* Important observations around Prefix Sum

↳ carrying the team

	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 Google)

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

Queries

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

			+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

i) Idea 1 : TC → O(Q*N)

```
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 the impact of val in range s to e
        for(int k=s; k<=e; k++) {
            A[k] += val;
        }
    }
    return A;
}
```

(ii) Expected TC : O(N+Q)

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
2	7	3
4	6	5
1	5	-4

→ A[3] += 1 A[7] += -1

→ A[2] += 3 A[8] += -3

→ A[4] += 5 A[7] += -5

→ A[1] += -4 A[6] += 4

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

Queries =

S	e	val
2	4	2
1	5	3
0	4	-1

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

Queries

=

S	e	val		
2	4	2	A[2] += 2	A[5] += -2
1	5	3	A[1] += 3	
0	4	-1	A[0] += -1	A[5] += 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 range s to e
        A[s] += val;
        if (e+1 < A.length) {
            A[e+1] += -val;
        }
    }

    // convert A[] into its prefix sum array
    for(int i=1; i< A.length; i++) {
        A[i] = A[i-1] + A[i];
    }

    return A;
}
```

Q → [[1,3,2] [2,4,3]]

A =

	2	3		-2	-3
0	~~0~~	~~0~~	0	~~0~~	~~0~~
0	1	2	3	4	5

A =

	2	3		-2	-3
0	~~0~~	~~0~~	0	~~0~~	~~0~~
0	1	2	3	4	5

0 2 5 5 3 0

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
pfmax	2	4	4	4	12	12	12	12
sfmax	12	12	12	12	12	8	8	8

prefix Max [i] → max of 0 to i

suffix Max [i] → max of i to n-1

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;
}
```

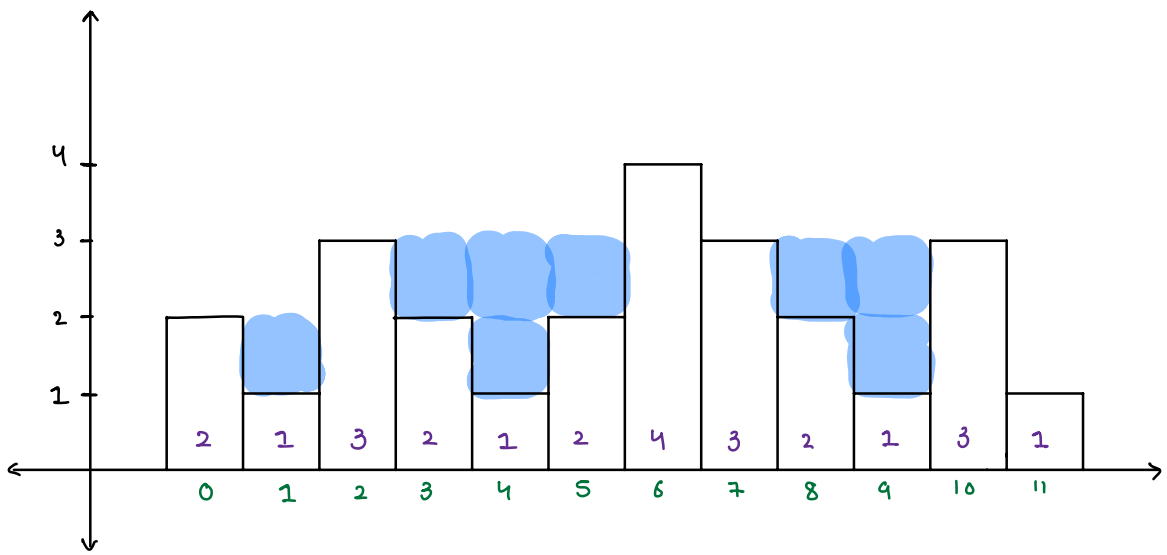
```
int[] suffixMax (int[] A) {
    int n = A.length;
    int[] sfmax = new int[n];
    sfmax[n-1] = A[n-1];
    for (int i=n-2; i>=0; i--) {
        sfmax[i] = Math.max(sfmax[i+1], A[i]);
    }
    return sfmax;
}
```

