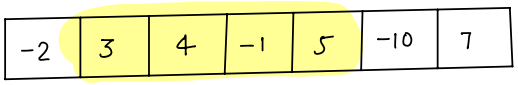
Lecture :- Arrays - 1

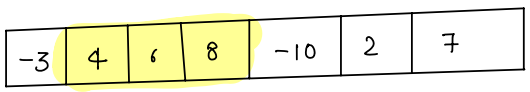
Agenda

- Kadane's Algorithm
- Beggars outside temple
- Rain water trapped

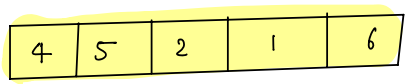
Qu-1 find max subarray sum.



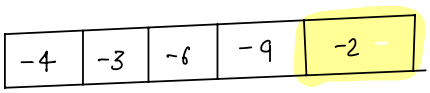
ans = 11



ans = 18



ans = 18



ans = -2

Brute force approach

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

TC: $O(n^3)$

SC: $O(1)$

Approach 2

```
int maxSumSubArray(int[] A) {
    int[] pf = getPrefixSumArray(A);
    ans = -∞;
    for(i=0; i<n; i++) {
        for(j=i; j<n; j++) {
            if(i==0) {
                sum = pf[j];
            } else {
                sum = pf[j] - pf[i-1]
            }
        }
    }
    return ans;
}
```

TC: $O(n^2)$

SC: $O(n)$

H|w: Carry forward. TC: $O(n^2)$

SC: $O(1)$

Approach 3

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

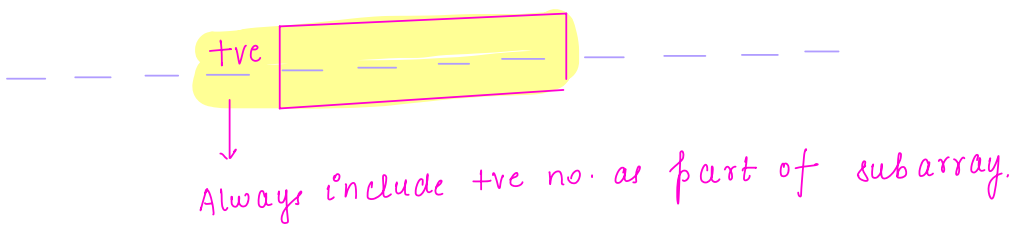
case1 :- All no. in array are positive.

ans = sum of array elements

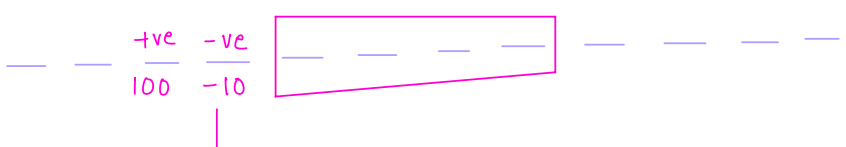
case2: All no. of array are negative.

ans = max element of the array.

case3 Some no. are +ve and -ve.



Always include +ve no. as part of subarray.



It depends

If +ve > -ve, include them as well.

A[] =	5	6	7	-3	2	-10	-12	8	12	21	-4	7
sum	5	11	18	15	17	7	~~-5~~ 0	8	20	41	37	44
max	5	11	18	18	18	18	18	18	20	41	41	44

A[] =	-20	-10	-6	-15	-2	-30
sum =	~~-20~~ 0	~~-10~~ 0	~~-6~~ 0	~~-15~~ 0	~~-2~~ 0	~~-30~~ 0
max = (-∞)	-20	-10	-6	-6	-2	-2

A[] =	-20	10	-12	6	5	-3	8	9
sum								
max								

Algorithm

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

kadane's Algorithm

TC: O(n)

SC: O(1)

H/W: find that subarray ?

```
int[] maxSubArray(int[] A) {

}
```

Beggars outside temple

Given arr[n]. All elements are zero initially.

Given Q queries { index, value }.

Add this value starting from 'idx' till end of array.

Queries = 4

idx	value.
1	3
4	2
2	1
1	-1

arr[7] =

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

Brute force:

for every query —

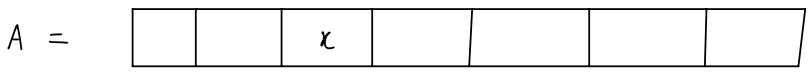
go and do the addition from idx till end of the array.

