sum)

→ Wed – (Arrays - 2)

Fri – (1 Holiday)

→ (Next Mon) - (Arrays-3)