Welcome to Advance Module. !!

Don't wait for the right time.

Create it.

1-D Arrays.

1. prefix sum
2. Max subarray Sum.
3. Rain Water Trapped

1. pSum[]
2. Carry forward.
3. sliding window.
4. contribution technique.

Answer. To: Jitender Punia

Question. To: Everyone or Question Stack.

Q Initially all elements of an arr[N] are 0. Then you are given Q queries. Every query contains i-idx & value. Increment elements from ith 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 → 3

idx	val
3	4
1	3
4	-2

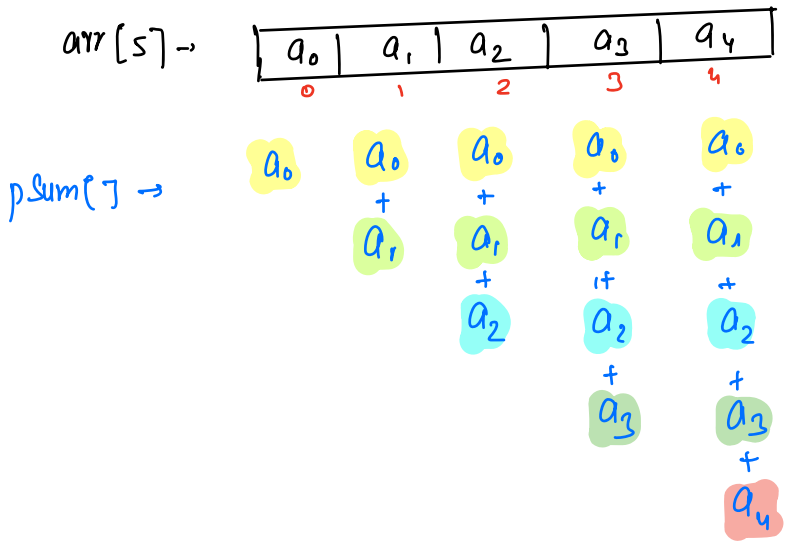
			+4	+4	+4	+4
	+3	+3	+3	+3	+3	+3
				-2	-2	-2
0	3	3	7	5	5	5

idea → For every query, iterate on array and update the elements.

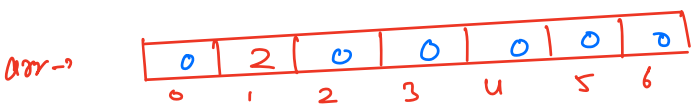
```
// take input of Queries
for( i = 1 ; i <= Q ; i++){
    // take input of idx and val
    for( j = idx ; j < N ; j++){
        arr[j] += val ;
    }
}
```

T.C → O(Q*N)
S.C → O(1)

idea-2 → use prefix Sum.



What if initial elements are not 0.



idx → 3, val → 4

[0 0 0 4 4 4 4]

fp arr → [0, 2, 0, 4, 4, 4, 4]

#code

A[][]

idx	val
3	4
4	-2
1	3
0	2

```
//take input of no. of Queries
for( i=0; i < A.length; i++){
    idx = A[i][0], val = A[i][1]
    arr[idx] += val;
}
//2. Apply psum technique on arr[]
for( i=1; i < N; i++){
    arr[i] = arr[i-1] + arr[i]
}
return arr[];
```

T.C → O(N+Q)
S.C → O(1)

Q) Initially all elements of arr[] are 0. Given Q queries, Every query contains [s, e, val]. Increment elements from s to e by val.
Return the final state of arr[].

