SCALER

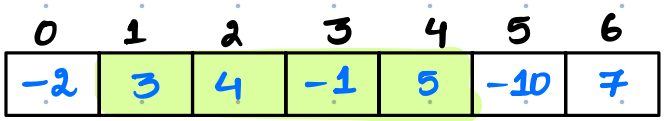
One Dimensional Array

TABLE OF CONTENTS

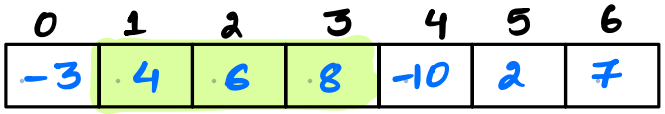
Maximum Subarray Sum

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

N(N+1)/2



Sum = 11



Sum = 18

Quiz : 1

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

A[ ] = { 4, 5, 2, 1, 6 } (0 1 2 3 4)

Sum = 18

Quiz : 2

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

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

-2 > -9

ans : -2

BF Idea

Consider all the subarrays. Find their sum & max.

</ > Code

```
ans = INT_MIN;
for(s=0; s<n; s++){
    for(e=s; e<n; e++){
    // Calculate sum
      int sum=0;
     for(i=s; i<=e; i++){
        sum += arr[i];
     }
     ans = Math.Max(ans,sum);
}
return ans;
```

T.C: O(N^3)

S.C: O(1)

Idea -1 Using psum[]

```
// Calculate psum[]
ans = INT_MIN;

for (s = 0; s < n; s++) {
    for (e = s; e < n; e++) {
        // Calculate sum
        int sum = 0;

        if (s == 0)  sum = psum[e];
        else         sum = psum[e] - psum[s-1];

        ans = Math.Max(ans, sum);
    }
}
return ans;
```

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

Idea-2 Using Carry Forward

```
int ans = INT_MIN;
for (s = 0; s < n; s++) {
    int sum = 0;
    for (e = s; e < n; e++) {
        sum += arr[e];

        ans = Math.max(ans, sum);
    }
}

return ans;
```

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

Optimised Approach

KADANE' S algorithm

Case I: All elements are POSITNE

0	1	2	3	4
4	1	2	6	3

ans : Sum of all the elements.

Case II: All elements are NEGATIVE

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

ans : Max element of array

Case III:

-20 5 -6

-1

-2 5 5

ans : Sum of all positive elements.

Case IV :

sum = x y

↳ Add y to the sum

Case V :

Sum 1	Sum 2	Sum 3
10	-3	5
10	-12	5

Sum 1 + Sum 2

> 0 → Consider these (-ve) elements in sum b'coz in future if some (+ve) elements come, they will increase the sum.

< 0 → Ignore (-ve) elements.

i

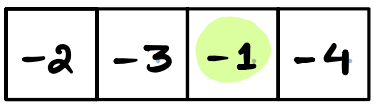
2	3	-4	6

Sum 0	2	5	1	7
Ans INT_MIN	2	5	5	7

i

2	3	-6	7

Sum 0	2	5	-1 → 0	7
Ans INT_MIN	2	5	5	7



Sum 0

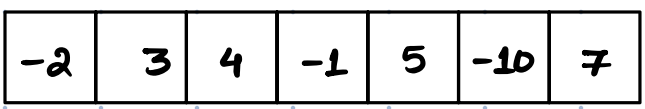
Ans INT_MIN

-2	-2	-1	-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

Ans INT_MIN

-2	3	7	6	11	1	8
-2	3	7	7	11	11	11

< / > Code

```
int maxSubarraySum(int arr[], int n){
    Sum = 0 , ans = INT_MIN;
    for(i=0; i<n; i++){
        Sum += arr[i];
        if(sum > ans)
            ans = sum;
        if(sum<0)
            sum = 0;
    }
    return ans;
}
```

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

ans = Math.Max(ans, sum);

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

BF Idea

For every query, iterate from index i to (N-1) & add value "x" to all the elements.

T.C : O(Q*N)

S.C : O(1)

Observations :

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
						2+3	2+3	2+3

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+2	1+2	1+2	1+2 +6	1+2 +6	1+2 +6+3	1+2 +6+3	1+2 +6+3

Optimised Approach:

Add x in arr[i] & then calculate prefix sum.

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		1	2		

psum[]:	0	3	3	4	6	6	6

< / > Code

```
for (i=0 ; i< queries.length ; i++){
    int index = queries[i][0];
    int x = queries[i][1];
     arr[index] += x ;
}

// Calculate psum[].
psum[0] = arr[0];
for( i=1 ; i< n ; i++){
    psum[i]= psum[i-1]+ arr[i]; // arr[i] = arr[i-1]+arr[i]
}
return psum[i];                 return arr[];
```

T.C : O(Q+N)

S.C : O(1)

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.

