Ques Maxm Subarray Sum

Given an integer array A, find the maximum subarray sum out of all the subarrays.

arr[] = {-2, 3, 4, -1, 5, -10, 7}
↳ 11 (Max Sum)

len → 7.

$\frac{7(7+1)}{2}$ => $\frac{7 \times 4}{2}$ => 28 subarray.

arr[] = {-3, 4, 6, 8, -10, 2, 7}
↳ 18 Ans.

e.g 1) +ve +ve +ve +ve → Sum of all elements

A[ ] = { 4, 5, 2, 1, 6 }

↳ 18 Ans.

e.g 2) -ve -ve -ve -ve -ve → Max -ve element

A[ ] = { -4, -3, -6, -9, -2 }

↳ Ans -2.

Brute force

arr[] = {-3, 4, 6, 8, -10, 2, 7 3}

No of possible subarrays: `N * (N + 1) / 2`

Iterate over all subarrays, calculate sum and maintain the maximum sum.

```
ans = A[0];
for (i = 0; i < N; i++) { // start to N
    for (j = i; j < N; j++) { // end
        for (k = i; k <= j; k++) {
            sum += A[k];
        }
        ans = Math.max(ans, sum);
        sum = 0; // Reset sum for the next iteration
    }
}
return ans;
```

start points

end points

start → end.

T.C -> O(n^3)

S.C -> O(1).

Idea 2 :- use Prefix Sum to find sum of subarray from i-j.

T.C -> O(n^2)

S.C -> O(n)

Idea 3 :- use carry forward.

↳ revise carry forward from lecture.

```
ans = A[0]
for(i = 0 to N - 1){ //start to N
    sum = 0
    for(j = i to N - 1){ //end
        sum += A[k]
        ans = max(ans, sum)
    }
}
return ans;
```

T.C → O(n²)

S.C → O(1)

Idea 4:- Kadane's Algorithm

Case - 1 + + + + + +

↳ Sum of all elements

Case - 2 -ve -ve -ve -ve -ve

↳ Max of array.

Case - 3 - - - + + + - - - -

↳ Max Subarray Sum

Case - 4

+10 -8

-10 +8

-ve -ve -ve +ve -ve +ve +ve +ve +ve -ve -ve -ve

+ve

x

arr[] = [-2, 3, 4, -1, 5, -10, 7]

csum =	~~-2~~ 0	3	7	6	11	1	8
maxsum =	-2	3	7	7	11	11	11

arr[] = [-20, 10, -20, 12, 6, 5, -3, 8, -2]

csum =	~~-20~~ 0	10	~~-10~~ 0	12	18	23	20	28	26
maxsum =	-20	10	10	12	18	23	23	28	28

[-10, -3, -1, -2]

csum =	~~-10~~ 0	~~-3~~ 0	~~-1~~ 0	-2
msum =	-10	-3	-1	-1

msum = $-\infty$

csum = 0

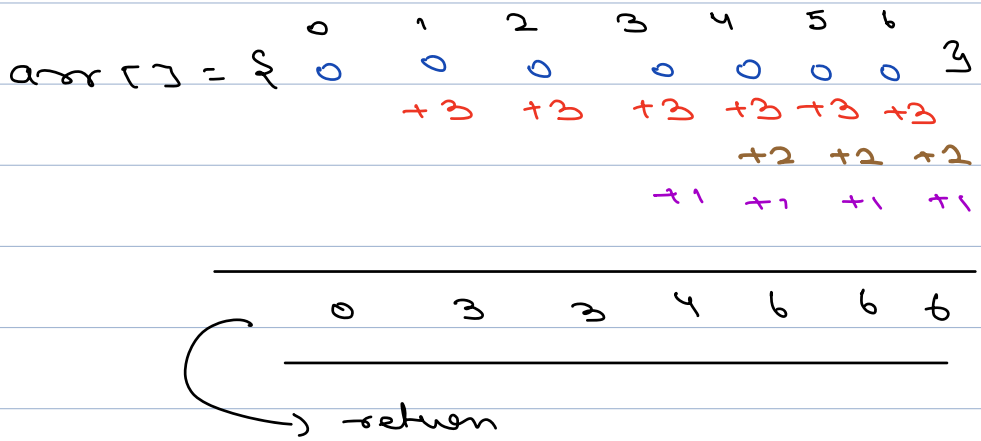
T.C → O(N), S.C → O(1)

```
for (i=0; i<n; i++) {
    csum += arr[i];
    if (csum > maxsum) {
        maxsum = csum;
    }
    if (csum < 0) {
        csum = 0;
    }
}
return maxsum;
```

Ques.

Given an integer array A where every element is 0, return the final array after performing multiple queries

Query (i, x): Add x to all the numbers from index i to N-1



Queries

idx	val
1	3
4	2
3	1

Brute force

for every Query, iterate from idx till end & add the value.

