Lecture :- Arrays I

Agenda

- Max subarray sum.
- Beggars outside temple
- Rain water trapping.

Class starts at 8:35 PM

Qu1 Given arr[n], find max subarray sum.

arr[] = [-1 | 2 3 -4 6 9 2 -1 8 3]

28

arr[] = [-7 | 4 3 | -2 -8 -4 6 -2]

7

Brute force

Go to all subarrays — $O(n^2)$

for each subarray, calculate the sum — $O(n)$

and update your answer.

TC: $O(n^3)$

SC: $O(1)$

Approach 2

Prefix sum

↓

TC: $O(n^2)$

SC: $O(n)$

Carry forward

↓

TC: $O(n^2)$

SC: $O(1)$

Approach 3

TC: $O(n)$

SC: $O(1)$

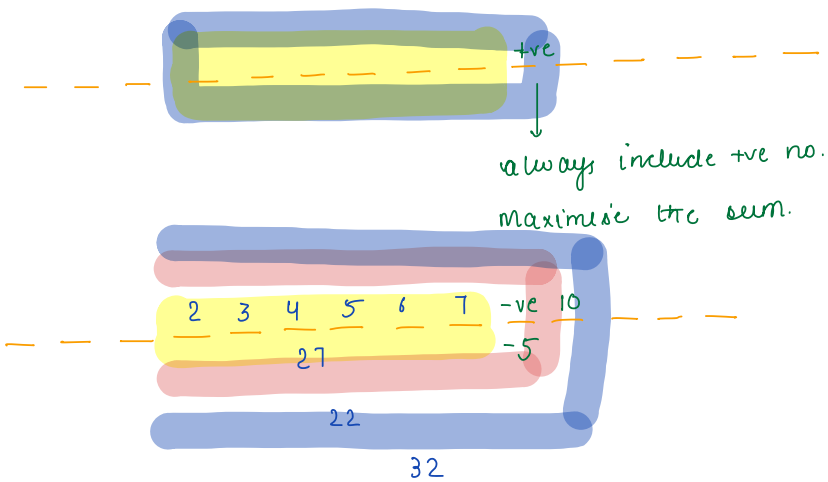
Case1: arr[] = [2 3 8 1 9 6] all +ve no.

Sum of array.

Case2: arr[] = [-2 -3 -8 -1 -9 -6] all -ve.

max el of array.

Case3: some +ve and -ve. [Kadane's Algorithm]



-ve :- It depends

if +ve > -ve , include the -ve no as well

Eg:

arr	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

Ans

arr	-20	10	-12	6	5	-3	8	9
sum[0]	~~-20~~ 0	10	~~-2~~ 0	6	11	8	16	25
ans [-∞]	-20	10	10	10	11	11	16	25

```
int maxsum(int[] arr) {
    int sum = 0;
    int ans = -∞;
    for(int el: arr) {
        sum += el;
        ans = max(ans, sum);
        if(sum < 0) {
            sum = 0;
        }
    }
    return ans;
}
```

TC: O(n)

SC: O(1)

H|w:- Given arr[n], find and return the subarray with max sum.

```
int[] maxsubarraysum(int[] arr) {


}
```

Qu2: Beggars outside temple

Given arr[n]. All el of array are zero.

Given Q queries {idx, value}

Add this value from idx till end.

Example:

q = 4

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

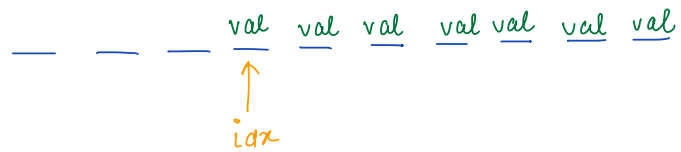
Brute force: Go to each query — O(Q)

