Suryansh Gupta
SWE-III at Google (L4)
2021 grad, I love problem solving, was former competitive coding.
Previously interned at Microsoft & Directi

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

$$A[\,] = \begin{array}{ccccccc} 0 & 1 & 2 & 3 & 4 & 5 & 6 \\ -2 & 3 & 4 & -1 & 5 & -10 & 7 \end{array}$$

-2 3 4 ✓
-2 4 ×
4 ✓
-2 3 4 -1 . . . 7 ✓

Max Sum : 11
Subarray : 3 4 -1 5

$$A[\,] = \begin{array}{ccccccc} 0 & 1 & 2 & 3 & 4 & 5 & 6 \\ -3 & 4 & 6 & 8 & -10 & 2 & 7 \end{array}$$

Max Sum : 18
Subarray : 4 6 8

Quiz 1: A[]: { 4 5 2 1 6 }
Ans : 18

Quiz 2: A[]: { -4 -3 -6 -9 -2 }

Ans: -2

Brute Force : Consider all the subarrays, find sum and take max.

l r => to represent a subarray

0	[0 N-1]	=> N
1	[1 N-1]	=> N-1
2	[2 N-1]	=> N-2
⋮		
N-1	[N-1]	=> 1

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

```
ans = A[0]
for( i=0; i<N; i++) {
    for ( j=i; j<N; j++) {
        int sum = 0
        for( k=i; k<=j; k++) {
            sum += A[k]
        }
        ans = max(ans, sum)
    }
}
return ans
```

TC: $O(N^2 * N)$
$\approx O(N^3)$

SC: $O(1)$

Prefix Sum: Use it for getting sum of each subarray.

TC: $O(N^2)$ SC: $O(N)$

Carry Forward:

```
ans = A[0]
for(i=0; i<N; i++) {
    sum = 0
    for(j=i; j<N; j++) {
        sum += A[j]
        ans = max(ans, sum);
    }
}
return ans
```

0	0
0	1
0	2
0	3
1	1
1	2
1	3
2	2
2	3
3	3

TC: $O(N^2)$ SC: $O(1)$

Case 1: All elements are positive

A[] = {1, 2, 3 . . . 8}

Ans: Sum of all elements

Case 2: All elements are negative

A[] = {-1, -2, -5, -3}

Ans: Largest element (smallest absolute value)

Case 3: If positive are present in between -ves.

A = [-ve -ve -ve -ve +ve +ve +ve +ve -ve -ve -ve]

Ans : Sum all positives.

Case 4 : If all positives are on corner.

A = [+ve +ve +ve +ve −ve −ve −ve −ve]

A = [−ve −ve −ve −ve +ve +ve +ve +ve]

Ans : Sum of all positive

Case 5 : Some +ve then −ve then +ve the −ve
Mix of +ve negative.

A[] = { −ve +ve −ve −ve −ve +ve +ve −ve −ve }

Relationship Example

$$A = \begin{array}{ccccccccc} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ -20 & 10 & -20 & -12 & 6 & 5 & -3 & 8 & -2 \end{array}$$

i	curSum	maxSum
0	−20	−20
curSum = 0		
1	10	10
2	10+(−20) = −10	10
curSum = 0		
3	−12	10
curSum = 0		
4	6	10
5	6+5 = 11	11
6	11−3 = 8	11

7	8+8 = 16	16
8	16-2 = 14	16

Kadane's Algo

Quiz 3 : { -2 3 4 -1 5 -10 7 }

cursum = ~~-2~~ ~~0~~ ~~3~~ ~~7~~ ~~6~~ ~~11~~ ~~1~~ 8

max = ~~-2~~ ~~3~~ ~~7~~ 11

```
int maxSumSubarray (int [] arr, int n) {
        int maxSum = arr[0] | INT_MIN
        int curSum = 0
        for (int i=0; i<n ; i++) {
              curSum += arr[i]
              maxSum = max(maxSum, curSum)
              if (curSum < 0) {
                    curSum = 0
              }
        }
        return maxSum
}
```