Google

arr[10]: [0 0 0 0 0 0 0 0 0 0]
0 1 2 3 4 5 6 7 8 9

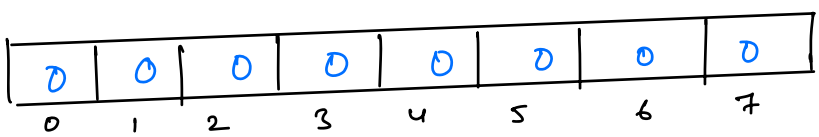
Queries →3

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

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

0 4 1 4 4 4 4 1 4 4

Quiz →



Q.

s	e	val.
1	4	3
0	5	-1
2	2	4
4	6	3

+3 +3 +3 +3
-1 -1 -1 -1 -1 -1
+4
+3 +3 +3

-1 2 6 2 5 2 3 0

arr[10]: [0 4 -3 3 0 0 0 -3 3 0]
0 1 2 3 4 5 6 7 8 9

Queries → 3

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

|| prefix sum

[0 4 1 4 4 4 4 1 4 4]

#code

```
//take input of no. of Queries
for( i=0; i < A.length; i++){
        s = A[i][0],   e = A[i][1] , val = A[i][2]
        arr[s] += val;
        if(e+1 < N){ arr[e+1] -= val; }
}
//2. Apply psum technique on arr[]
for( i=1; i < N; i++){
        arr[i] = arr[i-1] + arr[i]
}
return arr[];
```

T.C → O(N+Q)
S.C → O(1)

Q) Given N array elements. Calculate maximum subarray sum.

⇓

contiguous part of an array.

arr → [-3 | 2 4 -1 3 | -4 3]

0 1 2 3 4 5 6

[ans = 8]

idea-1 → Consider all subarrays and for every subarray iterate and calculate the sum.

```
maxSum = -∞
for( i = 0; i < N; i++){
    for( j = i ; j < N; j++){
        //subarray from i to j
        sum = 0
        for( k = i; k <= j; k++){
            sum += arr[k];
        }
        maxSum = Max( maxSum, sum);
    }
}
```

T.C → O(N³)
S.C → O(1)

O(N²)

By using prefixSum or carry forward.

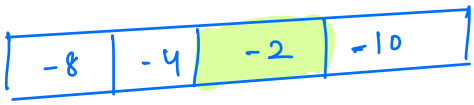
Case-1. All elements are positive.

Entire array will be our ans.

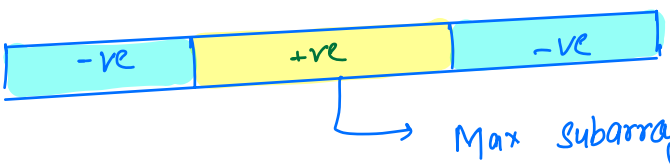
ans = 15.

Case-2. All elements are negative.

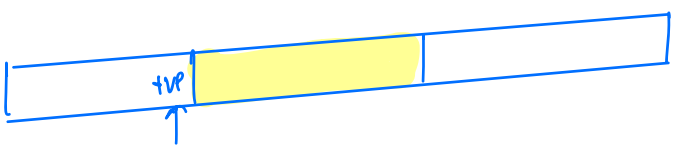
Max of entire array will be our ans.

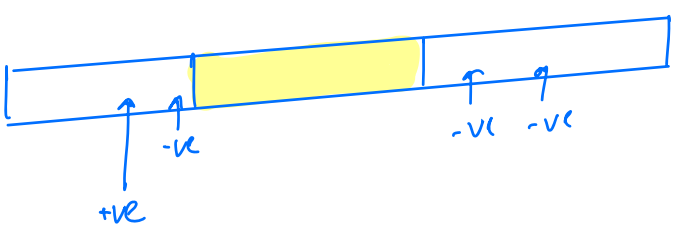


Case-3.



Max subarray sum.





