SCALER

One Dimensional Array

TABLE OF CONTENTS

Welcome to Advanced Module

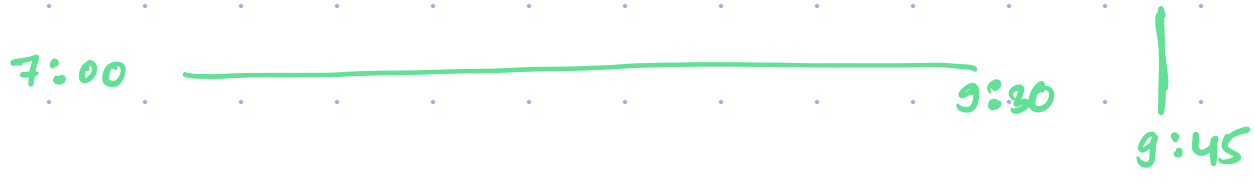
Class starts at 7:05 IST

Skill Evaluation Contest

The last class of the DSA curriculum will be your skill Evaluation contest which will consist of the entire DSA module as its syllabus and this is a mandatory contest which everyone has to clear.

Mock interview :

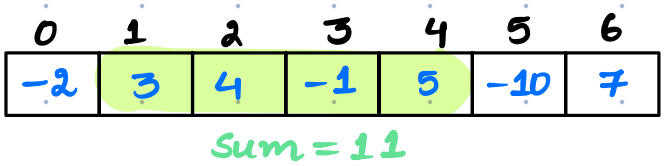
It's a 1-hour coding interview designed to assess your problem-solving abilities and provide a real-life interview experience. You'll have the opportunity to interact with industry experts and explain your ideas in the live session.
You will be given a specific timeframe to clear it, and once you successfully complete the mock interview, you will receive a badge from Scaler, which you can proudly showcase on various platforms!



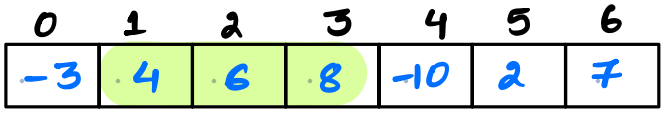
Amazon
Flipkart
Goldman Sach

Problem: Maximum Subarray Sum

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



Sum = 11



Sum = 18

Quiz : 1

For the given array A, what is the maximum subarray sum ?

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

ans : Entire array

4 + 5 + 2 + 1 + 6 = 18

Quiz : 2

For the given array A, what is the maximum subarray sum ?

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

-2 > -9

ans : Max element of array

-2

BF Idea

- Generate all subarrays
- Find their sum & max sum out of it.

< / > Code

```
ans = arr[0];
// Generate subarrays
for(s=0; s<n; s++) {
    for(e=s; e<n; e++) {
        int sum=0;
        for(i=s; i<=e; i++) {
            sum+= arr[i];
        }
        ans = max(ans, sum);
    }
}
return ans;
```

T.C : O(N3)
S.C : O(1)

Idea -1

Using psum[ ]

T.C : $O(N^2)$
S.C : $O(N)$

SCALER

Idea-2 Using Carry Forward

```
ans = arr[0];
// Generate subarrays
for(s=0; s<n; s++) {
    int sum=0;
    for(e=s; e<n; e++) {
        sum += arr[e];
        ans = max(ans, sum);
    }
}
return ans;
```

T.C : $O(N^2)$

S.C : $O(1)$

1	2	3	
1			arr[0]
1	2		arr[0] + arr[1]
1	2	3	arr[0] + arr[1] + arr[2]

SCALER

Optimised Approach

KADANE' S algorithm

Case I : All elements are POSITIVE

0	1	2	3	4
4	1	2	6	3

ans : Sum of all positive elements

Entire array.