TC: O(N)

SC: O(1)

$$A = \begin{array}{cccc} 0 & 1 & 2 & 3 \\ [-5 & -2 & -8 & -1] \end{array}$$

i	curSum	ans
0	-5	-5
curSum = 0		
1	-2	-2
curSum = 0		
2	-8	-2
curSum = 0		
3	-1	-1

$$A = \begin{array}{cccc} 0 & 1 & 2 & 3 \\ (3 & 4 & -1 & 5] \end{array}$$

i	curSum	max
0	3	3
1	3+4= 7	7
2	7+(-1)= 6	7
3	6+5 = 11	11

Q: Min subarray ? ⟹ multiply -1 to all elements

Q: How to get the subarray ?

- Track curl, curr, maxl, maxr

Q2: Given an integer array A where every element is 0, return a final array after performing multiple queries.

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

$$\begin{array}{ccccccccc} & 0 & 1 & 2 & 3 & 4 & 5 & 6 & \\ \text{Example: } [& 0 & 0 & 0 & 0 & 0 & 0 & 0 &] \end{array} \quad N=7$$

Query (1, 3)
Query (4, 2)
Query (3, 1)

Index	0	1	2	3	4	5	6
Array	0	0	0	0	0	0	0
Q1		+3	+3	+3	+3	+3	+3
Q2					+2	+2	+2
Q3				+1	+1	+1	+1
Ans[]=	0	3	3	4	6	6	6

Brute Force: For each query, loop in the array and add x

TC: O(Q * N) SC: O(1)

Optimised : Note down / Mark queries and only go once in the end to update array

Index	0	1	2	3	4	5	6
Array	0	0	0	0	0	0	0
Q1		+3					
Q2					+2		
Q3				+1			
	0	3	0	1	2	0	0
	0	3	3	4	6	6	6

Lazy Sum

```
for(i=0; i< Q.size(); i++) {
        index = Q[i][0]
        val = Q[i][1]
        A[index] += val
}

for(i=1; i<N; i++) {
    A[i] = A[i] + A[i-1]
}
return A
```

TC : $(Q + N)$ SC : $O(1)$

Q3: Given an integer array A where every element is 0, return a final array after performing multiple queries.

Query(i, j, x) : Add x to all the elements from index i to j

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

Q1 = (1, 3, 2)

Q2 = (2, 5, 3)

Q3 = (5, 6, -1)

	0	1	2	3	4	5	6
	0	0	0	0	0	0	0
Q1		+2	+2	+2			
Q2			+3	+3	+3	+3	
Q3						-1	-1
	0	2	5	5	3	2	-1

Quiz: N = 8,

	i	j	x
Q1 =	1	4	3
Q2 =	0	5	-1
Q3 =	2	2	4
Q4 =	4	6	3

	0	1	2	3	4	5	6	7
	0	0	0	0	0	0	0	0
Q1		+3	+3	+3	+3			
Q2	-1	-1	-1	-1	-1	-1		
Q3			4					
Q4					+3	+3	+3	
	-1	2	6	2	5	2	3	0

Brute Force : For each query iterate i to j and add x.

TC : O(Q * N)   SC : O(1)

Optimize : Think about using lazy sum.

→ We want to stop after j

→ Can we do something to neutralise it?

$$\begin{tikzcd} & (\overset{i}{2} \quad \overset{j}{4} \quad \overset{x}{1}) \arrow[dl] \arrow[dr] & \\ (\overset{i}{2} \quad \overset{x}{1}) & & (\overset{i}{5} \quad \overset{x}{-2}) \end{tikzcd}$$

$$\begin{tikzcd} & (i \quad j \quad x) \arrow[dl] \arrow[dr] & \\ (i \quad x) & & (j+1 , \ -x) \end{tikzcd}$$

	0	1	2	3	4	5	6	7
	0	0	0	0	0	0	0	0
Q1		+3				-3		
Q2	-1						+1	
Q3			+4	-4				
Q4					+3			-3
	-1	3	4	-4	3	-3	1	-3
	-1	2	6	2	5	2	3	0