T.C → O(Q*N)

S.C → O(1)

Optimized Soln :-

	a_0	a_1	a_2	a_3	a_4
pf →	a_0	$a_0 + a_1$	$a_0 + a_1 + a_2$	$a_0 + a_1 + a_2 + a_3$	$a_0 + a_1 + a_2 + a_3 + a_4$

arr[] = { 0 0 0 0 0 0 3 }

0 1 2 3 4 5 6

+3 +5 +1 +2

Queries

idx	val
1	3 ✓
4	2 ✓
3	1 ✓
-1	2

```
for ( i = 0; i < Q.len; i++ ) {        → O(Q)
    idx ✓
    val ✓
    arr[idx] += val;
}
```

3

```
for(i=1; i<n; i++){ → O(N)
    arr[i] = arr[i-1] + arr[i]
}
```

3

T.C → O(Q+N).

S.C → O(1).

Ques

Given an integer array A such that all the elements in the array are 0. Return the final array after performing multiple queries

`Query: (i, j, x)`: Add x to all the elements from index i to j

arr[] =

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

Query

i	j	val
1	3	2
2	5	3
5	6	-1

e.g 2)

arr[8] =

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

-1	2	6	2	5	2	3	0

i	j	val	
1	4	3	✓
0	5	-1	✓
2	2	4	✓
4	6	3	✓

arr[8] =

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

-1	2	6	2	5	2	3	0

i	j	val	
1	4	3	✓
0	5	-1	✓
2	2	4	✓
4	6	3	✓
i	j	val	
2	7	3	

```
arr[i]+ = val ;
arr[j+1]- = val;
```

```
for (k=0; k<Q; k++) {
    // i, j, val
    arr[ij] += val;
    if(j != n-1){
        arr[i+1][j] -= val;
    }
}
```

T.C → O(Q)

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

T.C → O(N)

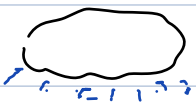
T.C → O(Q+N)

S.C → O(1).

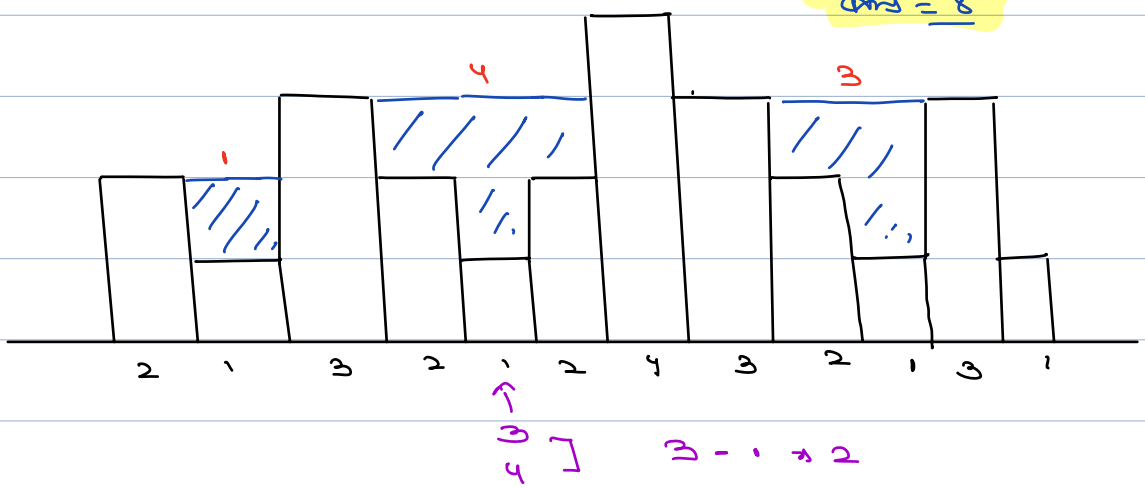
Ques **Rain water trapping** .

Given N buildings with height of each building, find the rain water trapped between the buildings.