	0	1	2	3	4
A =	4	2	6	5	3
sfmax	6	6	6	5	3

Q-3 **Rainwater trapping**

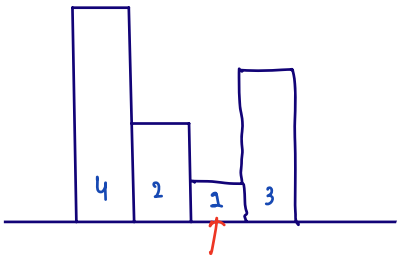
Given an array A[], where A[i] denotes height of i^{th} building. Return amount of water trapped on all buildings.

	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

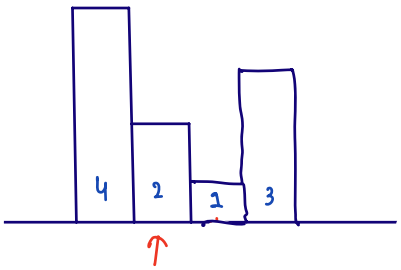
total water trapped = sum of water units trapped on each building's rooftop.



lb = 4

rb = 3

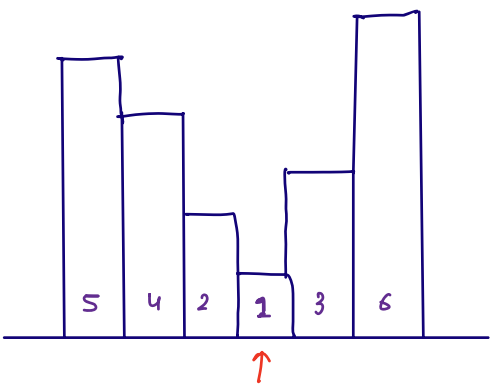
amt = 3 - 1 = 2



lb = 4

rb = 3

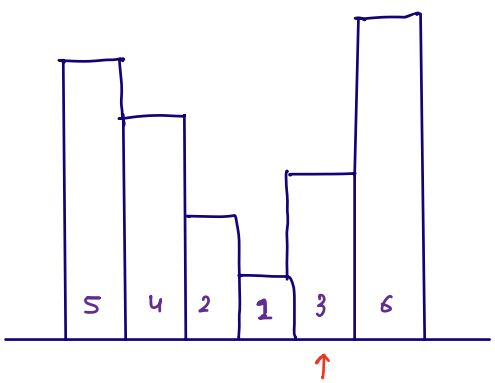
amt = 3 - 2 = 1



lb = 5

rb = 6

amt = 5 - 1 = 4



lb = 5

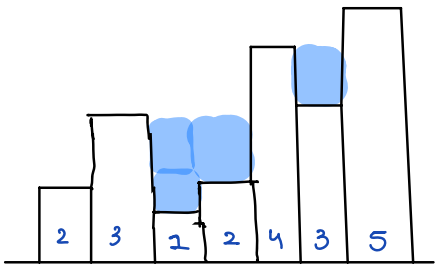
rb = 6

amt = 5 - 3 = 2

lb = max of 0 to i-1 → Lmax[i-1]

rb = max of i+1 to n-1 → Rmax[i+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;

