Today's Content

* Continuous Sum Query
* LeftMax[] & RightMax[]
* Rain Water Trapping
* Kadane's Algorithm [Max. Sum Subarray]

7022371038

Qn:

	0	1	2	3	4	5	6	7	8	9
ar[10] =	3	2	-1	5	6	8	2	3	2	6
pf[] =	3	5	4	9	15	23	25	28	30	36

sum[1 4] = 12 $\Rightarrow$ pf[4] - pf[0] = 15 - 3 = 12.
sum[3 6] = 21
sum[1 7] = 25

idea: Calculate prefix sum[]. - Sum of all elements from [0 i].

```
pf[0] = a[0]
for(i=1; i<n; i++) {
    pf[i] = pf[i-1] + a[i]
}
```

for Q queries:

```
while(Q-- > 0) {
    // get i, j as idx values
// (0,j) = (0,i-1) + (i,j)
// pf[j] = pf[i-1] + sum(i,j)

// sum(i,j) = pf[j] - pf[i-1]

    if(i != 0) {
        sum(i,j) = pf[j] - pf[i-1]
    }
    else {
        sum(0,j) = pf[j]
    }
}
```

TC: O(N) + O(Q)
SC: O(N): create new pf[]
O(1): changing input array.

Qn: Given a[n], where array elements are zero.
Given Q queries where each query has idx & value. Add the value from idx till end of array.

Q

idx	val
2	4
3	-1
0	2
4	1

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

idea: For every query, iterate & update the array.

```
while (Q-- > 0) {
    // Given idx & val
    for( i = idx ; i < n ; i++) {
        a[i] = a[i] + val
    }
}
```

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

idea2: Somehow propagate the value through every idx till end.

ar[5] = a_0 a_1 a_2 a_3 a_4

pf[] = a_0 a_0 a_0 a_0 a_0
$+a_1$ $+a_1$ $+a_1$ $+a_1$
$+a_2$ $+a_2$ $+a_2$
$+a_3$ $+a_3$
$+a_4$

Q

idx	val
2	4
3	-1
0	2
4	1

	0	1	2	3	4	5	6
a[7] =	0	0	0	0	0	0	0
(+)	2			4	-1	1	
	2	0	4	-1	1	0	0
pf[]:	2	2	6	5	6	6	6

idea 2 (summarized): * Add the given value, at the given idx for all queries.
* Create pfsum[] & return it.

```
while (Q-- > 0) {
    // Given idx & val
    a[idx] += val   // a[idx] = a[idx] + val
}
// Create pf array.
// a[0] → No change
for (i = 1; i < n; i++) {
    a[i] = a[i-1] + a[i]
}
return a
```

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

Eg: a[] = {1 2 3 4 5}

	0	1	2	3	4
		+2	+4		
	1	4	7	4	5
pf[]:	1	5	12	16	21

Q

idx	val
2	4
1	2

	0	1	2	3	4
	1	2	3	4	5
			4	4	4
		2	2	2	2
	1	4	9	10	11

Problem: 1, 2, 3, 4, 5 are also getting propagated.

Soln: * Create an empty array of size n & initialize it to 0
* Use above approach to fill this array & create pf[]
* Add this array to original array.

	0	1	2	3	4
a[] = {	1	2	3	4	5 }

③ temp pf [] = 0 2 6 6 6

Q

1	4	9	10	11

idx	val
2	4
1	2

	0	1	2	3	4
① temp[] =	0	0	0	0	0
		2	4		
	0	2	4	0	0

② temp pf [] = 0 2 6 6 6

[Google]

Qn: Given a[n], where array elements are zero.
Given Q queries where each query has start idx, end idx & value.
Add the value from start idx to end idx.