N = 8,

	i	j	x
Q1 =	1	4	3
Q2 =	0	5	-1
Q3 =	2	2	4
Q4 =	4	6	3

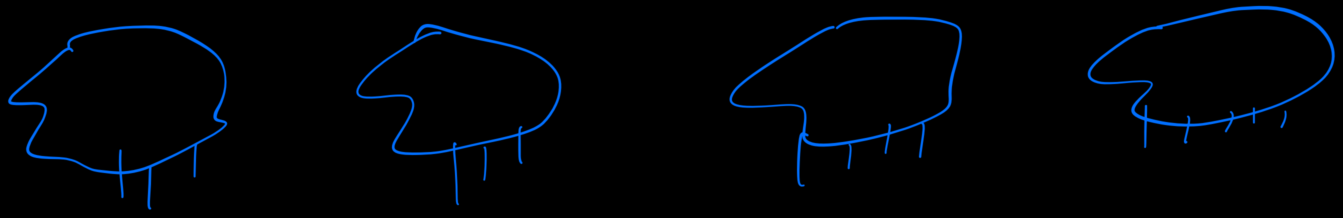
```
zeroQ ( int N, int start [], int end [], int val []) {
    int Q = start.size();
    int A[N] = {0};
    for( i =0; i< Q; i++) {
        int s = start [i] , e = end [i], v= val [i]
        A [s] += v
        if(e+1 < N) {
            A [e+1] -= v;
        }
    }
    for( i =1; i<N; i++) {
        A [i] = A [i] + A [i-1]
    }
    return A
}
```

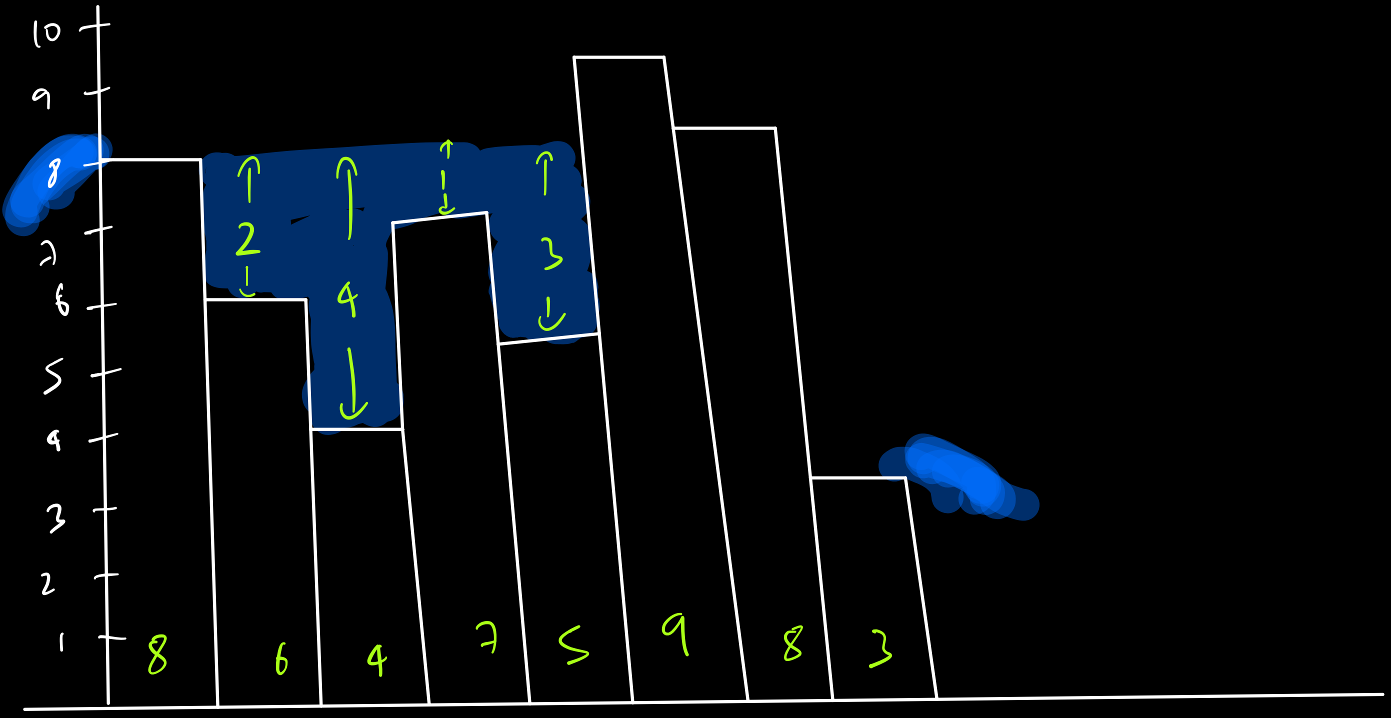
TC : O(Q+N)

