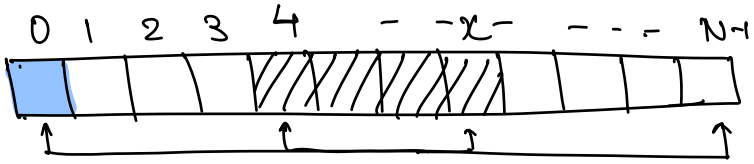
Q.1 Max Subarray sum

GS | Linkedin | Paytm | Amazon | Ola | Walmart - - - -

↓

Contigous part of Array

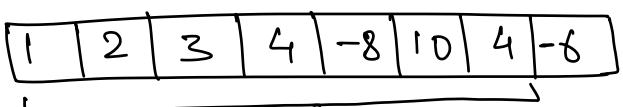
⇒ Total # of subarrays :- $\frac{N(N+1)}{2}$



Start	end		# of subarray
0	[0, N-1]	⇒	N
1	[1, N-1]	⇒	N-1
2	[2, N-1]	⇒	N-2
⋮	⋮		
N-1	[N-1, N-1]	⇒	1

$$\frac{}{N+(N-1)+ \cdots 2+1}$$

$$\Rightarrow \frac{N(N+1)}{2}$$



16

Brute force

```
ans = -∞ / INT_MIN
for(s = 0; s < N; s++) {
    for(e = s; e < N; e++){
        sum = 0
        for(i = s; i <= e; i++){
            sum += a[i];
        }
        ans = max(ans, sum);
    }
}
```

TC: $O(N^3)$

SC: $O(1)$

⇒ Better Approach?

Range ⇒⇒ Prefix Sum.

$Sum[s,e] = PS[e] - PS[s-1]$

TC: $O(N^2)$

SC: $O(N)$

↳ Prefix Sum Array

Carry forward

Sum[0-4] = Sum[0-3] + a[4]

s	e	Sum
0	0	A[0]
0	1	A[0] + A[1]
0	2	A[0] + A[1] + A[2]
0	3	A[0] + A[1] + A[2] + A[3]

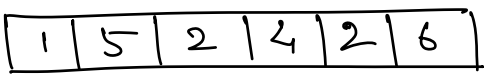
```
Ans = -∞
for(i = 0; i < N; i++){ // s
    sum = 0
    for(j = i; j < N; j++){ // e
        sum += A[j]
        ans = max(ans, sum);
    }
}
```

TC: $O(N^2)$

SC: $O(1)$

Observations

1) If all the array elements are +ve.



$$sum = 1+5+2+4+2+6$$

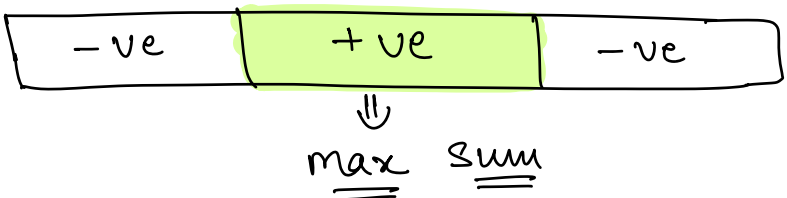
⇒ Return array_sum.

2) If all the array elements are -ve.

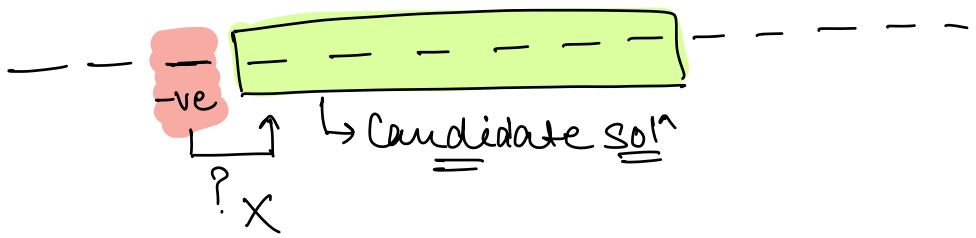
-1, -25, -10, -16, -2

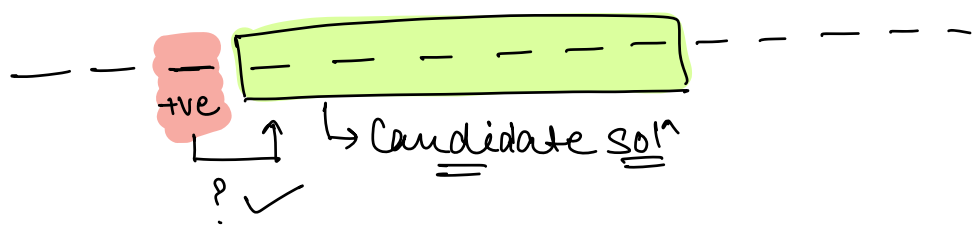
↳ max sum subarray.