Q		
s	e	val
3	6	1
2	7	3
5	8	-3
1	9	2

	0	1	2	3	4	5	6	7	8	9
a[10]=	0	0	0	0	0	0	0	0	0	0
				1	1	1	1			
				3	3	3	3	3	3	
						-3	-3	-3	-3	
		2	2	2	2	2	2	2	2	2
	0	2	5	6	6	3	3	2	-1	2

Q		
s	e	val
3	6	1
2	7	3
5	8	-3
1	9	2

	0	1	2	3	4	5	6	7	8	9
a[10]=	0	0	0	0	0	0	0	0	0	0
		+2	+3	+1		-3		-1	-3	+3
	0	2	3	1	0	-3	0	-1	-3	3
pf[]:	0	2	5	6	6	3	3	2	-1	2

s	val	e+1	-val
3	1	6+1	-1
2	3	7+1	-3
5	-3	8+1	+3
1	2	9+1	-2

(don't do anything)

Generalize:

idx:	0	1	2	3 ...	s	s+1	e	e+1 ...	n-1
					v	v ...	v	v	v
								-v	-v

Query

s e v

add v : [s n-1]
add -v : [e+1 n-1]

Pseudocode:

```
while (Q-- > 0) {
    // Given s, e, val
    a[s] += val  // a[s] = a[s] + val
    if (e+1 < n) {   → Edge case.
        a[e+1] -= val
    }
}

// Create pf sum[]
```

TC: O(Q+n)

SC: O(1)

Qn: Create prefix max. array (LeftMax[] array)

	0	1	2	3	4	5
a[6] = {	1	-6	3	2	8	7 }
pfmax[]:	1	1	3	3	8	8

pfmax[i] = max of all elements from [0 i]

```
pf[0] = a[0]
for(i=1; i<n; i++) {
|       pf[i] = max(pf[i-1], a[i])
}
```

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

Qn: Create suffix max. array (RightMax[] array)

	0	1	2	3	4	5	6
a[7] = {	3	10	6	7	0	2	-1 }
sfmax[]:	10	10	7	7	2	2	-1

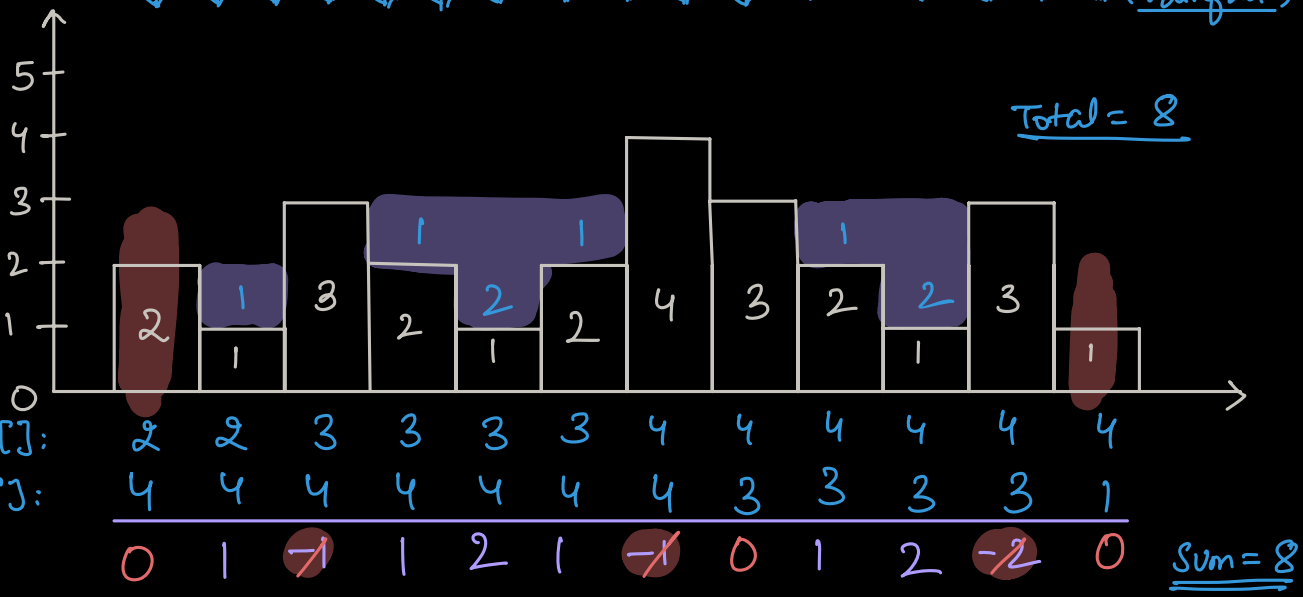
sfmax[i] = Max. of all elements from i to n-1
[i n-1]