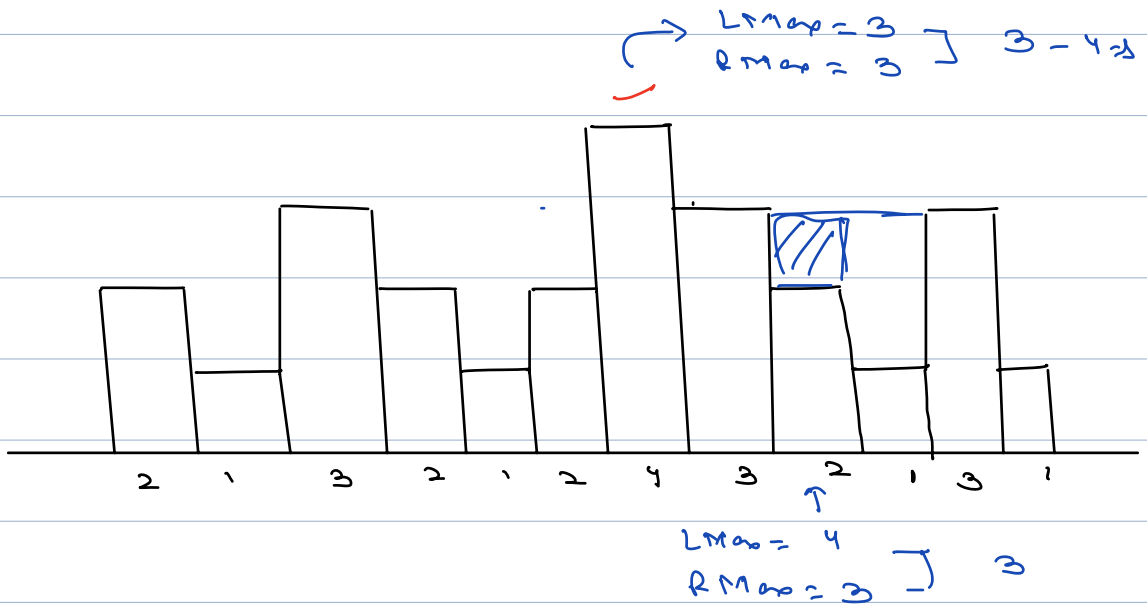
arr[] = 2, 1, 3, 2, 1, 2, 4, 3, 2, 1, 3, 1



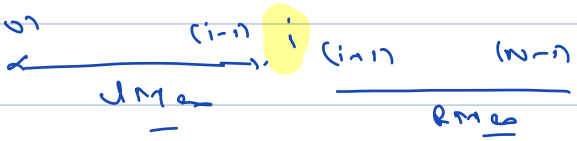
Ans = 8



Observations :- water at curr building = min(lMax, rMax) - h[i];



water at curr building = min(lMax, rMax) - h[i];



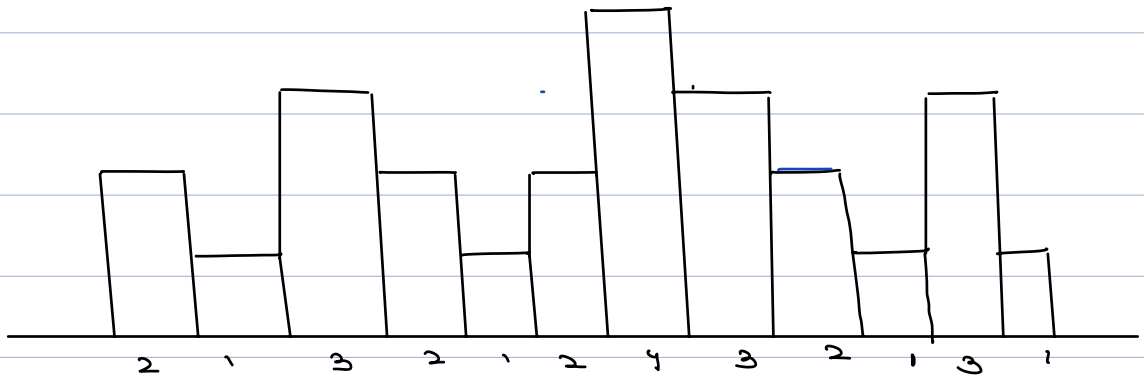
Brute force :-

```
ans = 0;                                          → O(n)
for (i = 1; i < n-1; i++) {
    maxh = maxHeight (0 to i-1);                  → O(n)
    maxR = maxRight (i+1 to n-1);                 → O(n)
    water = min(maxh, maxR) - arr[i];
    if (water > 0) {
        ans += water;
    }
}
```

3

$n(n+m) \Rightarrow O(n^2) \rightarrow$ T.C.

S.C $\Rightarrow O(1)$.



pfMax =	2	2	3	3	3	3	4	4	4	4	4	4
sfMax =	4	4	4	4	4	4	4	3	3	3	3	1
water =	-	1	0	1								-

water at curr building = $\min(lMax, rMax) - h[i]$;

pfMax[i] = Max of (0 to i)

sfMax[i] = Max of (i+1 to n-1)

```
pfMax[0] = arr[0];
for (i=1; i<n; i++) { → O(n)
    pfMax[i] = Max(pfMax[i-1], arr[i])
}
```

```
sfMax[n-1] = arr[n-1];
for(i=n-2; i>=0; i--) { → O(n)
    sfMax[i] = max(sfMax[i+1], arr[i])
}

for(i=1; i<n-1; i++) { → O(n)
    maxL = pfMax[i-1];
    maxR = sfMax[i+1];
    water = min(maxL, maxR) - arr[i];
    if(water > 0){
        ans += water;
    }
}
```

T.C → O(n)

S.C → O(N)

↓

One more soln → S.C → O(1).