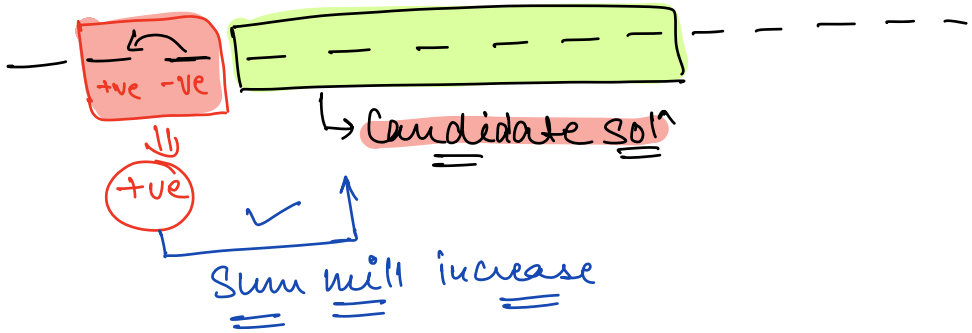
3)



4)







Ex :-

	5	6	7	-3	2	-10	-12	8	12	21	-4	7
Sum =0	5	11	18	15	17	7	~~-5~~ 0	8	20	41	37	44
ans =-∞	5	11	18	18	18	18	18	18	20	41	41	44

ans = max(ans, sum);

8+12+21-4+7
= 44

Ex :-

	-20	-10	-1	-5
Sum = 0	~~-20~~ 0	-10	~~-1~~ 0	-5
ans = -∞	-20	-10	-1	-1

Ex :-

	-20	10	-2	6	11	8	-16	-8	30
Sum=0	~~-20~~ 0	10	8	14	25	33	17	9	39
ans=-∞	-20	10	10	14	25	33	33	33	39

	-20	10	-2	6	11	8	-16	-18	30
Sum=0	~~-20~~ 0	10	8	14	25	33	17	~~-10~~ 0	30
ans=-∞	-20	10	10	14	25	33	33	33	33

KADANE's Algorithm

```
ans = -∞
sum = 0
for(i = 0; i < N; i++){
    sum += a[i]
    ans = max(ans, sum);
    if(sum < 0)
        sum = 0;
}
```

TC: O(N)

SC: O(1)

HW:- Print the start & end index of Max sum subarray

Q. Given an Array of size N & Q queries.
Every query will have ⇒ (x) & (val).
Add the given val to all the indices from (x) to (N-1). Initially all array elements are 0. Return the final state of the Array.

Ex:-

	0	1	2	3	4	5	6
A :	0	0	0	0	0	0	0

Q=3

x	val						
1	3	0	3	3	3	3	3	3
4	2	0	3	3	3	5	5	5
2	1	0	3	4	4	6	6	6

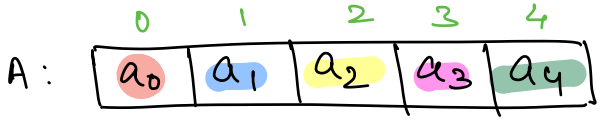
Brute force

```
for( k=1 ; k<=Q ; k++ ){ ⇒ Q
    for(i=x ; i<N ; i++ ){ ⇒ N
        a[i] += val;
    }
}
```

TC: O(N·Q)

SC: O(1)

Optimization



PF:

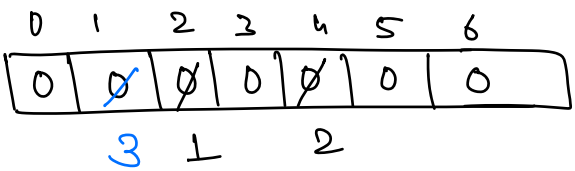
a_0

$a_0 + a_1$

$a_0 + a_1 + a_2$

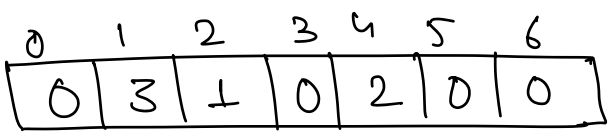
$a_0 + a_1 + a_2 + a_3$

$a_0 + a_1 + a_2 + a_3 + a_4$

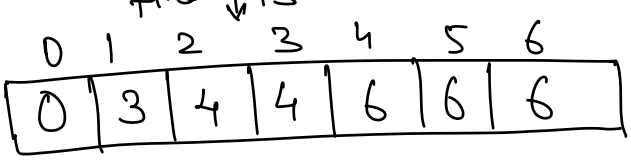


x	value
1	3
4	2
2	1

A[x] += value



find ↓PS



① for every query :- } ⇒ O(Q)

A[x] += value;

② find PS of (A). ⇒ O(N)

TC: O(N+Q)

SC: O(1)

Ex :-

	0	1	2	3	4	5	6	7
A :	0	0	~~0~~ 4	~~0~~ 2	0	~~0~~ 1	0	0

x	value
2	3
5	1
3	2
2	1

	0	1	2	3	4	5	6	7
A :	0	0	4	2	0	1	0	0

	0	1	2	3	4	5	6	7
A :	0	0	4	6	6	7	7	7

Q. Beggar's Outside Temple

Given N array elements & Q queries.
Initially all array elements are 0.
Each query ⇒ s, e, value
Return the array after all the queries.

	0	1	2	3	4	5	6	7
A:	0	0	0	0	0	0	0	0

s	e	val							
2	5	3	0	0	3	3	3	3	0	0
1	3	1	0	1	4	4	3	3	0	0
4	7	5	0	1	4	4	8	8	5	5
0	4	2	2	3	6	6	10	8	5	5

Brute force

```
for (k = 1 ; k <= Q ; k++) { ⇒ Q
    s, e, val
    for (i = s ; i <= e ; i++) { ⇒ N
        a[i] += val;
    }
}
```

TC: O(N·Q)
SC: O(1)

	0	1	2	3	4	5	6
A:	0	0	~~0~~ 4	0	0	~~0~~ -4	0

s	e	val
2	4	4

```
A[s] += val;
A[e+1] -= val;
```

A: 0 0 4 0 0 -4 0

↓ PS

0 0 4 4 4 0 0

Ex

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

s	e	val
3	6	1
1	4	2
2	5	3
4	7	1
5	6	2

```
A[s] += val;
A[e+1] -= val;   ( e+1 < N )
```

A: 0 2 3 1 1 0 -3 -3

PS ↓

A: 0 2 5 6 7 7 4 1

Steps :-

① for every query :- ⇒ Q

A[s] += val;