TC: O(n*q)

SC: O(1)

Expected TC: Linear

Intuition:



idx	value
1	3
4	2
2	1
1	-1

	0	1	2	3	4	5	6
	0	0	0	0	0	0	0
	0	3	0	0	0	0	0
	0	3	0	0	2	0	0
	0	3	1	0	2	0	0
	0	2	1	0	2	0	0
pf sum=	0	2	3	3	5	5	5

Algorithm

```
void beggarsoutsideTemple( int[] A,
                           int[][] queries) {

    for(i=0; i<queries.length; i++) {

        idx = queries[i][0]

        val = queries[i][1];

        A[idx] += val;

    }

    for(i=1; i<A.length; i++) {

        A[i] = A[i] + A[i-1];

    }

}
```

TC: O(n+q)

SC: O(1)

Extension of Qu.2

Given arr[n]. All elements are zero initially.

Given Q queries { start index, end index, value }

Add this value starting from 'idx' till end of array.

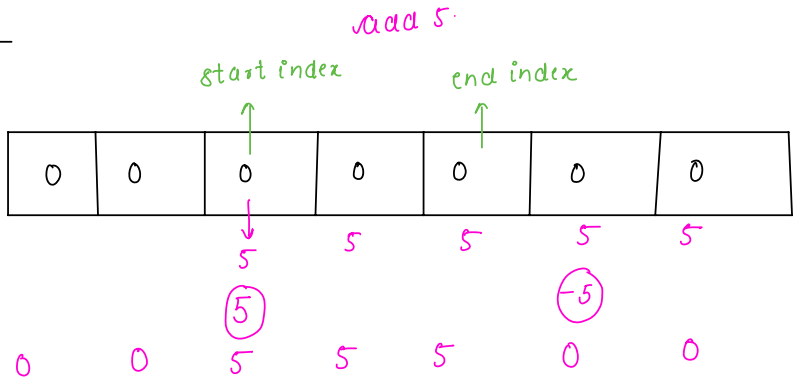
s_idx	e_idx	value
2	4	4
1	3	1
0	2	3
3	5	4

