* Important observations around Prefix Sum

↳ carrying the term

	0	1	2	3	4	5	6	7	8	9
A =	0	0	2	0	0	0	0	0	0	0
PS =	0	0	2	2	2	2	2	2	2	2

	0	1	2	3	4	5	6	7	8	9
A =	0	0	2	0	0	0	-2	0	0	0
PS =	0	0	2	2	2	2	0	0	0	0

	0	1	2	3	4	5	6	7	8	9
A =	0	0	2	0	4	0	0	0	0	0
PS =	0	0	2	2	2+4	2+4	2+4	2+4	2+4	2+4

	0	1	2	3	4	5	6	7	8	9
A =	0	0	2	0	4	0	-2	0	0	-4
PS =	0	0	2	2	2+4	2+4	4	4	4	0

Q.1 **Continuous sum query** (Frequently asked in Hoogle)

Given array with all elements = 0 and Q queries. For every query {start, end, val} do **+val** in the range start to end. Return the final answer array.

	0	1	2	3	4	5	6	7	8	9
A =	0	0	0	0	0	0	0	0	0	0
				+1	+1	+1	+1			
			+3	+3	+3	+3	+3	+3		
					+5	+5	+5			
		-4	-4	-4	-4	-4				
	0	-4	-1	0	5	5	9	3	0	0

Queries

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

i) Brute idea : TC → **O(Q*N)**

				1				-1		
A =	0	0	0	0	0	0	0	0	0	0
	0	1	2	3	4	5	6	7	8	9
PS =	0	0	0	1	1	1	1	0	0	0

s	e	val
3	6	1

A =	0	-4	3	1	5	0	4	-5 -1	-3	0
	0	1	2	3	4	5	6	7	8	9
PS =	0	-4	-1	0	5	5	9	3	0	0

Queries

s	e	val		
3	6	1	→	A[3] += 1, A[7] += -1
2	7	3	→	A[2] += 3, A[8] += -3
4	6	5	→	A[4] += 5, A[7] += -5
1	5	-4	→	A[1] += -4, A[6] += 4

	0	1	2	3	4	5
A =	0	0	0	0	0	0
		+3	+3	+3	+3	+3
		+2	+2	+2		
	-1	-1	-1	-1		
	-1	4	4	4	3	3

Queries =

1	5	3
1	3	2
0	3	-1

		2			+1	
	-1	3			-2	
A =	~~0~~	~~0~~	0	0	~~0~~	0
	0	1	2	3	4	5
ps:	-1	4	4	4	3	3

Queries
=

1 5 3 , A[1] += 3

1 3 2 , A[1] += 2 A[4] += -2

0 3 -1 , A[0] += -1 A[4] += 1

```
int [] solve ( int [] A , int [][] Q ) {

    for (int i = 0; i < Q.length; i++) {
        int s = Q[i][0];
        int e = Q[i][1];
        int val = Q[i][2];
        // give impact of val in s to e
        A[s] += val;
        if (e+1 < A.length) {
            A[e+1] += -val;
        }
    }

    // converting A[] into its prefix sum array

    for (int i=1; i<A.length; i++) {
        A[i] = A[i-1] + A[i];
    }

    return A;
}
```

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

Q: 1 2 5
0 2 1

TC: O(Q + N)

SC: O(1)

Q. 2 Create prefixMax and suffixMax array.

A =	2	4	3	1	12	5	6	8
	0	1	2	3	4	5	6	7

prefixMax[i] => max of (0, i)

suffixMax[i] => max of (i, n-1)

	0	1	2	3	4	5	6	7
A =	2	4	3	1	12	5	6	8
pfmax	2	4	4	4	12	12	12	12
sfmax	12	12	12	12	12	8	8	8

pfmax[i] = max(pfmax[i-1], A[i]);

sfmax[i] = max(sfmax[i+1], A[i]);