[If sum > 0, only then carry forward the sum and updak your ans.]

arr[] → [5 6 7 -3 2 -10 -12 8 12 -4 7 -2]

sum = 0	5	11	18	15	17	7	~~-10~~	8	20	16	23	21
ans → ~~-∞~~	5	11	18						~~20~~		23	

Kadane's Algorithm

code →

```
ans = INT_MIN ,   sum = 0

for( i=0; i<N; i++){
        sum += arr[i]
        ans = Max( ans, sum)
        if (sum < 0){
            sum = 0 ; //resetting
        }
}
return ans;
```

T.C → O(N)
S.C → O(1)

arr →

-7	-15	-3	-9

sum = 0	~~-7~~ 0	~~-15~~ 0	~~-3~~ 0	-9
ans → ~~-∞~~	~~-7~~	-3		

Q: How to find si and ei of the subarray with maximum sum?

```
ans = INT_MIN , sum = 0 , res[2], si = 0;

for( i=0; i<N; i++){
        sum += arr[i]
        if( sum > ans ){
            ans = sum
            res[0] = si , res[1] = i
        }

        if (sum < 0){
            sum = 0 ; //resetting
            si = i+1
        }
}

return res[];
```

arr[]	5	6	7	-3	2	-10	-12	8	12	-4	7	-2
	0	1	2	3	4	5	6	7	8	9	10	11
sum=0	5	11	18	15	17	7	-5	8	20	16	23	21
ans=-∞	5	11	18						20		23	

si = 0 7

	7	10
res	0	2
	0	1

Flip. (Additional problem)

Q) Given arr[N]. Create lmax[N], rmax[N]

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

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

arr[] → [4 (0) 2 (1) 9 (2) 6 (3) 11 (4) 5 (5)]

lmax[] → [4 4 9 9 11 11]

code →

```
lmax[0] = arr[0];
for(i = 1; i < N; i++){
    lmax[i] = Max(lmax[i-1], arr[i]);
}

rmax[N-1] = arr[N-1];
for(i = N-2; i >= 0; i--){
    rmax[i] = Max(rmax[i+1], arr[i]);
}
```

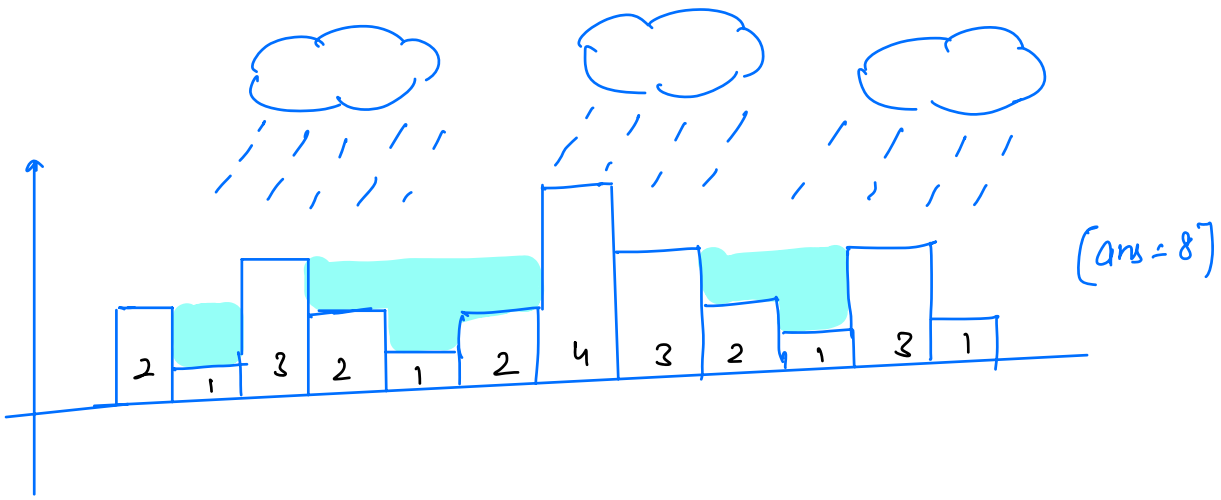
T.C → O(N)

Rain water Trapped

Given arr[N], where arr[i] → ht. of building.