Case II : All elements are NEGATIVE

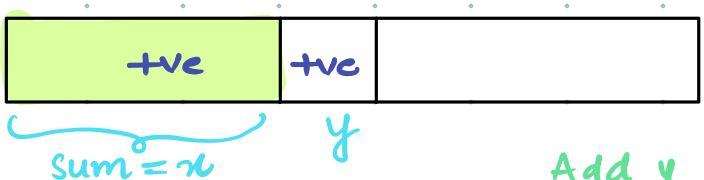
0	1	2	3	4
-4	-2	-1	-9	-7

ans : max element of array/ min negative element

Case III :

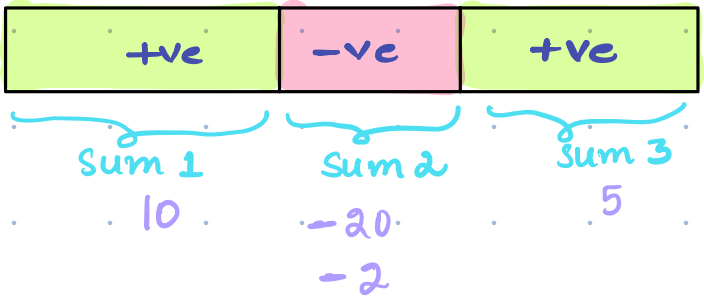
ans : Sum of all positive elements

Case IV :



Add y to the sum

Case V :



sum1 + sum2 >0 → Consider these (-ve) elements b'cox in future if some positive elements is there it will increase sum.

sum1 + sum2 <0 → Ignore (-ve) elements

2	3	-4	6

Sum 0: 2 5 1 7

Ans -∞: 2 5 5 7

ans : 7

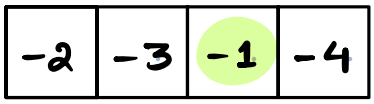
2	3	-6	7

Sum 0: 2 5 ~~-1~~ 0 7

Ans -∞: 2 5 5 7

ans = 7

SCALER



Sum
0

-2 0 -3 0 -1 0 -4 0

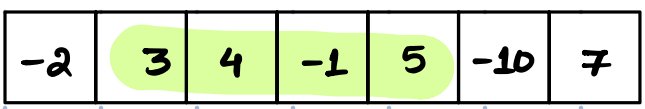
Ans
$-\infty$

-2 -2 -1 -1

ans = -1

Quiz : 3

Tell the output of the below example after running the Kadane's Algorithm on that example
A[ ] = { -2, 3, 4, -1, 5, -10, 7 }



Sum
0

-2 3 7 6 11 1 8
0

Ans
$-\infty$

-2 3 7 7 11 11 11

ans: 11

< / > Code

```
int maxSubarraySum(int arr[], int n) {
    sum = 0 , ans = INT_MIN ;

    for ( i=0 ; i<n ; i++) {
        sum = sum + arr[i] ;
        ans = max (ans, sum) ;
        if (sum<0)
            sum = 0 ;
    }
    return ans ;
}
```

T.C : O(N).
S.C : O(1)

Problem : Zero Queries -1

Given an array where every element is 0. Find the final array after performing multiple queries.

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

Eg:

N = 7 , Q = 3

Queries :

i	x
1	3
4	2
3	1

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

Quiz : 4

Return the final array after performing the queries Note:

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

```
A = [0, 0, 0, 0, 0]
Query(1, 3)
Query(0, 2)
Query(4, 1)
```

0	1	2	3	4
0	0	0	0	0
	3	3	3	3
2	2	2	2	2
				1
2	5	5	5	6

SCALER

BF Idea

For each query, iterate & add x from [i ---- N-1]

T.C : O(Q*N)

S.C : O(1)

arr[ ] :

0	1	2	3	4	5
0	0	2	0	0	0

psum[ ]:

0	1	2	3	4	5
0	0	2	2	2	2

arr[ ] :

0	1	2	3	4	5	6	7	8
0	0	2	0	0	0	3	0	0

psum[ ]:

0	1	2	3	4	5	6	7	8
0	0	2	2	2	2	5	5	5
						3+2	3+2	3+2

arr[ ] :

0	1	2	3	4	5	6	7	8	9	10	11
0	0	1	0	2	0	0	6	0	3	0	0

psum[ ]:

0	1	2	3	4	5	6	7	8	9	10	11
0	0	1	1	3	3	3	9	9	12	12	12
			1	1+2	1+2	1+2	1+2+6	1+2+6	1+2+6+3	1+2+6+3	1+2+6+3