0	1	2	3	4	5
0	0	0	0	0	0
0	0	4	4	4	0
0	1	5	5	4	0
3	4	8	5	4	0
3	4	8	9	8	4

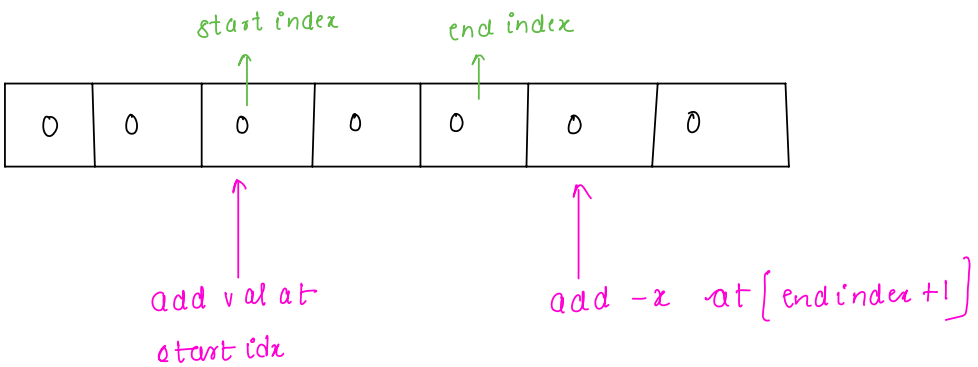
i	j	val
1	4	3
0	5	-1
2	2	4
4	6	3

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

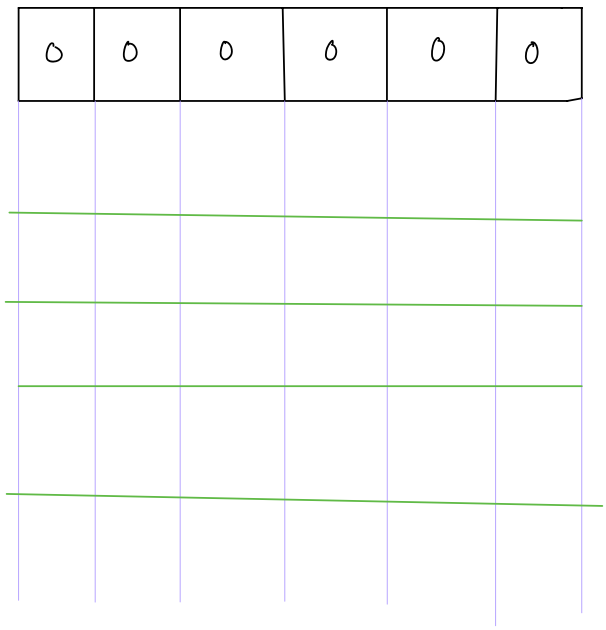
Approach



Previous approach



i	j	value
2	4	2
1	3	1
0	2	3
3	5	4



Algorithm

```
void beggarsoutsideTemple( int[] A,
                           int[][] queries) {

    for(i=0; i<queries.length; i++) {
        s = queries[i][0];
        e = queries[i][1];
