Lecture: Stacks - 2

Agenda

- Nearest Smaller element
- Largest rectangle in histogram
- Max - min sum for all subarrays

Qu! Given arr[n]. for every idx, find nearest smaller element on left of i, which is smaller than i.

Example:

	0	1	2	3	4	5
A[] =	4	2	5	10	8	2
	-1	-1	2	5	5	-1

	0	1	2	3	4
A[] =	6	10	11	12	7
	-1	6	10	11	6

	0	1	2	3	4	5	6
A[] =	4	6	11	7	8	3	5
	-1	4	6	6	7	-1	3

Brute force

```
int[] nearestSmaller(arr[]) {
    ans[n];
    for(i=0; i<n; i++) {
        int idx = -1;
        for(j=i-1; j>=0; j--) {
            if(arr[j] < arr[i]) {
                idx = j;
                break;
            }
        }
        ans[i] = idx;
    }
    return ans;
}
```

TC: $O(n^2)$

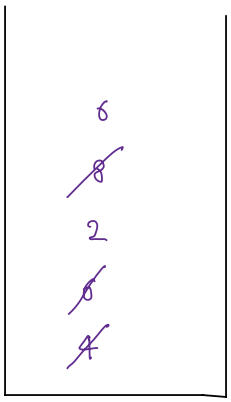
SC: $O(1)$

Approach 2 Stacks

Hint : put all possible candidate ans in our stack.

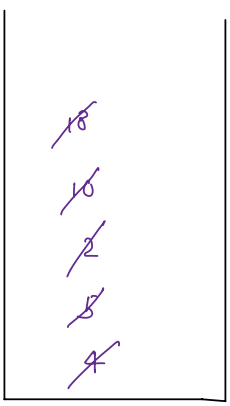
Example: 1.) A[] = [4 6 2 8 6]

nsl : -1 4 -1 2 2



2.) A[] = [4 5 2 10 18 2]

-1 4 -1 2 10 -1



Stack.peek()

< A[i]

ans = stack.peek();

Stack.push(A[i]);

>= A[i]

Stack.pop() till my stack is empty (or) stack.peek < A[i]

Stack.push(A[i]);

Code:

```
int [] nearestSmallerLeft( arr[]) {
        n = arr.length;
        Stack<Integer> stack = new Stack<>();
        ans[n];
        for( i=0; i<n; i++) {
            while( !stack.isEmpty()      &&
                        stack.peek >= arr[i]) {
                stack.pop();
            }
            if( stack.isEmpty()) {
                ans[i] = -1;
            } else {
                ans[i] = stack.peek();
            }
            stack.push[arr[i]];
        }
    return ans;
}
```

TC: O(n)

SC: O(n)

Modification

A[] =	0 4	1 5	2 2	3 10	4 18	5 2
Nearest smaller el on left	-1	4	-1	2	10	-1
Nearest smaller idx on left	-1	0	-1	2	3	-1

```
int [] nearestSmallerLeftIdx(arr[])
    n = arr.length;
    Stack<Integer> stack = new Stack<>();
    ans[n];
    for(i=0; i<n; i++) {
        while( !stack.isEmpty() &&
               arr[stack.peek()] >= arr[i]) {
            stack.pop();
        }
        if( stack.isEmpty()) {
            ans[i] = -1;
        } else {
            ans[i] = stack.peek();
        }
        stack.push(i);
    }
    return ans;
}
```

TC: O(n)

SC: O(n)

TODO (H|w)