SCALER

Optimised Approach: Add the value x in arr[i] & then find prefix sum

N = 7 , Q = 3

Queries :

i	x
1	3
4	2
3	1

arr[]:

0	1	2	3	4	5	6
0	~~0~~	0	~~0~~	~~0~~	0	0
	3		1	2		

arr[]:

0	3	0	1	2	0	0

psum[]:

0	3	3	4	6	6	6

< / > Code

```
for (i = 0 ; i < Queries.length() ; i++) {
    int index = Queries [i] [0] ;
    int value = Queries [i] [1] ;
    arr [index] += value ;
}
// Find prefix sum
for (int i = 1 ; i < n ; i++) {
    arr [i] += arr [i-1] .
}
```

T.C : O(Q+N)
S.C : O(1)

Break till 8:30

SCALER

Problem: Zero Queries -2

Given an array where every element is 0. Find the final array after performing multiple queries.

Query (i, j, x): Add x to all numbers from index i to j. (i<=j)

Eg: N = 7 , Q = 3

Queries:

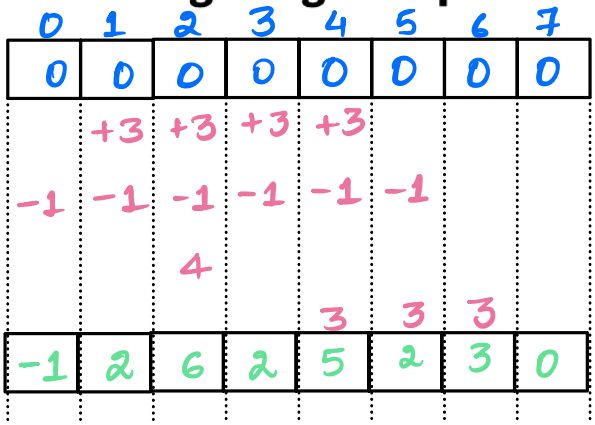
i	j	x
1	3	2
2	5	3
5	6	-1

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

Quiz : 5

Find the final array after performing the given queries on array of size 8.

i	j	x
1	4	3
0	5	-1
2	2	4
4	6	3



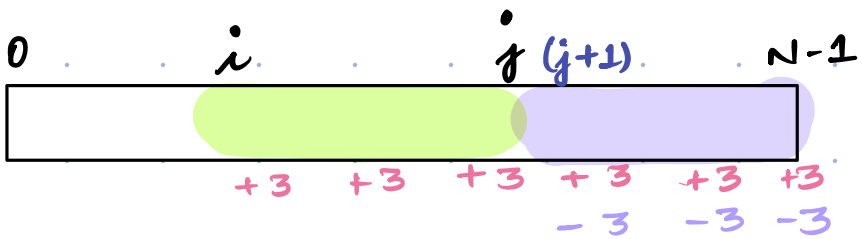
BF Idea

For each query, add value x from index [i ... j]

T.C : O(N*Q)

S.C : O(1)

Observations :



arr[i] += 3

arr[j+1] += (-3)

SCALER

Optimised Approach:

Add value x in $arr[i]$ & $-x$ in $arr[j+1]$.

Calculate prefix sum.

$N = 7$, $Q = 3$

Queries :

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

if (e == N-1)
nothing to add at (j+1) index.

arr[]:

0	1	2	3	4	5	6
0	~~0~~	~~0~~	0	~~0~~	~~0~~	~~0~~

+2 -2 +3 -3 -1

arr[]:

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

psum[]:

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

SCALER

< / > Code

```
int RangeSum(int[] arr, int n, int[][] queries)
{
    for(k=0; k<Queries.length(); k++){
        int i = Queries[k][0];
        int j = Queries[k][1];
        int x = Queries[k][2];
        // Add +x in arr[i] & -x in arr[j+1]
        arr[i] += x;
        if(j != N-1)
            arr[j+1] += (-x);
    }
    // calculate psum[]
    for(int i=1; i<n; i++){
        arr[i] += arr[i-1];
    }
}
```