A[e+1] -= val; { Edge case e == N-1 }

② Find PS ⇒ N

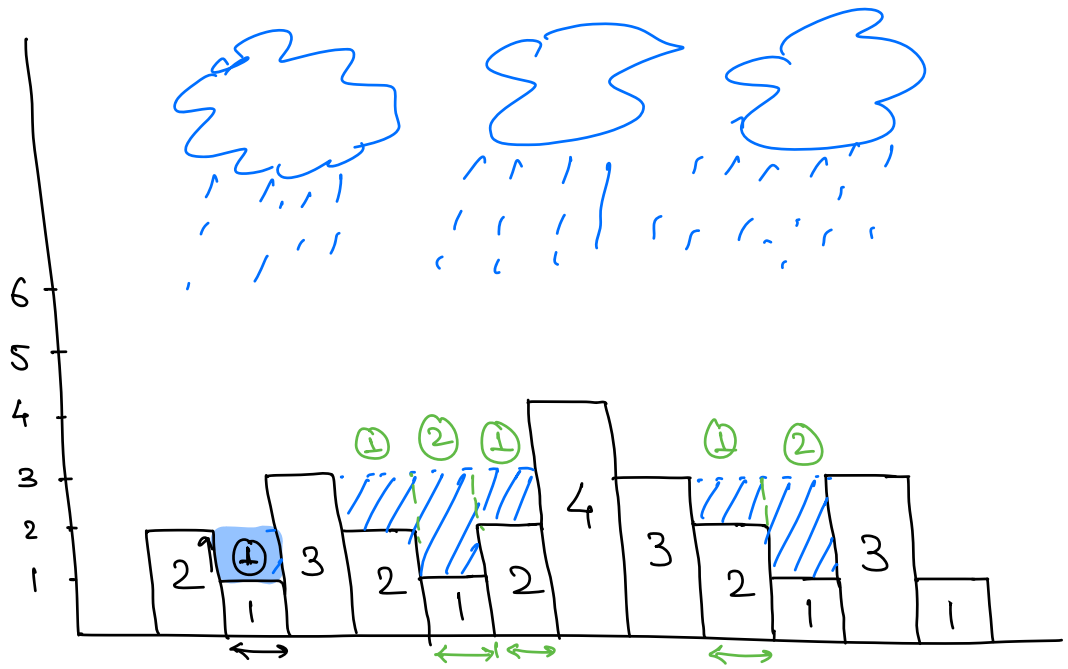
TC : O(N+Q)

SC : O(1)

Q. Rain Water Trapping MS | Amazon | Apple | GS | JPM | Bloomberg | Paytm | - -

Given N array elements, A[i] denotes the height of ith Building. Return the units of water trapped b/w the buildings

Ex:- { 2, 1, 3, 2, 1, 2, 4, 3, 2, 1, 3, 1 }
(0 1 2 3 4 5 6 7 8 9 10 11)

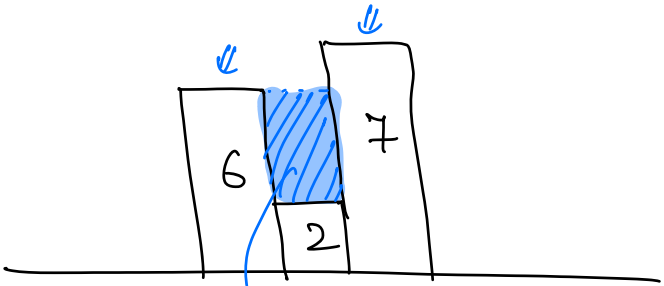


Total water = 1 + 1 + 2 + 1 + 1 + 2

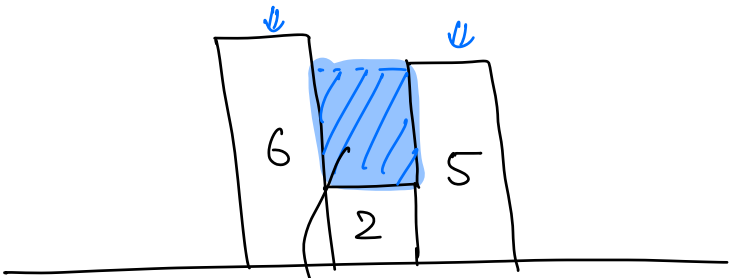
= (8)

Ans:- Sum of the units of water trapped over each building.