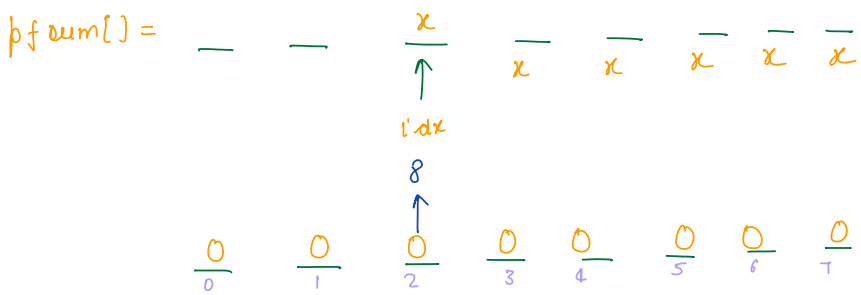
for each query, go and do the add^n of val from idx to end. O(n)

TC: O(Q * n) ≃ quadratic

Expected: TC: O(Q + n) ≃ linear.

SC: O(1)





query: [2, 8]

0 0 8 8 8 8 8 8 8 ⇒ ans.

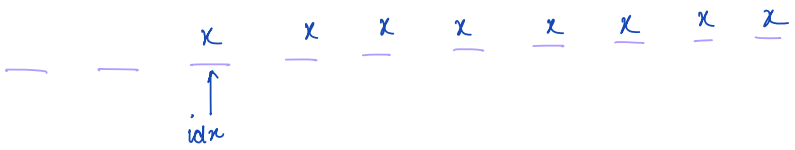
Example:

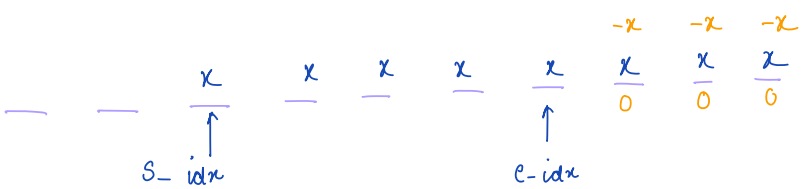
idx	val	0	0	0	0	0	0	0
		0	1	2	3	4	5	6
1	3	0	3	0	0	0	0	0
4	2	0	3	0	0	2	0	0
2	1	0	3	1	0	2	0	0
1	-1	0	2	1	0	2	0	0
2	3	0	2	4	0	2	0	0

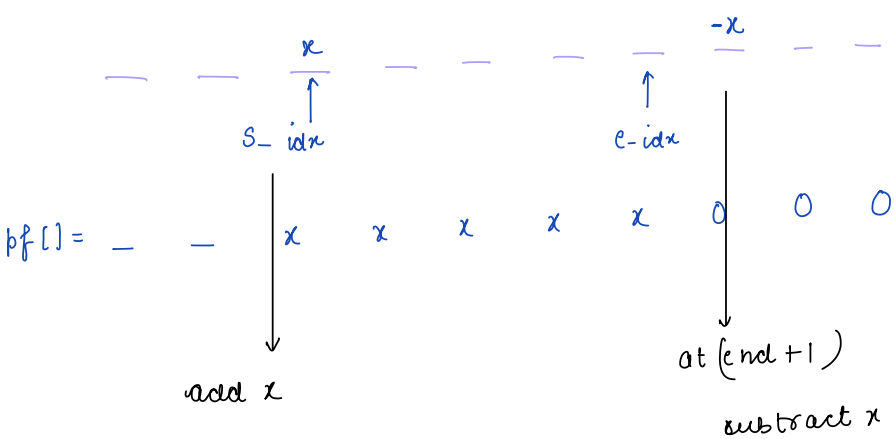
Extension: queries: { start, end, val }

start	end	val
2	4	2
1	3	1
0	2	3
3	5	4

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







start	end	val	0	0	0	0	0	0
			0	0	2	0	0	-2
2	4	2						
1	3	1	0	1	2	0	-1	-2
0	2	3	3	1	2	-3	-1	-2
3	5	4	3	1	2	1	-1	-2