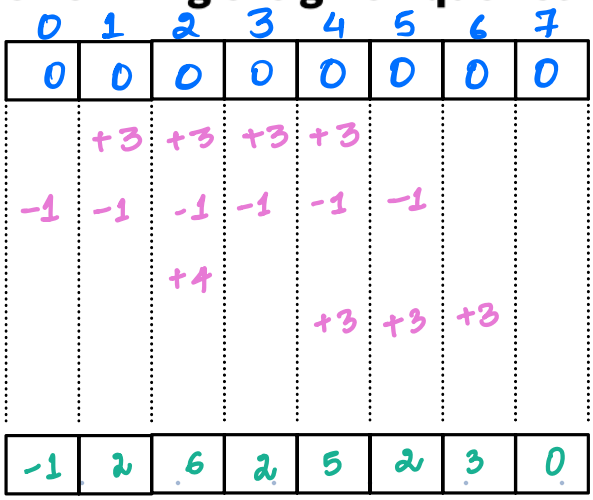
i j x

1 4 3

0 5 -1

2 2 4

4 6 3



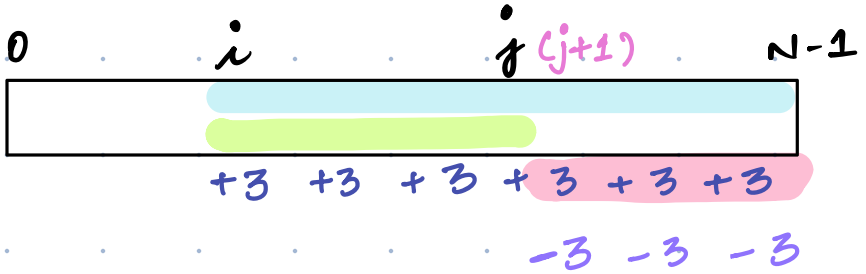
BF Idea

For every query, iterate from index i to j & add value "x" to all the elements.

T.C : O(Q*N)

S.C : O(1)

Observations :



Optimised Approach:

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

Calculate psum[].

N = 7 , Q = 3

Queries:

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

e == N-1

Nothing to add at (j+1)

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

+2 −2 +3 −3 −1

arr[]:

0	2	3	0	-2	-1	-3

psum[]:

0	2	5	5	3	2	-1

< / > Code

```
int RangeSum(int[] arr, int n, int[][] queries){

    for(q=0; q<queries.length; q++){
        int i = queries[q][0];
        int j = queries[q][1];
        int x = queries[q][2];

        arr[i] += x;

        if(j != N-1)
            arr[j+1] -= x;
    }

    // calculate psum
    for(i=1; i<n; i++){
        arr[i] = arr[i-1] + arr[i];
    }
    return arr;
}
```