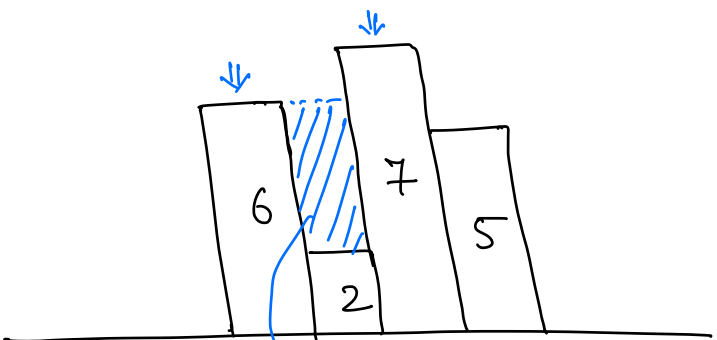
⇒



④ = min(6, 7) - 2

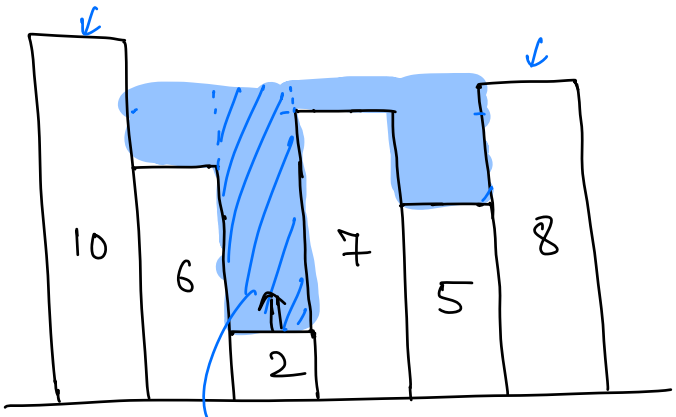


③ = min(6, 5) - 2



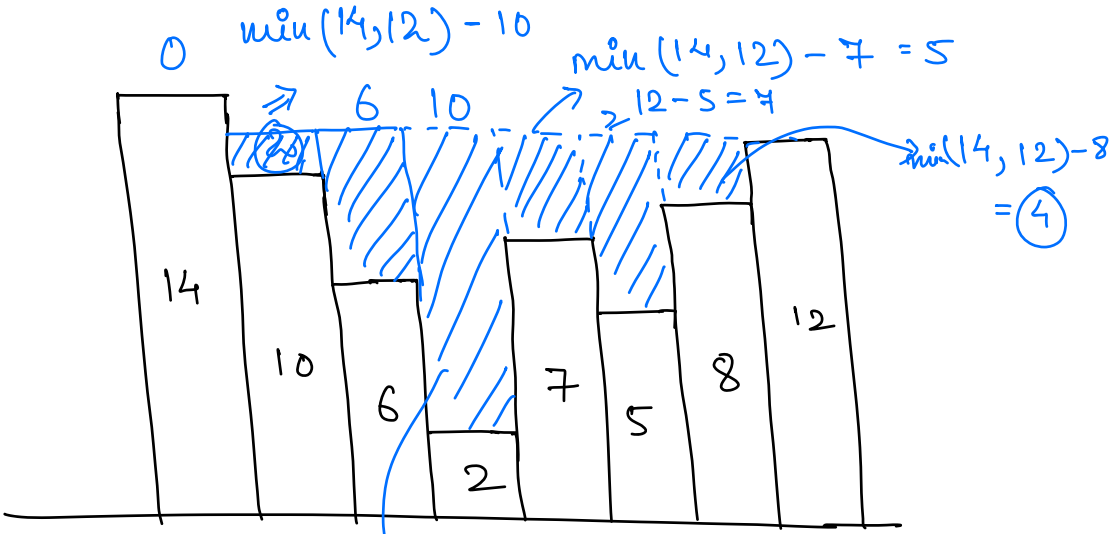
④ = min(6, 7) - 2

= 6 - 2



(6) = min(10, 8) − 2

= 8 − 2 = 6



min(14, 12) − 2

12 − 2

= 10

for any building :-

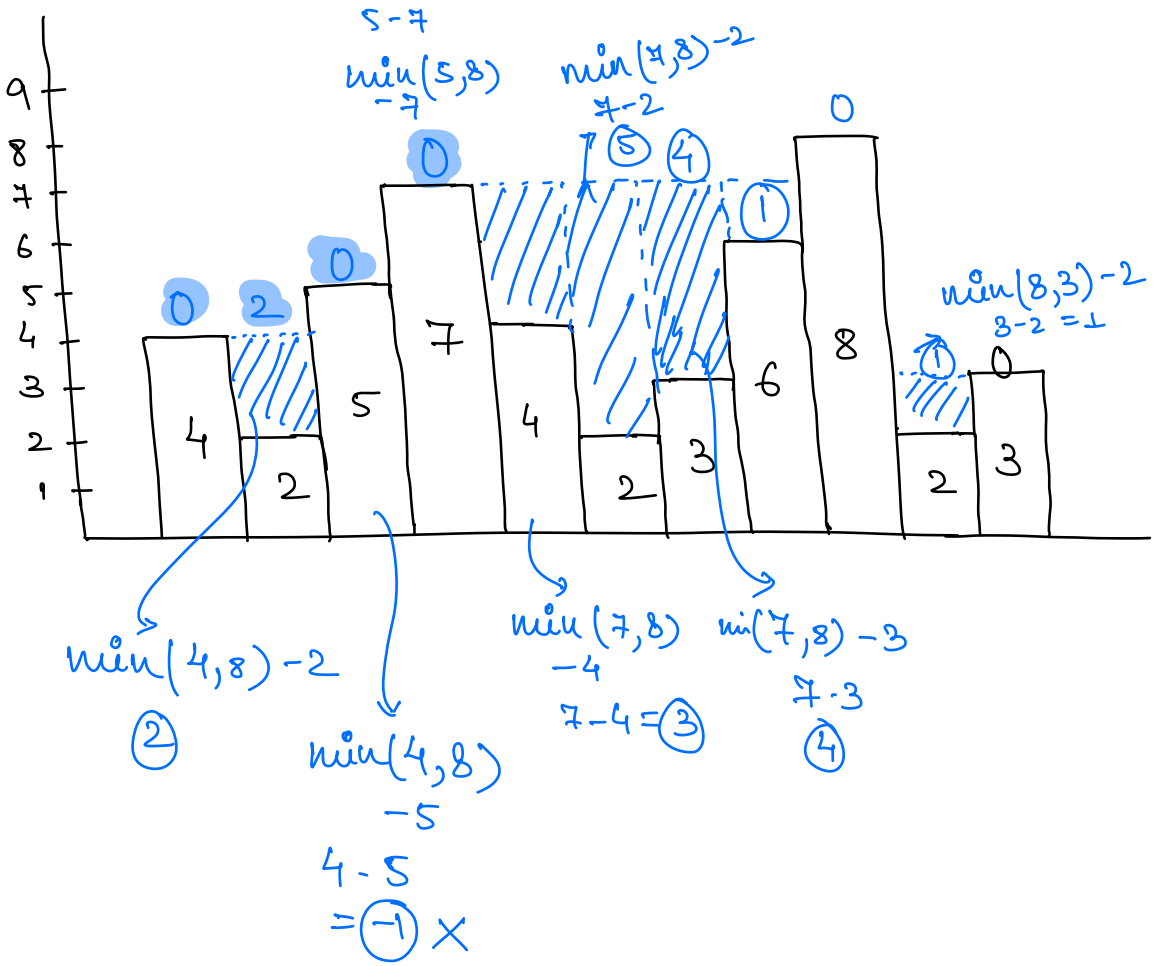
leftBoundary = leftMax

rightBoundary = rightMax

ans = min(leftBoundary, rightBoundary) - a[i]

ans = min(leftMax, rightMax) - A[i]

Ex:- {4, 2, 5, 7, 4, 2, 3, 6, 8, 2, 3}



ans ⇒ 16

① for any Building, if value is coming out be -ve then NO water can be trapped over this Building.

② NO water will trapped over the buildings at index 0 & N-1.

```
ans = 0
for (i = 1 ; i < N-1 ; i++ ){