}
```



pfmax	2	3	3	3	4	4	5
sfmax	5	5	5	5	5	5	5

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

ans = 4

TC: O(N)

SC: O(N)

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

Given an array, find max sum subarray.

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

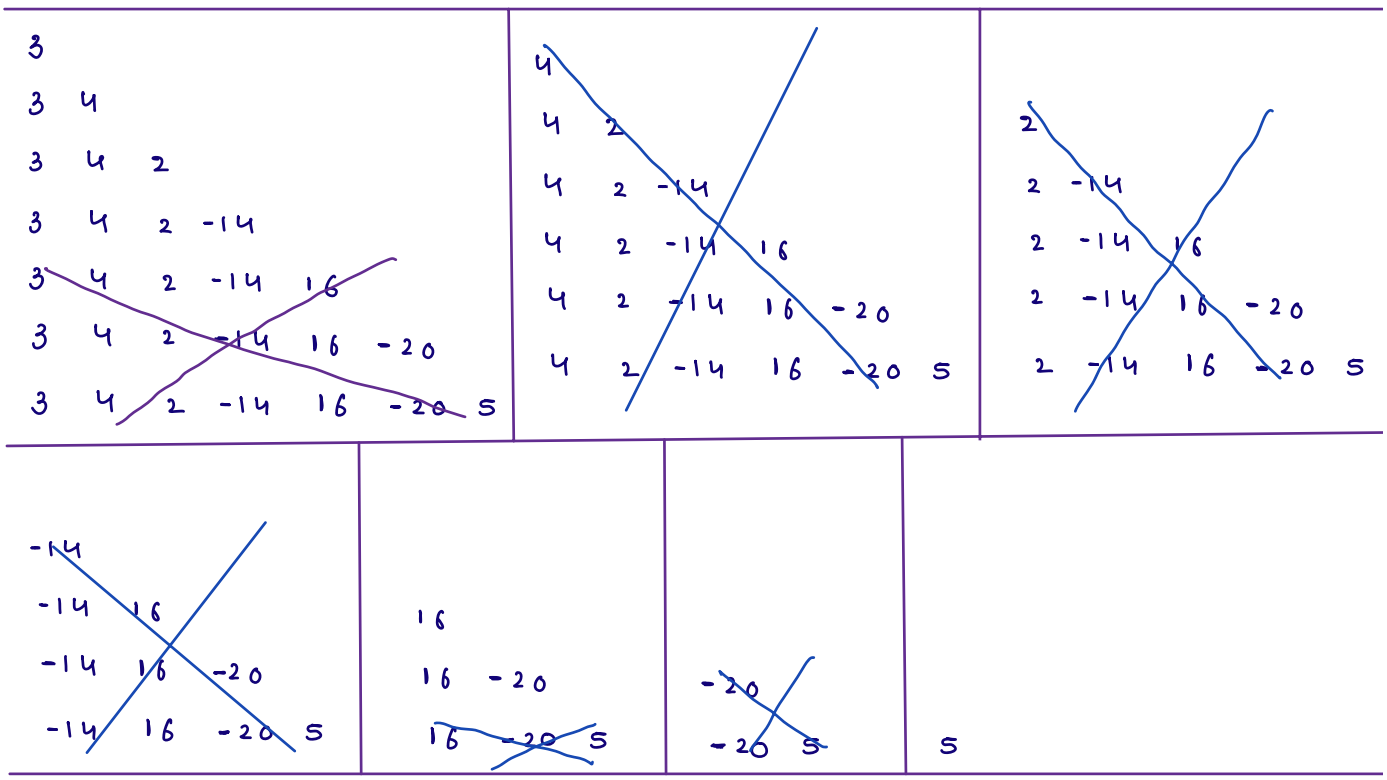
A = [-3 2 4 -1 3 -4 3]
0 1 2 3 4 5 6 ans = 8

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

i) **idea 1** : Go on every subarray and find sum using prefix sum and compare that with overall ans. $O(N^2)$

ii) Expected 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



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

Sum = 0	-3	2	6	5	8	4	7
ans = -∞	-3	2	6	6	8	8	8

```
int kadane(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;
}
```

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

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

Sum = 0 3 7 9 -5 16 -4 5

ans = -∞ 3 7 9 9 16 16 16

A = [-3 -1 -5]
0 1 2

Sum = 0 -3 -1 -5

ans = -∞ -3 -1 -1