Code: To do

Qn: Rainwater Trapping Problem

Given a[N] where a[i] represents height of the 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 }

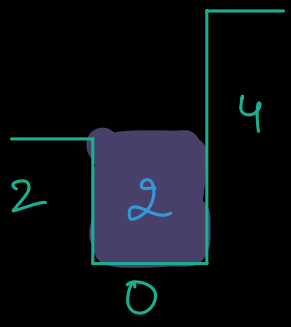


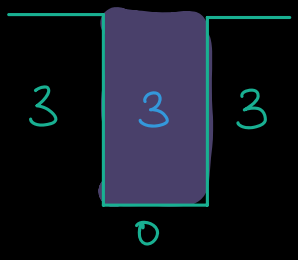
pfmax[]:	2	2	3	3	3	3	4	4	4	4	4	4
sfmax[]:	4	4	4	4	4	4	4	3	3	3	3	1
	0	1	~~-1~~	1	2	1	~~-1~~	0	1	2	~~-2~~	0

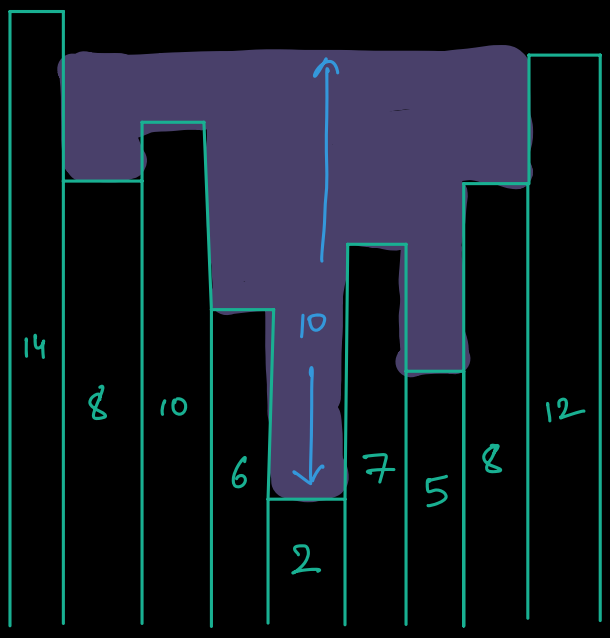
Sum = 8

idea: Calc sum of water accumulated on each building

Eg:







min (max left H, max right H) − own H

pfmax

sfmax

Pseudo code

```
int rainwater(int a[n]) {
    ans = 0
    // Compute pfmax[] & sfmax[]
    for (i=1; i<=n-2; i++) {
        lmax = {max of all H from 0...i-1} = pfmax[i-1]
        rmax = {max of all H from i+1...n-1} = sfmax[i+1]
        water = min(lmax, rmax) - a[i] // own height
        if (water > 0) {
            ans = ans + water
        }
    }
    return ans
}
```

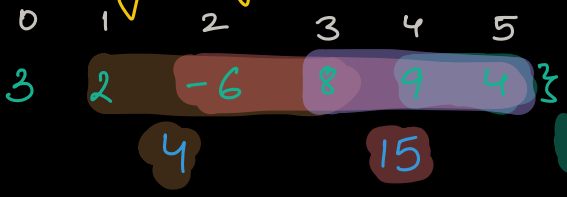
TC: $O(n+n+n) = O(n)$

SC: $O(n+n) = O(n)$

[Adobe]

Qn: Given a[N] containing integers. Find max. subarray sum.

Eg: a[7]: { 3 2 -6 8 9 4 }



idea 1: Create all subarrays, calc their sum & take max.

TC: $N \times \frac{N(N+1)}{2} = O(N^2) * O(N) = O(N^3)$ SC: O(1)

idea 2: Get all subarray sum using pf sum[] technique

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

idea 3: Get all subarray sum using carry forward technique.

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

Kadane's Algorithm

Scenario 1: all elements ≥ 0

a[] = { 3 2 1 6 } sum of all elements is ans.