SC : O(1)

Rain Water Trapping

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





Ans: 2 + 4 + 1 + 3 = 10

Obs: * Look at building 4

=) Look at tallest building on my left and right

left max = 8, right max = 9

=) The water will be upto minimum of left & right

* Look at building with height 9

leftmax = 8 , right max = 8

height of wat = min(8,8) = 8

water = min(leftmax, rightmax) - height
if(water < 0)
    water = 0

* Look at building with height 7

leftmax = 8 , righmax = 9

water = min(8, 9) − 7 = 1

Brute Force : For every building loop over and find left max and right max and calculate water.

```
ans = 0
for(int i = 1 ; i < N-1 i++){
  maxL = max(0 to i-1)   // Loop O(N) to find
  maxR = max(i+1 to N-1)   // Loop O(N) to find
  water = min(maxL, maxR) - A[i];
```

```
    if(water >0)
        ans += water

}
```

TC: $O(N^2)$

SC: $O(1)$

Optimize TC: Use prefix and suffix max array.

	0	1	2	3	4	5	6	7
	8	6	4	7	5	9	8	3
LMax	8	8	8	8	8	9	9	9
RMax	9	9	9	9	9	9	8	3
water	-	8-6	8-4	8-7	8-5	8-9	3-8	-
		2	4	1	3	-1	-5	
						‖	‖	
						0	0	

ans = 10

```
ans = 0
int lmax[N] = {0}
lmax[0] = A[0]
for(i=1; i<N; i++){
    lmax[i] = max(lmax[i-1], A[i]);
}
```

```
int rmax[N] = {0}
rmax[N-1] = A[N-1]
for(i=N-2; i>=0; i--){
    rmax[i] = max(rmax[i+1], A[i]);
}

for(i=1; i<N-1; i++){
    water = min(lmax[i-1], rmax[i+1]) - A[i]
    if(water > 0)
        ans += water
}
```

TC: $O(N)$ SC: $O(2N)$ we have prefix, suffix array

$\approx O(N)$

Carry Forward: We can carry forward one side and either remove prefix or suffix array. Other side we are bound to use arrays.

TC: $O(N)$ SC: $O(N)$

```
ans = 0
int rmax[N] = {0}

rmax[N-1] = A[N-1]
for(i=N-2; i>=0; i--){
    rmax[i] = max(rmax[i+1], A[i]);
}
int lmax = A[0]
for(i=1; i<N; i++) {
    water = min(lmax, rmax(i+1]) - A[i]
    if(water > 0)
        ans += water
    lmax = max(lmax, A[i])
}
```

If your interviewer still wants you to optimise to O(1) space, then use 2 Pointers

$$A[\,] = \begin{array}{ccccccccc} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ 6 & 4 & 3 & 5 & 2 & 4 & 7 & 3 & 4 \\ & & & & & & \uparrow & & \\ & & & & & & \uparrow & & \end{array}$$

lmax = 6
rmax = ~~4~~ 7

i	j	min(lmax, rmax)	Move	water	ans
0	8	4 < 6	j-1	4-4=0	0
0	7	4 < 6	j-1	4-3=1	0+1
0	6	6 < 7	i++	6-6=0	1
1	6	6 < 7	i++	6-4=2	1+2
2	6	6 < 7	i++	6-3=3	3+3
3	6	6 < 7	i++	6-5=1	6+1
4	6	6 < 7	i++	6-2=4	7+4
5	6	6 < 7	i++	6-4=2	11+2
6	6				