        val = queries[i][2];
        A[s] += val;
        if(e+1 < A.length) {
            A[e+1] -= val;
        }
    }

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

TC: O(n+q)

SC: O(1)

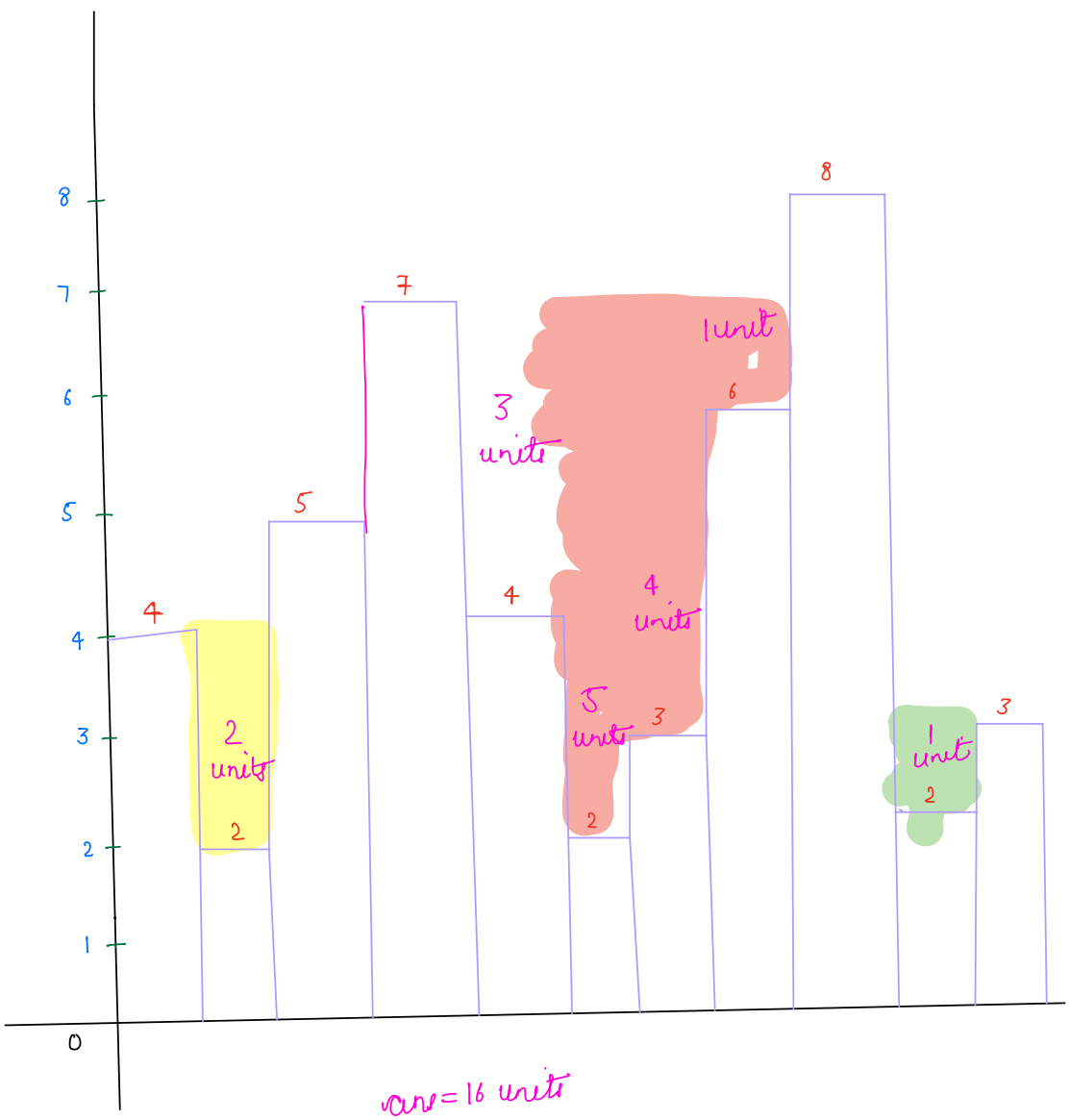
Break: 8:14 - 8:24 AM

Rain water trapping [Interview]

Given n buildings with height of each building.

find rain water trapped b/w each buildings.

height[] =	4	2	5	7	4	2	3	6	8	2	3



ans = 16 units

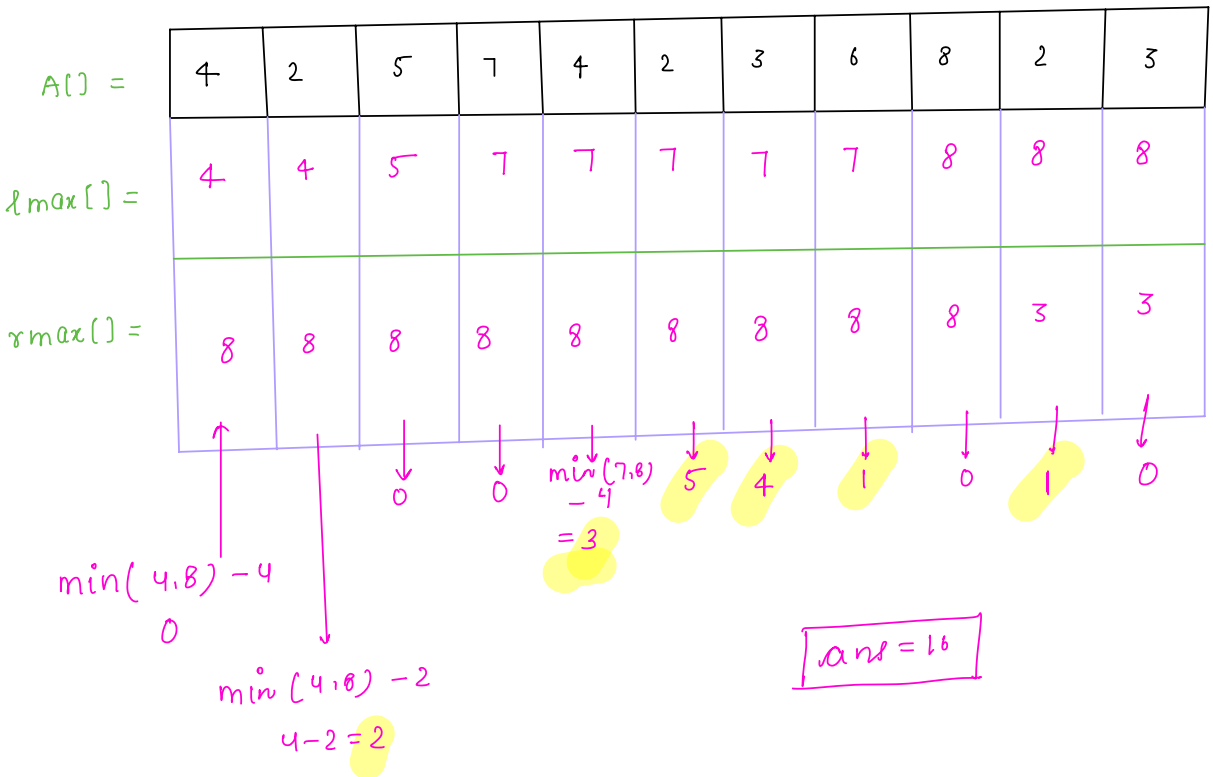
Brute force approach