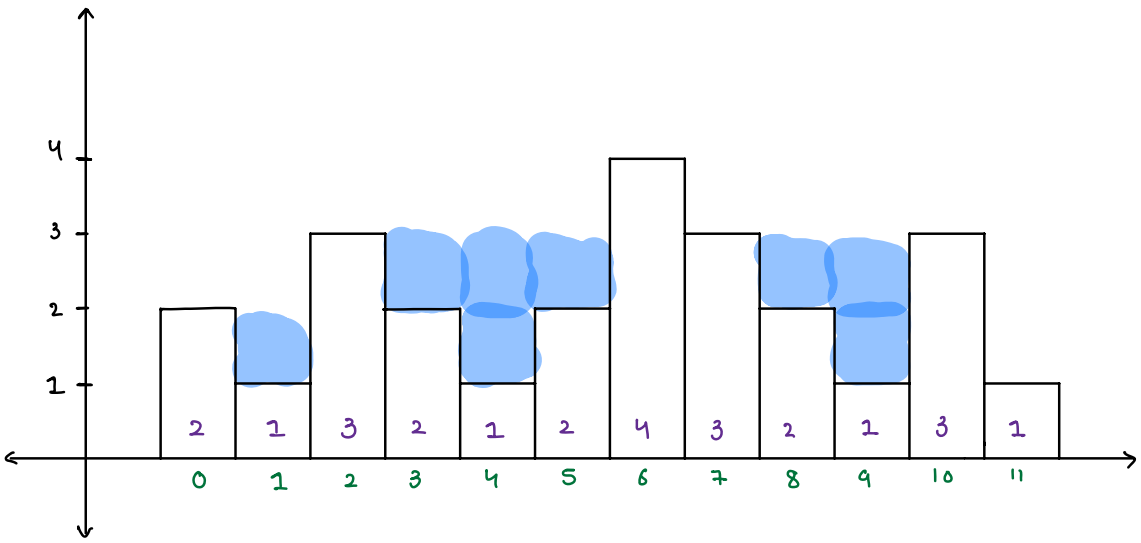
```
int [] prefixMax (int [] A) {
    int n = A.length;
    int [] pfmax = new int [n];
    pfmax [0] = A[0];
    for (int i=1; i<n; i++) {
        pfmax[i] = Math.max (pfmax [i-1], A [i]);
    }
    return pfmax;
}
```

```
int [] suffixMax (int [] A) {
    int n = A.length;
    int [] sfmax = new int [n];
    sfmax [n-1] = A [n-1];
    for (int i=n-2; i>=0; i--) {
        sfmax[i] = Math.max (sfmax [i+1], A [i]);
    }
    return sfmax;
}
```

Q-3 **Rainwater trapping**

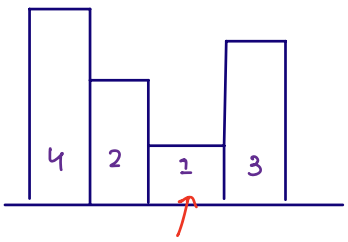
Given an array A[], where A[i] denotes height of i^{th} building.
Return amount of water trapped on all buildings.

	0	1	2	3	4	5	6	7	8	9	10	11
A = [	2	1	3	2	1	2	4	3	2	1	3	1]



Ans = 8

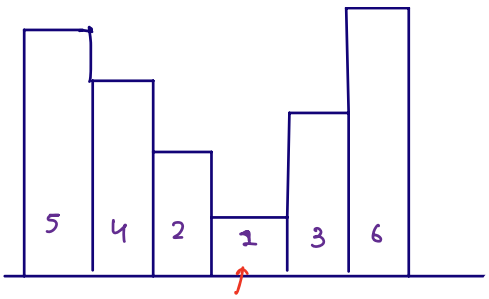
total water trapped = sum of water units trapped on each building's rooftop.