T.C : O(Q+N)
S.C : O(1)

leetcode hard
Google

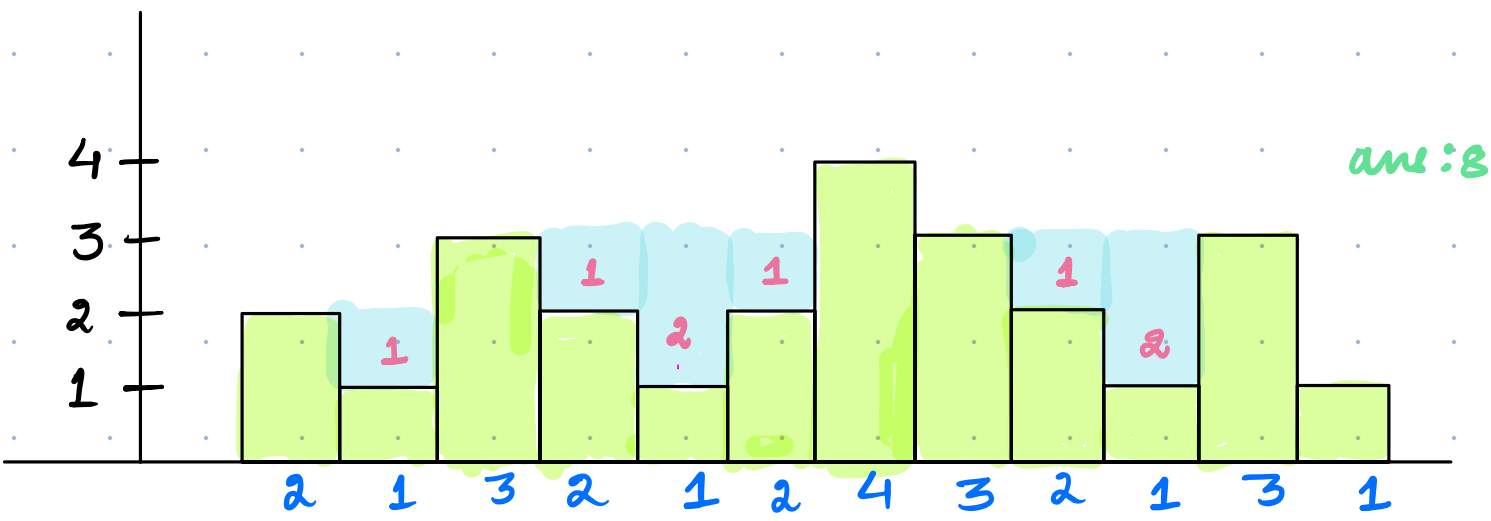
Problem: Trapping Rain Water

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

eg:

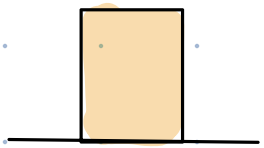
0	1	2	3	4	5	6	7	8	9	10	11
2	1	3	2	1	2	4	3	2	1	3	1