```
int totalWaterTrapped(int[] A) {
        ans = 0;
O(n) ——— for(i=1; i<n-1; i++) {
    O(n) ——— maxL = max(0 to i-1);
    O(n) ——— maxR = max(i+1 to n-1);
        water = min(maxL, maxR) - A[i];
        if(water < 0) {
            water = 0;
        }
        ans = ans + water;
    }
    return ans;
}
```

TC: $O(n^2)$

SC: $O(1)$

Approach 2


A[] =
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
0
0
min(7,8) - 4 = 3
5
4
1
0
1
0
min(4,8) - 4
0
min(4,8) - 2
4-2 = 2
ans = 16

Algorithm

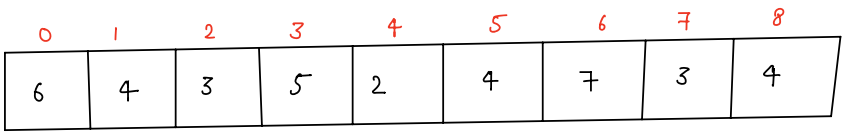
```
int countwaterTrapped(int[] A) {

    int[] lmax = getLeftMaxprefixArray(A);

    int[] rmax = getRightMaxprefixArray(A);

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

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

Approach 3

T.C. $O(n)$

S.C: $O(1)$



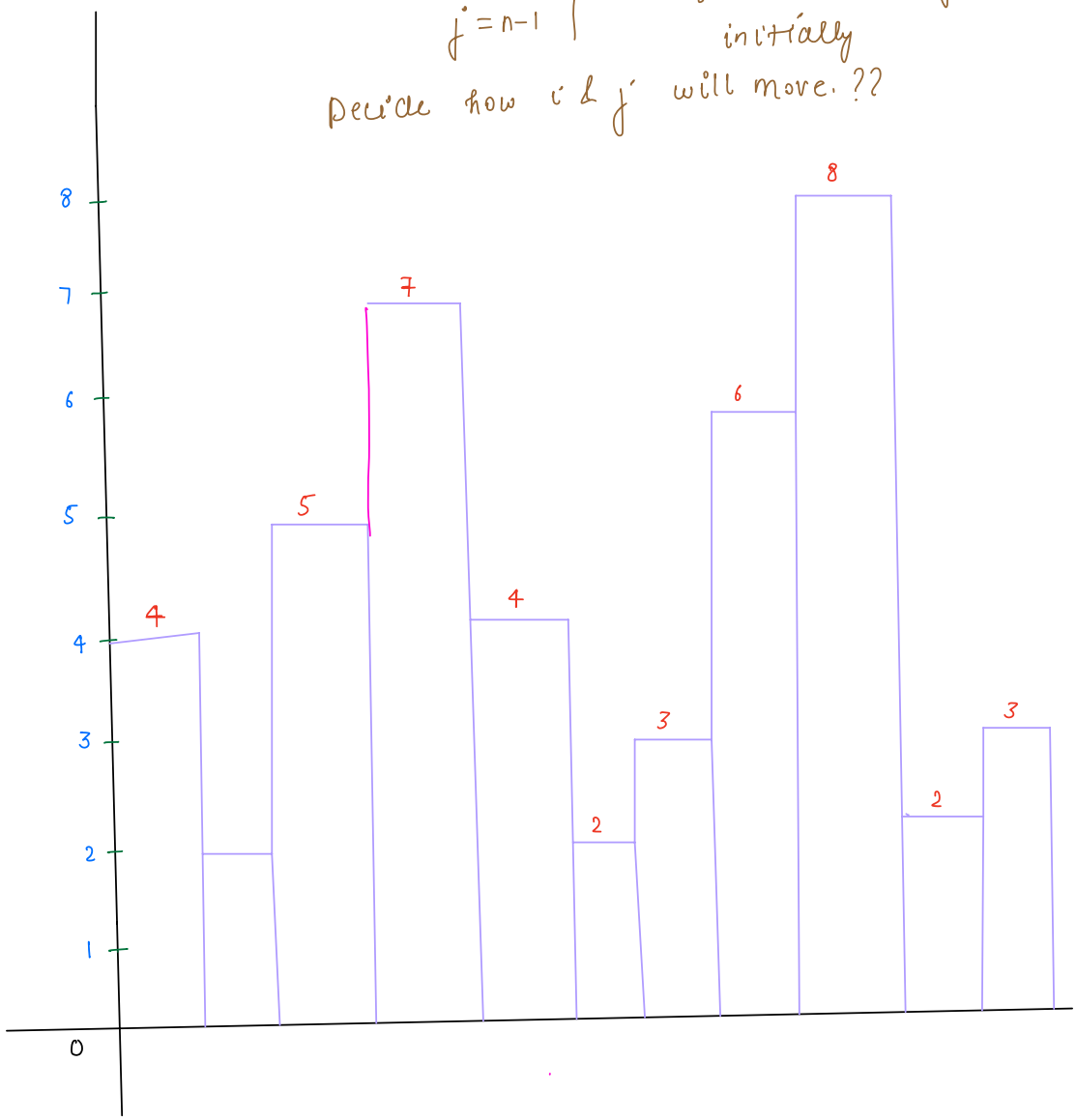
Dry run:

i	j	min (lmax, rmax)	move	ans

Hint: Consider 2 pointers —

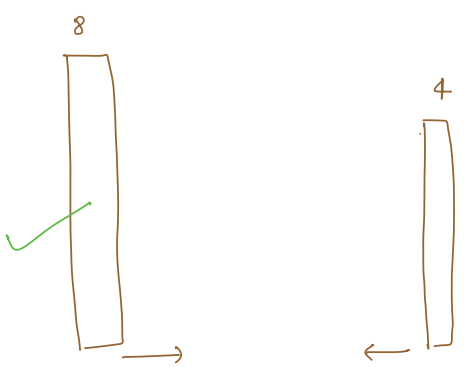
$i = 0$
$j = n-1$ } — tallest building initially

Decide how i & j will move. ??



lmax = 4

rmax = 3



Thankyou ☺