1.) Get distance of nearest smaller on left side

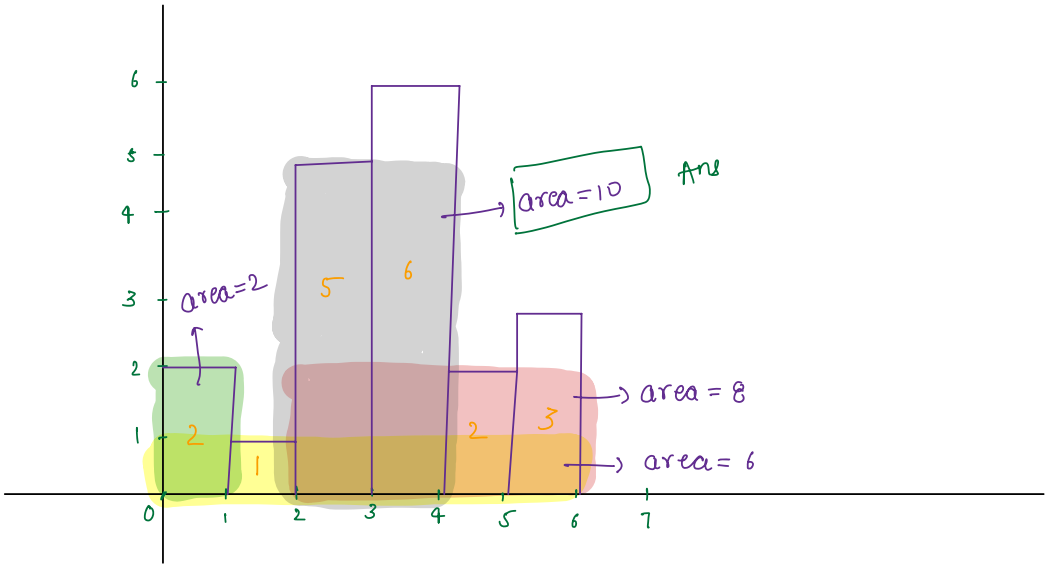
A[] =	0 4	1 5	2 2	3 10	4 18	5 2
Nearest smaller el on left	-1	4	-1	2	10	-1
Nearest smaller idx on left (dist)	-1	1	-1	1	1	-1