Return amount of water trapped on all the buildings.

arr[2 1 3 2 1 2 4 3 2 1 3 1]

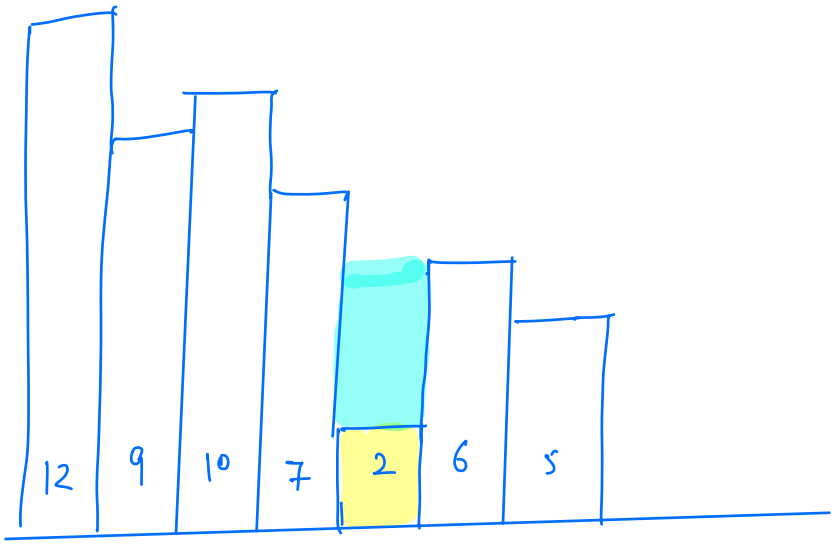


(ans = 8)

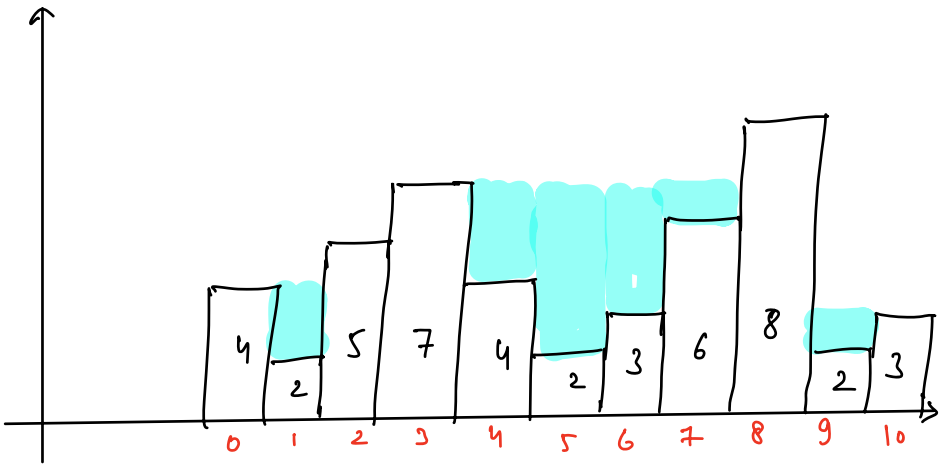
Idea → Water accumulated at every building.

left and right boundaries are required to accumulate the water.

observation :



Water accumulated = Min(leftmax, rightmax) - ht of the building.



lmax[7→ [4 4 5 7 7 7 7 7 8 8 8]

rmax[7→ [8 8 8 8 8 8 8 8 8 3 3]

Min(lmax[i], rmax[i]) → [4 4 5 7 7 7 7 7 8 3 3]

water accumulated → [0 2 0 0 3 5 4 1 0 1 0]

= 16.

Code :-

```
lmax[N], rmax[N];
lmax[0] = arr[0];
for(i=1; i<N; i++){
    lmax[i] = Max(lmax[i-1], arr[i]);
}
rmax[N-1] = arr[N-1];
for(i=N-2; i>=0; i--){
    rmax[i] = Max(rmax[i+1], arr[i]);
}
ans = 0
for(i=0; i<N; i++){
    ans += (Min(lmax[i], rmax[i]) - arr[i]);
}
return ans;
```

T.C → O(N)
S.C → O(N)

x x