N=12

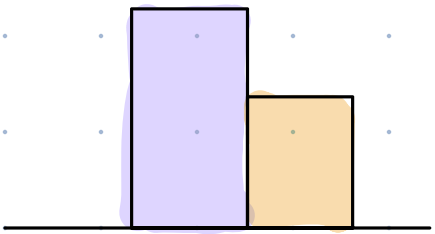


ans: 8

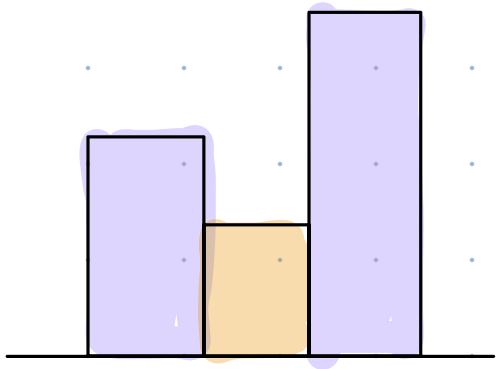
SCALER



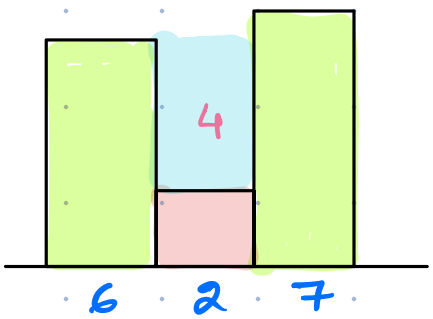
NO



NO



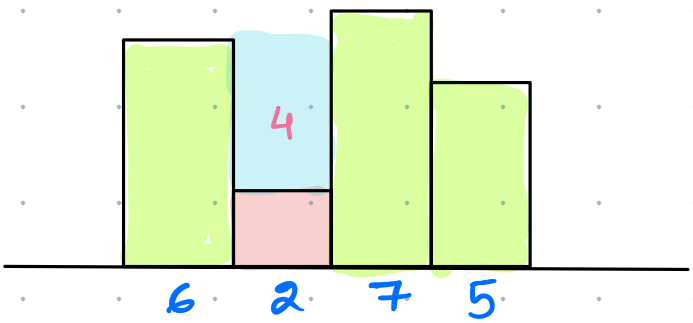
Water can only be trapped if there are taller buildings on both side.