2.) find nearest smaller on right side

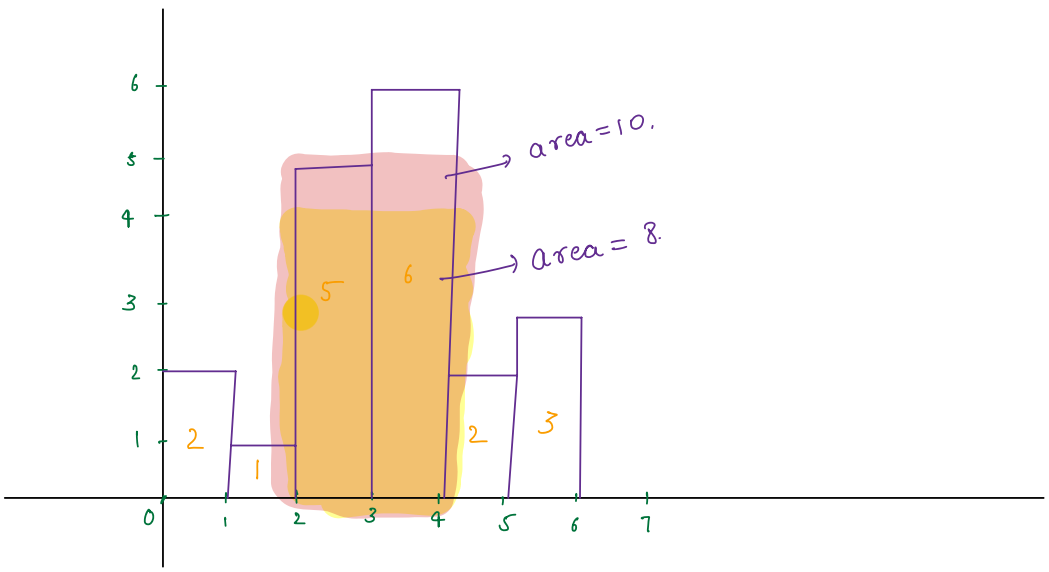
3.) find nearest greater on left

4.) find nearest greater on right

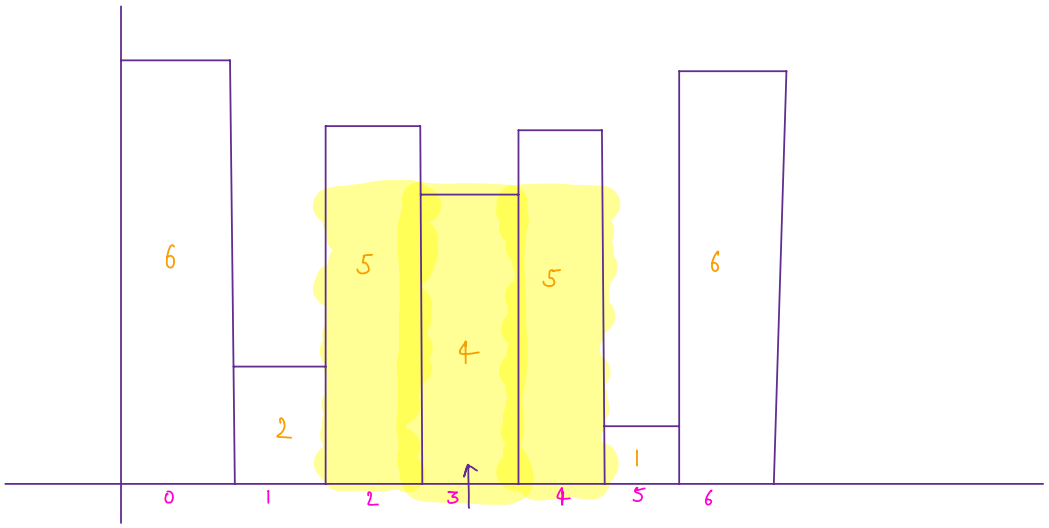
Qu 2 **Largest rectangle in histogram**



Claim: Max area :— 100% surely say, it will include height of any of the buildings.



Logic

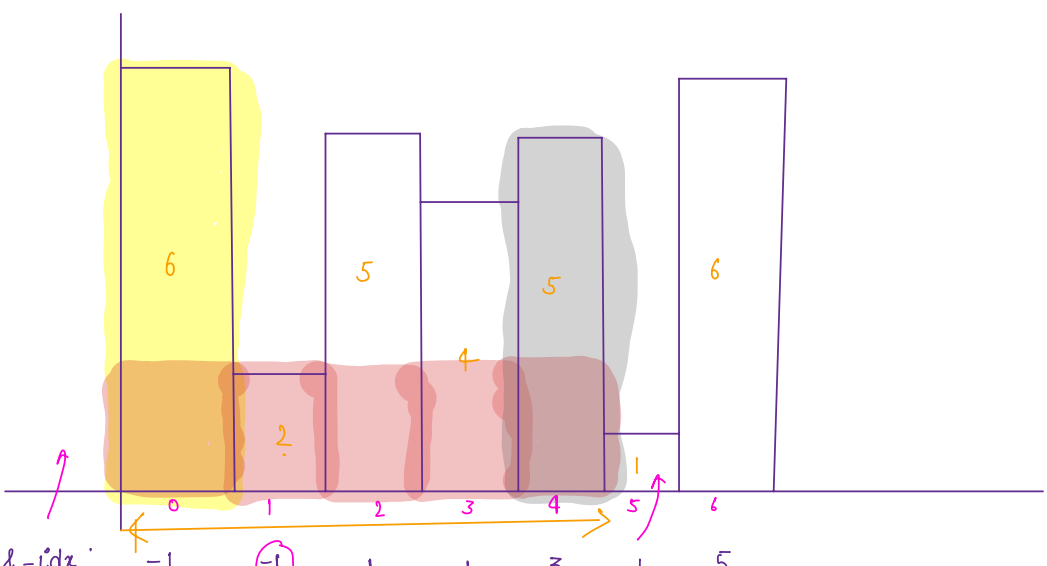


nsl_idx → 1st

nsr_idx → 5th.

area = height (arr[i]) * (width (5 − 1 − 1)) = 12 Ans
nsr nsl

Generalisation



nsl_idx : −1 | −1 | 1 | 1 | 3 | −1 | 5

nsr_idx : 1 | 5 | 3 | 5 | 5 | 7 | 7 → similar to −1.

area : 6 * (1 − (−1) − 1) = 6 | 2 * [5 − (−1) − 1] = 10 | 5 * [ ]

Code:

```
int largeetRectangle( arr[] ) {
    nsl[] = nearest smaller LeftIdx(arr);
    nsr[] = nearest smaller RightIdx(arr);
    area = 0;
    for( i=0; i<n; i++) {
        height = arr[i];
        width = nsr[i] - nsl[i] -1;
        area = max(area, height * width);
    }
    return area;
}
```

TC: O(n)

SC: O(n)

Break: 8:42 AM

Qu3 Given arr[n], find sum of (max - min) for all subarrays.

