Today's content

1. Smallest element on left side
2. max area in a histogram
3. design stack with getMin functionality.

class content starts at 7:05
from next class.

Q. Given ar[N], for every index calculate 1st smaller element on left side.

Ex1:

	0	1	2	3	4	5
ar[6] :	4	5	2	10	3	2
ans :	-1	4	-1	2	2	-1

Ex2:

	0	1	2	3	4	5	6	7
ar[8] :	4	6	10	11	7	8	3	5
ans :	-1	4	6	10	6	7	-1	3.

Idea1: For every ar[i], iterate on left and get 1st smallest element.

```
int[] smallerOnLeft (int[] ar)
    int[] ans = -1.
    for (i=1; i<n; i++)
        // find 1st smaller on left.
        for (j=i-1; j>=0; j--)
            if (ar[j] < ar[i])
                ans[i] = ar[j]
                break
    return ans
```

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

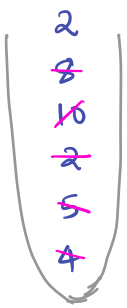
idea: Let's try using some DS, which one? --

Ex1:

	0	1	2	3	4	5
ar[6] :	5	2	8	10	6	1

Ex2 :

	0	1	2	3	4	5
ar[6] :	4	5	2	10	8	2



While inserting new element,

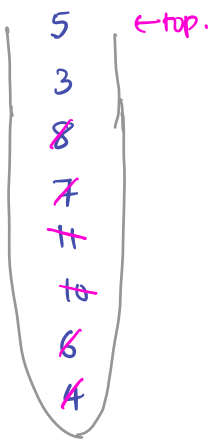
while top of stack is greater than or equal to new element, remove it.

top of stack is smaller ele on left.

If stack is empty => ans = -1.

Ex3:

	0	1	2	3	4	5	6	7
ar[8] :	4	6	10	11	7	8	3	5
	-1	4	6	10	6	7	-1	3




```
int[] smallerOnLeft(int[] ar)
    int[] ans = -1.
    Stack<Int> st
    i=0; i<n; i++
        while(st.size()>0 && st.top()>=ar[i])
            st.pop()
        if(st.size()>0)
            ans[i] = st.top()
        st.push(ar[i])
    return ans.
```

push pop.

TC: N + N => O(N).

SC: O(N).

Variations of 1st smaller ele on left.

(i) 1st smaller ele on right. ⇒ Traverse from right.

(ii) 1st smaller index on left and right.

Changes <
- instead of pushing ar[i], push 'i'.
- instead of st.top() ≥ ar[i], ar[st.top()] ≥ ar[i].

(iii) 1st greater ele on left and right.

changes for left ⇒ instead of st.top() ≥ ar[i] ⇒ st.top() ≤ ar[i]

Histogram area

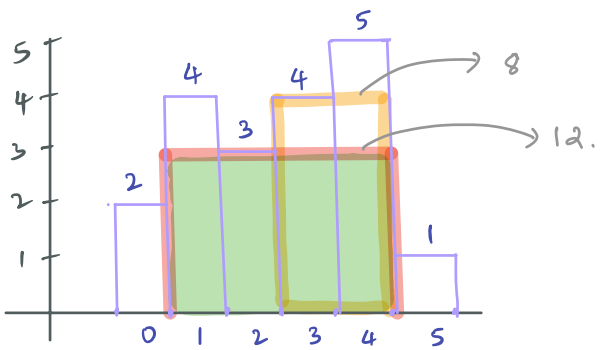
Given continuous blocks of histograms, find max rectangular area.

Note: Every histogram has a width of 1.

max rectangle that you can keep inside histograms.