$l_{max} = 6$ ✓

$r_{max} = 7$

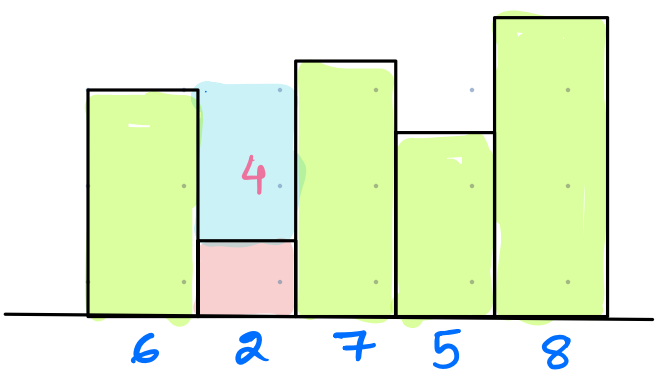
$6-2=4$



$l_{max} = 6$ ✓

$r_{max} = 7$

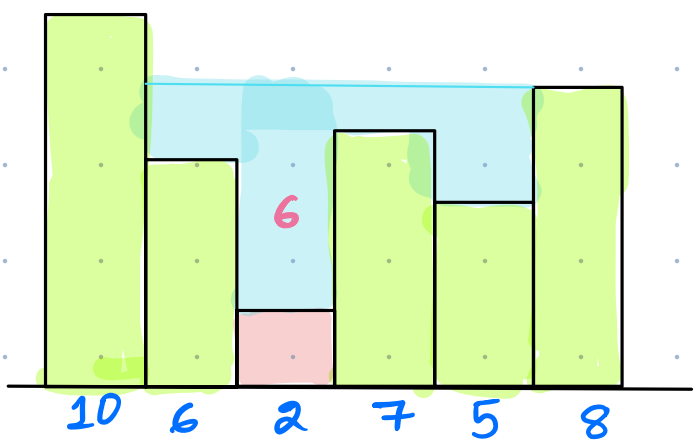
$6-2=4$



$l_{max} = 6$ ✓

$r_{max} = 8$

$6-2=4$

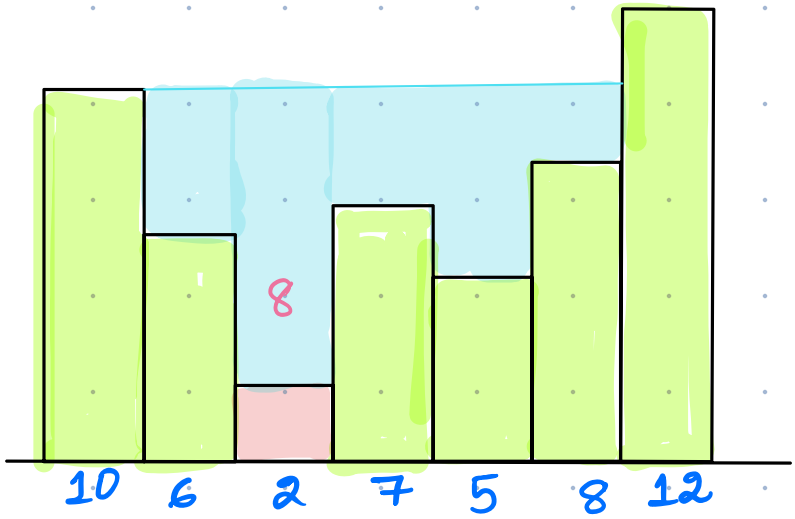


$l_{max} = 10$

$r_{max} = 8$ ✓

$8-2=6$

SCALER



l_{max} : 10 ✓

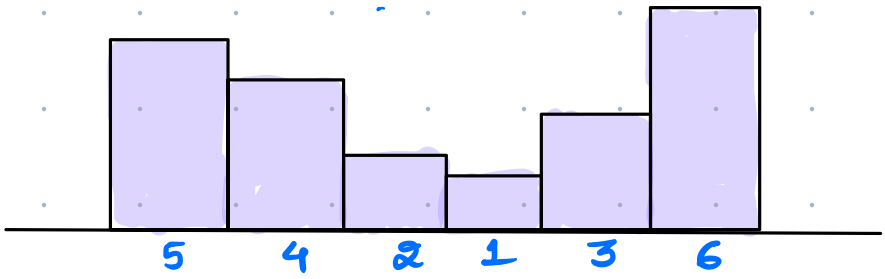
r_{max} : 12

$10 - 2 = 8$

Approach:

Water accumulated = min(l_{max}, r_{max}) − ht of building

SCALER

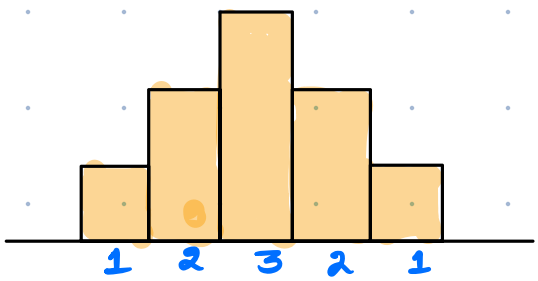


left max	5	5	5	5	5	6
right max	6	6	6	6	6	6
min(lmax, rmax)	5	5	5	5	5	6
water on building	0	1	3	4	2	0

ans : 1 + 3 + 4 + 2 = 10

Quiz : 6

Given N buildings with height of each building, find the rain water trapped between the buildings. `A = [1, 2, 3, 2, 1]`



lmax	1	2	3	3	3
rmax	3	3	3	2	1
min(lmax, rmax)	1	2	3	2	1
water on building	0	0	0	0	0

ans : 0

SCALER

< / > Code

```
ans = 0 ;
for(i = 0 ; i < n ; i++){
    // Find lmax : max from [0 .... i]
    // Find rmax : max from [i ..... N-1]
    int water = min(lmax , rmax) - h[i] ;
    ans += water ;

}
```

T.C : $O(N^2)$
S.C : $O(1)$

SCALER

Optimised Approach:

	0	1	2	3	4	5	6	7	8	9	10
	4	2	5	7	4	2	3	6	8	2	3
lmax[ ]	4	4	5	7	7	7	7	7	8	8	8
rmax[ ]	8	8	8	8	8	8	8	8	8	3	3

```
int lmax[n] , rmax[n];

// Build lmax
    lmax[0] = h[0];
for (i=1 ; i<n ; i++) {
    lmax[i] = max(lmax[i-1] , h(i));
}

// Build rmax
    rmax[N-1] = h[N-1];
for( i = N-2 ; i >=0 ; i--) {
    rmax[i] = max (rmax[i+1], h(i));
}

ans = 0;
for (i=0 ; i < n ; i++){
    int water = min( lmax[i] , rmax[i]) - h(i);
        ans += water;
}
return ans;
```

T.C : O(3N) = O(N)

S.C : O(N)