STOP

```
int n = A.size()
 int i = 0 , j = n-1
  int ans = 0
   int lmax = A[0], rmax = A[n-1]

   while(i < j) {
        if(lmax < rmax) {
             i++
        water = lmax - A[i]
        lmax = max(lmax, A[i]);
    }
    else {
        j--
        water = rmax - A[j]
        rmax = max(rmax, A[i]);
    }
    if (water > 0)
        ans += water
}
return ans;
```

TC: O(N)   SC: O(1)

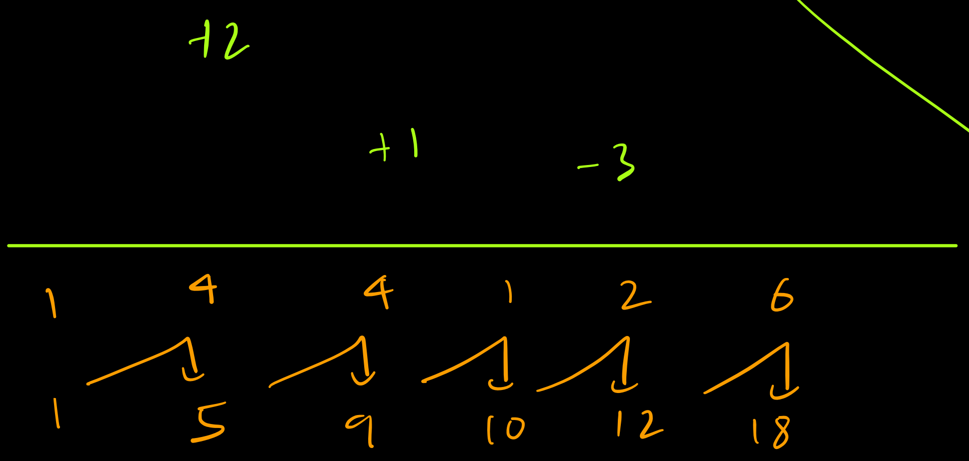
Doubts

Q) Can we do lazy sum with carry forward.

Not possible as we have to return each index.

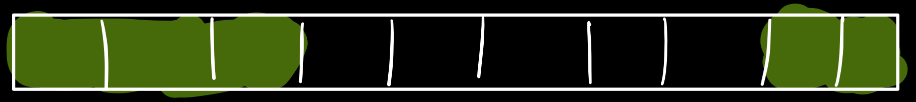
Q Lazy sum with non zero array.

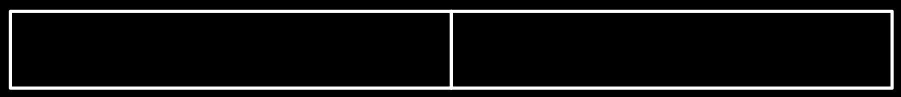
1   2   3   1   5   6



These were not meant to be carry forward.

Q Kadane with circular array.

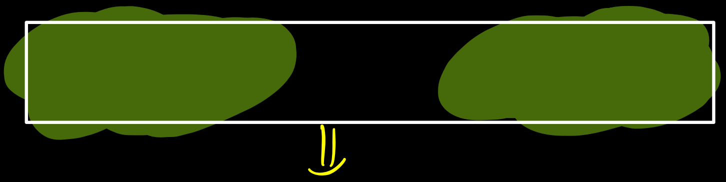




Apply Kadane ⇒ Only issue if we ever consider more than N elements

```
if (curLen == N) {
    curLen--;
    curSum -= A[i-N]
}
```

Approach :



max (Min Sum Subarray , Normal Kadane)