3 4 6 7 6 4

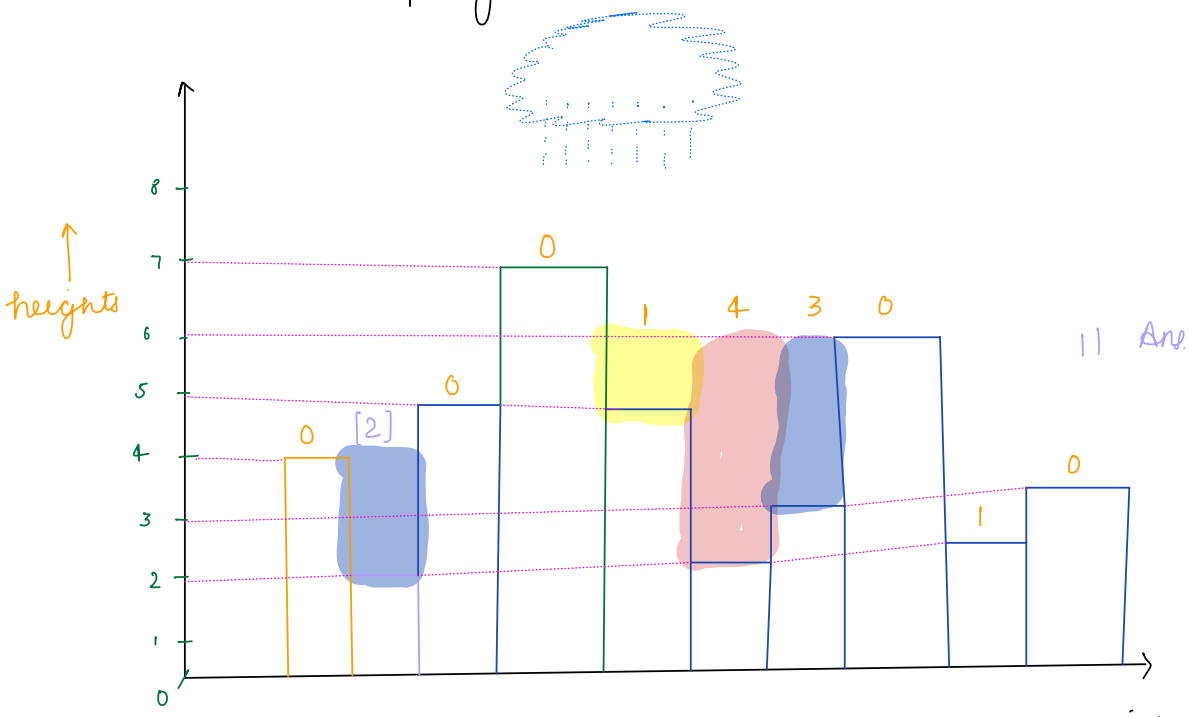
```
void beggarsOutsideTemple(int[] arr) {
                                int[][] queries
    for (int i=0; i < queries.length; i++) {
        int start = queries[i][0];
        int end = queries[i][1];
        int val = queries[i][2];
        arr[start] += val;
        if (end+1 < arr.length) {
            arr[end+1] -= val;
        }
    }

    // calculate prefix sum.
    for (i=1; i< arr.length; i++) {
        arr[i] = arr[i] + arr[i-1];
    }
}
```

TC: O(n+q)

SC: O(1)

Break: 10:00 PM

Q43 Rain water trapping



Given arr[n], each el of array represents buildings height.

calculate total water accumulated coz of rain.

Observation: figure of the max on left and right of each building.

lmax

rmax

water = min(lmax, rmax) − height of current building

Brute force: for every building — O(n)

calculate lmax & rightMax O(n)

TC: $O(n^2)$

Approach2:-

heights[]=	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

min(4,8) - 4
4 - 4
= 0

4 - 2
= 2

```
int rainwaterTrapped(int[] arr) {
    int n = arr.length;

    // find lmax[]
    int lmax[] = new int[n];
    lmax[0] = arr[0];
    for(i=1; i<n; i++) {
        lmax[i] = max(arr[i], lmax[i-1]);
    }

    // find rmax[]
    int rmax[] = new int[n]
    rmax[n-1] = arr[n-1];
    for(i=n-2; i>=0; i--) {
        rmax[i] = max(arr[i], rmax[i+1]);
    }
```

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

TC : O(n)

SC : O(n).

Q Can we reduce the space complexity?

yes [two pointers] Advanced module O(1)

Assignment:

lmax[] → carry forward

rmax[] → you can have space

save lmax[] space

Thank you