lb = 4

rb = 3

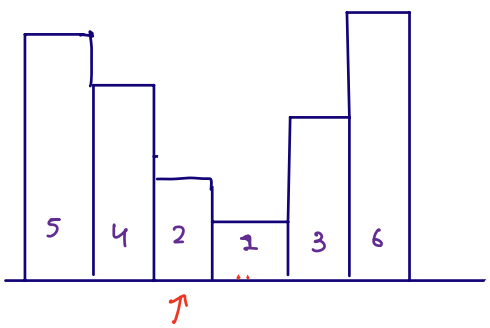
amt = 3 - 1 = 2



lb = 5

rb = 6

amt = 5 - 1 = 4



lb = 5

rb = 6

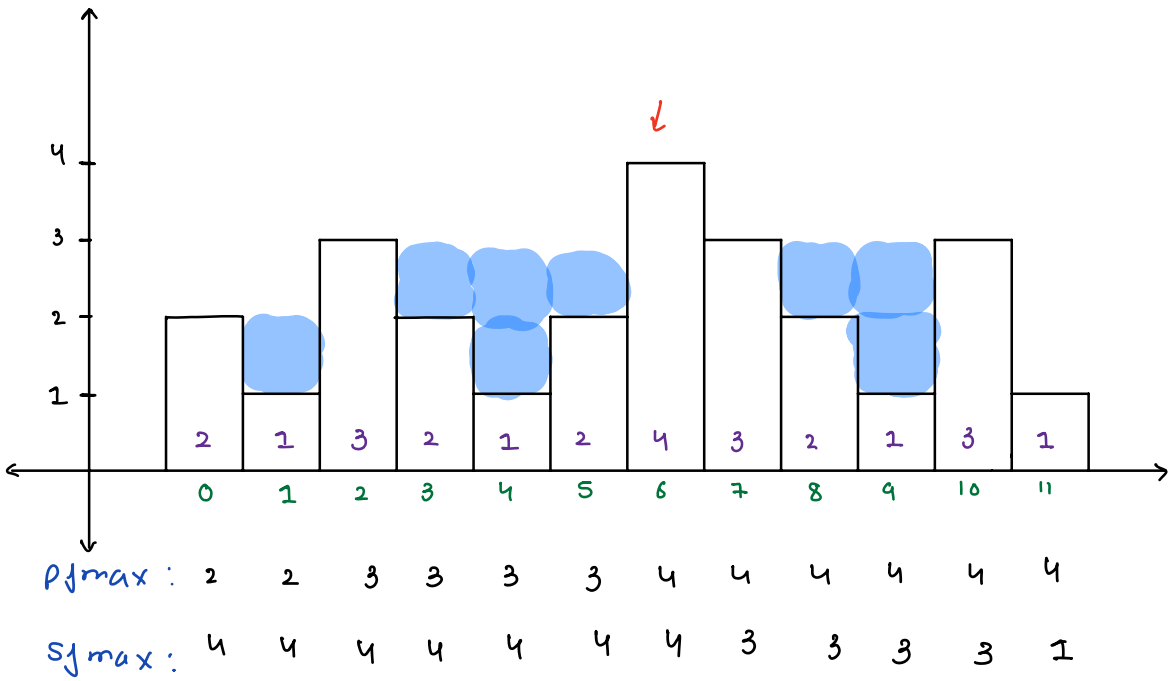
amt = 5 - 2 = 3

lb → max(0, i-1)

= Pjmax[i-1]

rb → max(i+1, n-1)

= Sjmax[i+1]



i	lb	rb	amt
1	2	4	2-1=1
2	2	4	2-3=-1
3	3	4	3-2=1
4	3	4	3-1=2
5	3	4	3-2=1
⋮			

```
int rainwater(int[] A) {
    int[] pfmax = prefixMax(A);
    int[] sfmax = suffixMax(A);
    int ans = 0;
    for(int i=1; i<A.length-1; i++) {
        int lb = pfmax[i-1];
        int rb = sfmax[i+1];
        int amt = Math.min(lb, rb) - A[i];
        if(amt > 0) {
            ans += amt;
        }
    }
    return ans;
}
```

Q-4 **Maximum Sum Subarray** (Kadane's Algo)

Given an array, find max sum **subarray**.