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

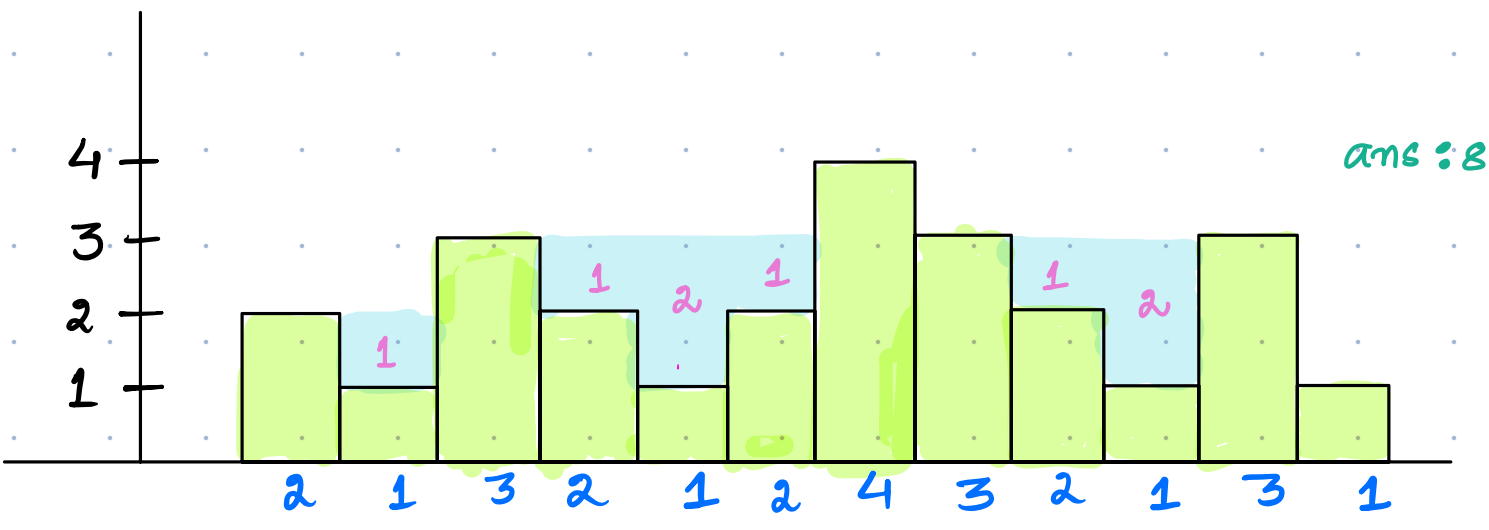
Problem: Trapping Rain Water

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

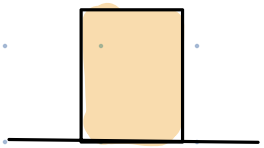
eg:

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

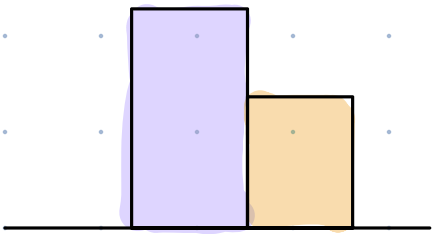
N=12



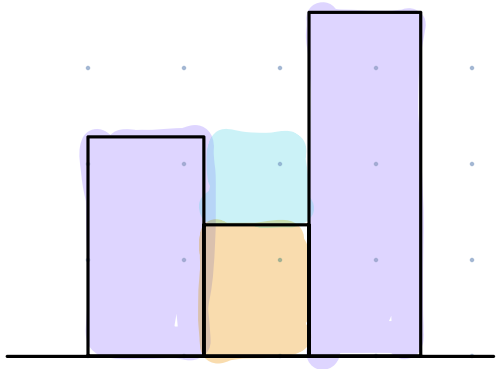
ans : 8



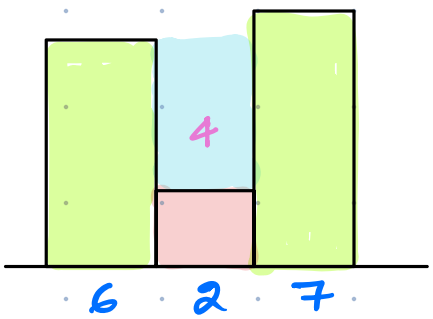
No



No



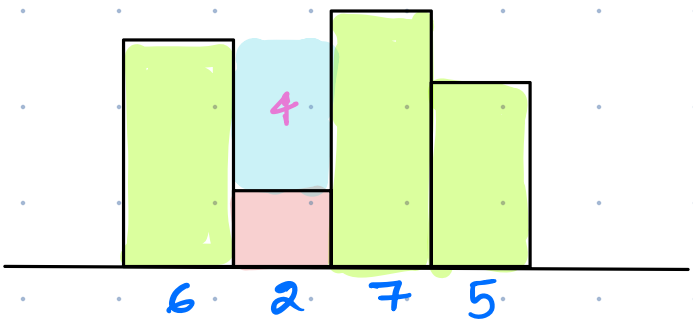
Water can be trapped if we have taller buildings on both sides.