    // ith Building
    lmax = max [0, i-1]  ⇒ O(N)
    rmax = max [i+1, N-1] ⇒ O(N)
    val = min(lmax, rmax) - a[i]
    if (val < 0) val = 0
    ans += val;
}
```

TC: $O(N^2)$

SC: $O(1)$

Optimization in lmax & rmax

	0	1	2	3	4	5	6	7
A:	2	4	1	7	9	6	5	3
leftMax:-	0	2	4	4	7	9	9	9
rightMax:-	9	9	9	9	6	5	3	0

leftMax[i] = max[0, i-1] ✓

rightMax[i] = max[i+1, N-1] ✓

leftMax[0] = 0 ⇒ LM[i] = max(LM[i-1], A[i-1])

rightMax[N-1] = 0 ⇒ HW

1) Create LMax ⇒ O(N)

2) Create RMax ⇒ O(N)

3) for i = 1 to N-2 ⇒ O(N)

   Val = min(LMax[i], RMax[i]) - A[i]

TC : O(N)

SC : O(N)

SC : O(1) ⇒ Two pointers / variables

TC : O(N) ↳ Two pointer Class

Doubts

	-10	-4	-2	-1
Sum=0	~~-10~~ 0	~~-4~~ 0	~~-2~~ 0	-1
ans=-∞	-10	-4	-2	-1 ✓