↳ continous part an array

A = [3 2 -6 **8 2 9 4**] ans = 23
0 1 2 3 4 5 6

A = [-3 **2 4 -1 3** -4 3] ans = 8
0 1 2 3 4 5 6

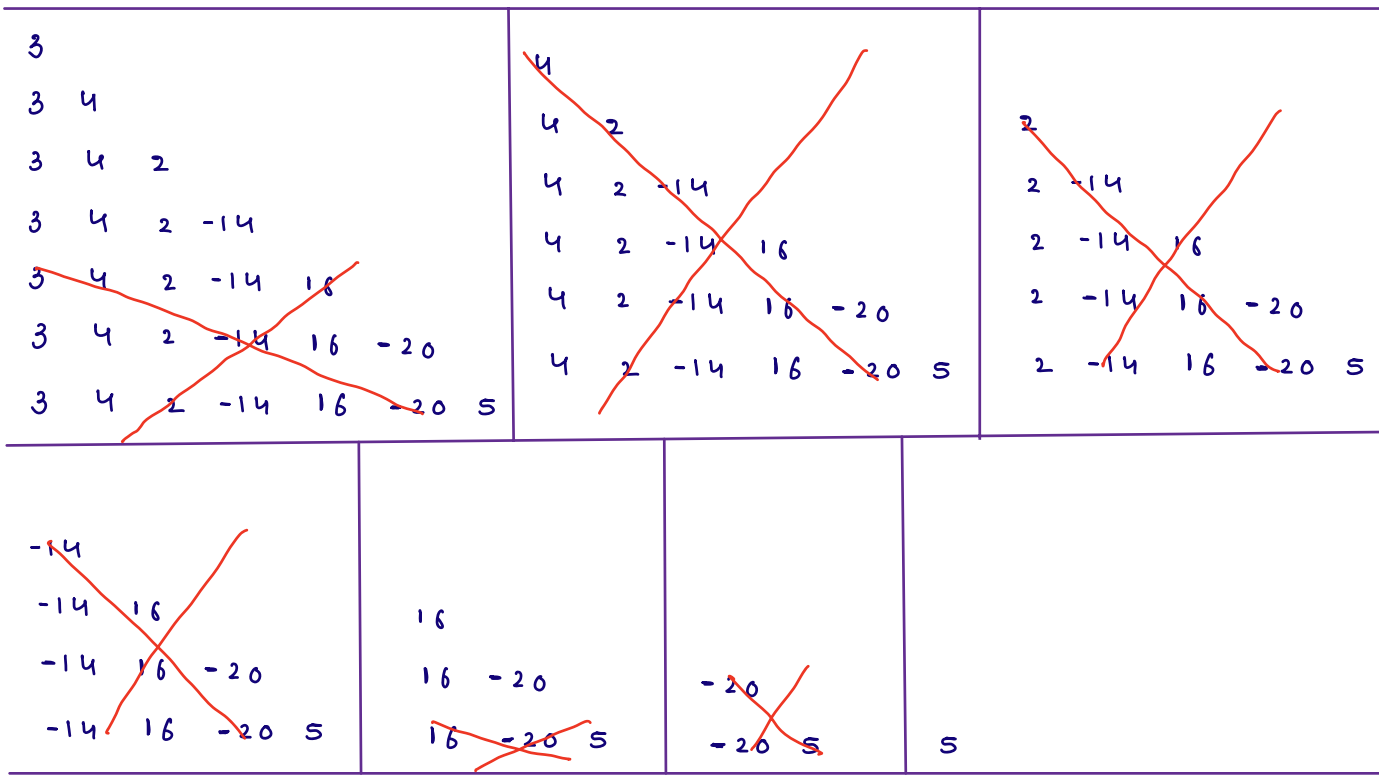
A = [3 4 2 -14 **16** -20 5] ans = 16
0 1 2 3 4 5 6

i) idea 1 TC: $O(N^2)$

create prefix sum array, now find sum of every subarray and keep a track of max ans.

ii) idea 2 TC: $O(N)$

	0	1	2	3	4	5	6
A	[3	4	2	-14	16	-20	5]
Sum = 0	3	7	9	-5	16	-4	5
ans = -∞	3	7	9	9	16	16	16



	0	1	2	3	4	5	6
A =	[-3	2	4	-1	3	-4	3]
Sum = 0	-3	2	6	5	8	4	7
ans = -∞	-3	2	6	6	8	8	8

```
int maxSumSubarray(int[] A) {
    int sum = 0;
    int ans = Integer.MIN_VALUE;
    for(int i=0; i< A.length; i++) {
        if(sum > 0) {
            sum += A[i];
        }
        else {
            sum = A[i];
        }
        if(sum > ans) {
            ans = sum;
        }
    }
    return ans;
}
```

A = [2 3 -7 4 1 -2 5 -1]
0 1 2 3 4 5 6 7

sum = 0	2	5	-2	4	5	3	8	7
ans = -∞	2	5	5	5	5	5	8	8