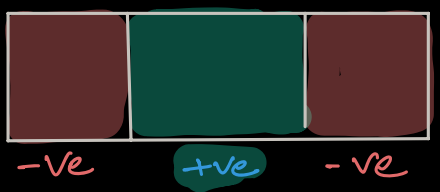
12

Scenario 2: all elements < 0

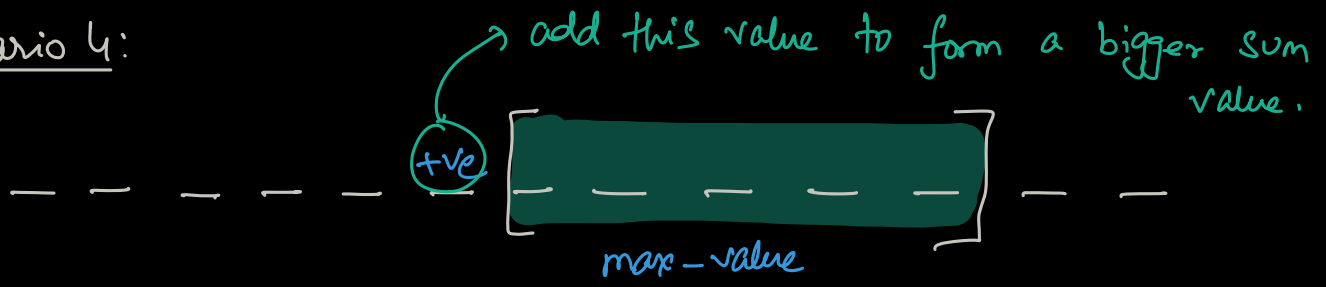
a[] = { -8 -4 -2 -10 } max. of entire array = ans

-2

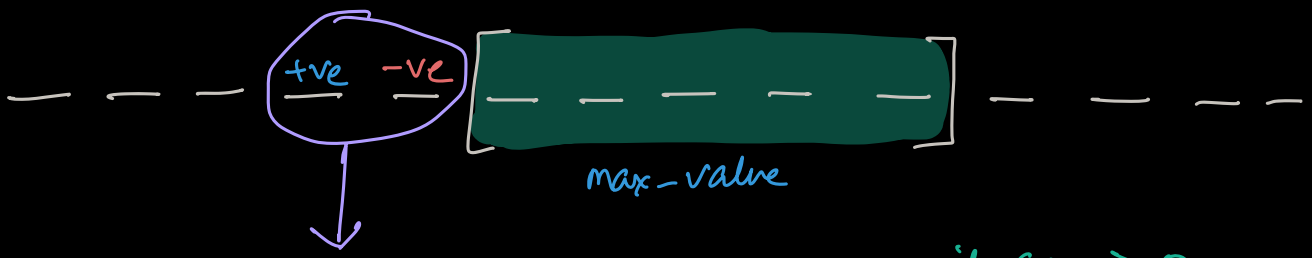
Scenario 3:



Scenario 4:



Scenario 5:



5 -3 ⇒ +2 ✓✓
5 -9 ⇒ -4 x
2 -6 ⇒ -4 x
4 -1 ⇒ +3 ✓✓

if sum > 0
carry it
else:
discard it

start ↓ , end ↙

ar[] = {	5	6	7	-3	2	-10	-12	8	12	21	-4	7 }
sum = 0	5	11	18	15	17	7	~~-5~~ 0	8	20	41	37	44
ans = -∞	5	11	18	18	18	18	18	18	20	41	41	44

Pseudo code :

```
int Kadane(int a[n]) {
    sum = 0
    ans = -∞  // int_min
    for(i = 0; i < n; i++) {
        sum += a[i]
        ans = max(ans, sum)
        if (sum < 0) {
            sum = 0
        }
    }
    return ans
}
```

TC: O(n)
SC: O(1)

Follow Up Qn: What if we want to know which subarray gave max sum ?

* Store idx whenever ans is getting updated

```
if (ans < sum) {
    ans = sum
    end = i
}
```

* Iterate back from end idx & calc sum. The moment sum == ans, that is start idx.