l_{max} : 6

r_{max} : 7

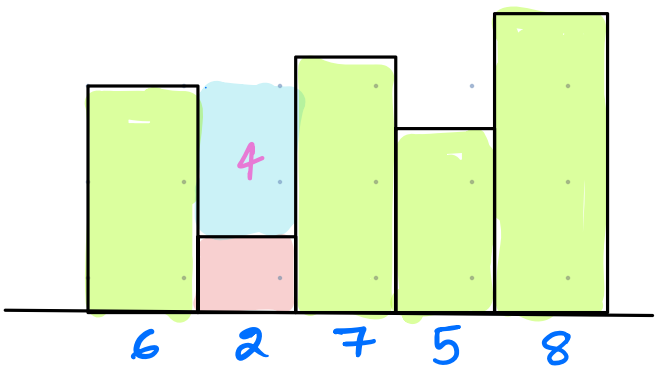
6 - 2 = 4



l_{max} : 6

r_{max} : 7

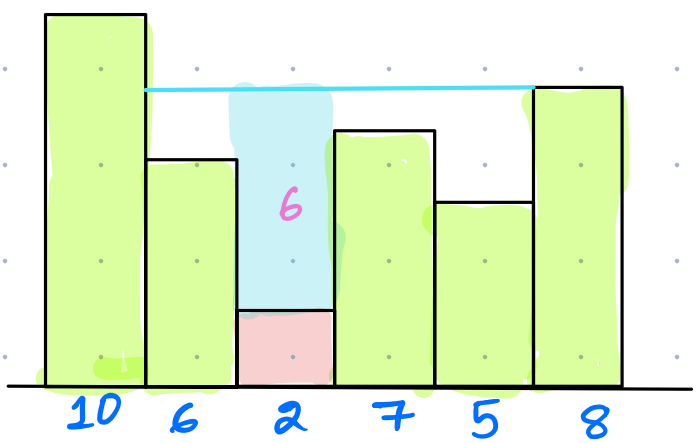
6 - 2 = 4



l_{max} : 6

r_{max} : 8

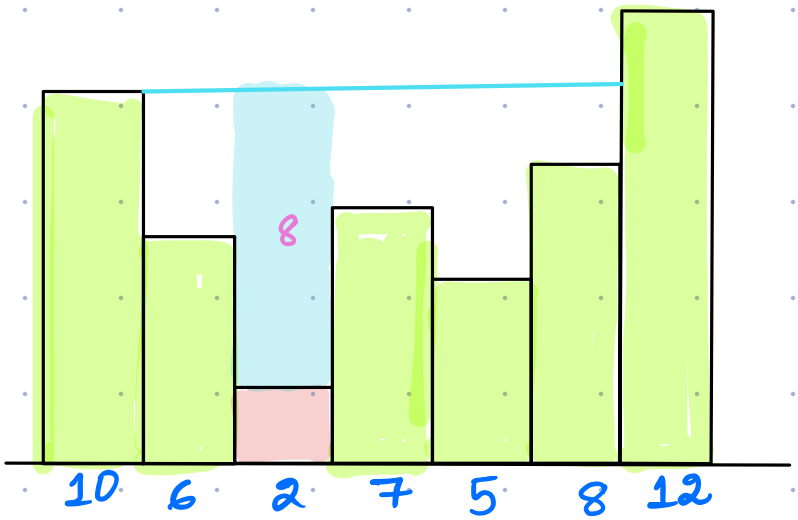
6 - 2 = 4



l_{max} : 10

r_{max} : 8

8 - 2 = 6

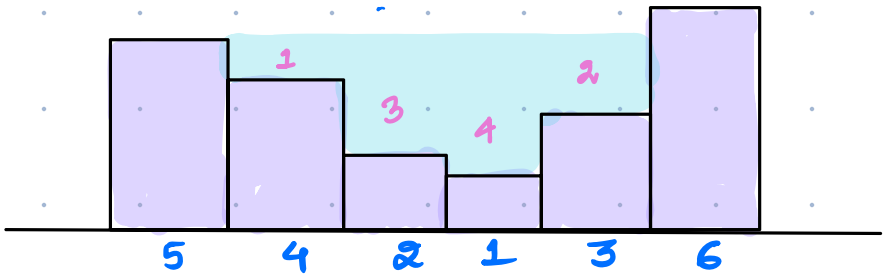


l_{max} : 10
r_{max} : 12

10 - 2 = 8

Approach:

Water Accumulated = Min(l_{max}, r_{max}) - ht of building.

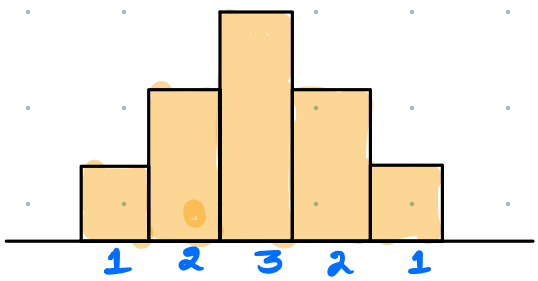


l_{max}	5	5	5	5	5	6
r_{max}	6	6	6	6	6	6
$Min(l_{max}, r_{max})$	5	5	5	5	5	6
Water on Building	0	1	3	4	2	0

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]`



l_{max}	1	2	3	3	3
r_{max}	3	3	3	2	1
$Min(l_{max}, r_{max})$	1	2	3	2	1
Water on Building	0	0	0	0	0

ans : 0

< / > Code

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

T.C : O(N²)
S.C : O(1)

Optimised Approach: Precompute lmax & rmax

lmax[n] rmax[n]

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

```
// Compute lmax
lmax[0] = h[0];
for(i=1; i<n; i++){
    lmax[i] = Math.Max(lmax[i-1], arr[i]);
}

// Compute rmax
    rmax[N-1] = h[N-1];
    for(i=N-2; i>=0; i--){
        rmax[i] = Math.Max(rmax[i+1], arr[i]);
    }

 ans = 0;
for(i=0; i<n; i++){
    water = Math.Min(lmax[i], rmax[i]) - h[i];
    ans += water;
}

return ans;
```