Example: arr[] = [2 (0) 5 (1) 3 (2)]

s	e	max	min	max - min
0	0	2	2	0
0	1	5	2	3
0	2	5	2	3
1	1	5	5	0
1	2	5	3	2
2	2	3	3	0
				8 Ans

Bruteforce: ∀ subarrays — $O(n^2)$

find max, min and add max - min — $O(n)$

to my final ans

return final ans;

TC: $O(n^3)$

SC: $O(1)$

arr[] = [2 (0) 13 (1) 8 (2) 4 (3) 1 (4) 5 (5) 3 (6) 6 (7) 2 (8) 7 (9)]

Qu In how many subarrays 5 will be maximum?

nearest Greater El idx on left :- 2nd idx (5)

4 1 5
3rd

1 5
4th.

nearest Greater El idx on right :- 7th idx.

5 3
6th

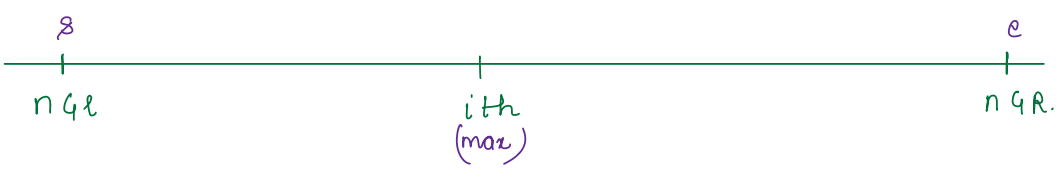
arr[] = [2 (0) 13 (1) 8 (2) 4 (3) 1 (4) 5 (5) 3 (6) 6 (7) 2 (8) 7 (9)]

In how many subarrays 6 will be max?

Left :- 2nd idx ⟶ 4 1 5 3 6
1 5 3 6
5 3 6
3 6

Right :- 9th idx
6 2

Generalisation



left :- [s+1 , i] = i − s − 1 + 1 = i − s.

right : [i , e−1] = e − 1 − i + 1 = e − i

ith idx is max = (i − s) * (e − i)
↑ nGl ↑ nGr

ith idx is min = (i - nsl) * (nsr - i)

	0	1	2	3	4	5	6	7	8	9
arr[] = [	2	13	8	4	1	5	3	6	2	7]

nsl = 4 th

nsr = 6 th

subarrays 5 is min = (5 - 4) * (6 - 5)

= 1.

5

ngl = 2nd

ngr = 7th

subarrays 5 is max = (5 - 2) * (7 - 5)

= 3 * 2 = 6

4 1 5

4 1 5 3

1 5

1 5 3

5

5 3

Contribution of arr[i] in my final ans —

arr[i] * # max - arr[i] * # min

arr[i] (# max - # min)

```
int findMinDiff (arr[]) {
    ans = 0;
    nSL[ ]
    nSR[ ]
    nGl[ ]
    nGr[ ]
    for ( i=0; i<n; i++) {
        noofmax = (i - nGl[i]) * (nGr[i] -i);
        noofmin = (i - nSl[i]) * ( nSr[i] -i);
        ans += arr[i] * (noofmax - noofmin);
    }
    return ans;
}
```

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

Thankyou ☺

Doubt

s	e	max	min
0	0	2	2
0	1	5	2
0	2	5	2
1	1	5	5
1	2	5	3
2	2	3	3

sum = 2 - 2 + 5 - 2 + 5 - 2 + 5 - 5 +

5 - 3 + 3 - 3

= 2 (1 - 3) + 5 (4 - 1) + 3 (1 - 2)

arr[i] #max #min

stack.pop();

temp = head;

head = head.next

temp.next = null.

3 ⟶ null

2 ⟶ 1