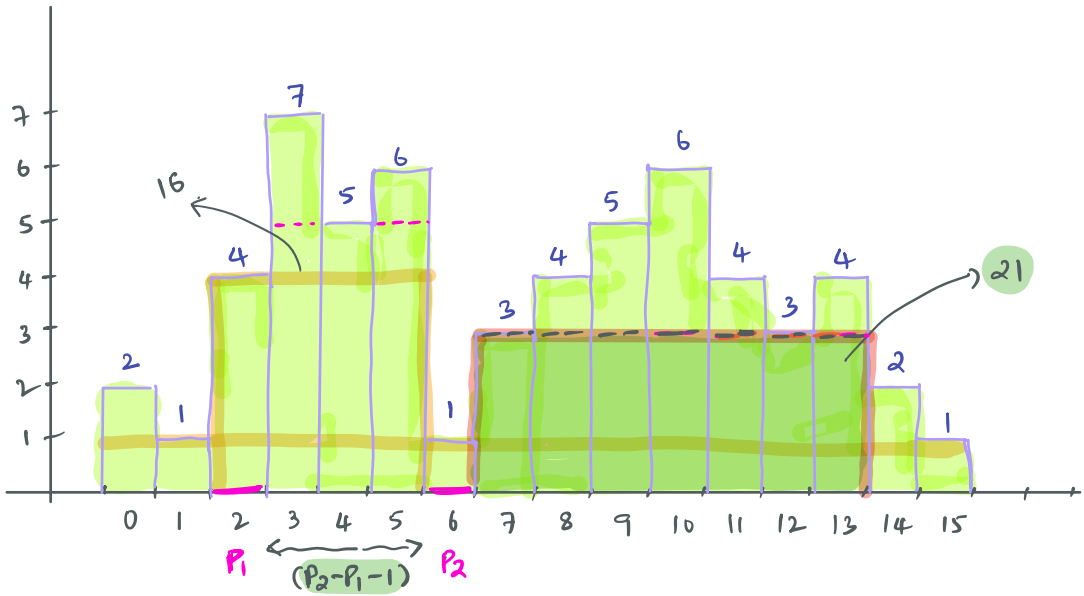
Ex1:

	0	1	2	3	4	5
ar[6] :	2	4	3	4	5	1



Ex2:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
ar :	2	1	4	7	5	6	1	3	4	5	6	4	3	4	2	1



$[P_1 \quad P_2] \rightarrow P_2 - P_1 + 1$

Excluding P_1 & P_2 => $P_2 - P_1 - 1$

Observations:

(i) Height of any rectangle will have to match with height of any one histogram.

(ii) Take every histogram height as height of rectangle and try to extend the rectangle on both the sides.

How to extend?

a) keep extending to left until we find height < rectangle height = P_1

b) keep extending to right until we find height < rectangle height = P_2.

P_1 → 1st smaller element on left.

P_2 → 1st smaller element on right.

(iii) Rectangle Area = $[P_2 - P_1 - 1] * height$.

Code:

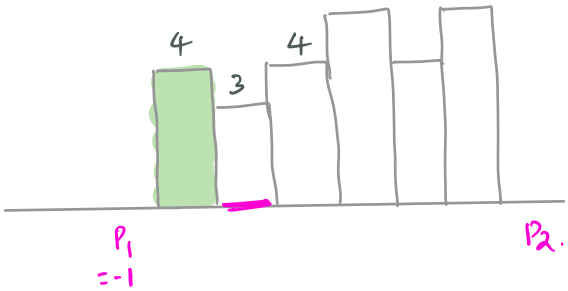
```
int rectangleArea(int[] hist, int n)
    // for every histogram, find smallest on left & right
    int[] left = smallestIndexLeft(hist)
    int[] right = smallestIndexOnRight(hist)
                                  // set default value for right as n
    ans = 0.
    for(i=0; i<n; i++)
        // consider every hist, hist[i] as height of rectangle.
        P1 = left[i], P2 = right[i]
        ans = max(ans, (P2 - P1 - 1) * hist[i])
    return ans
```

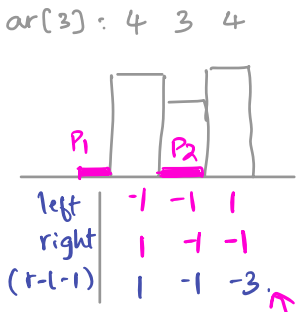
edge case.

3n iterations

Tc : O(n)

Sc : O(n)

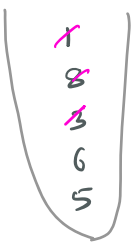




[Setting default value as (-1) for left will work, but not for right]

Q3: Design a stack that supports push, pop, top, getMin methods.
Note: The time complexity for all the methods should be O(1). in TC.

ex: push(5), push(6), push(3), push(8), getMin(), push(1), getMin(),
pop(), pop(), pop(), getMin().
3 1 5

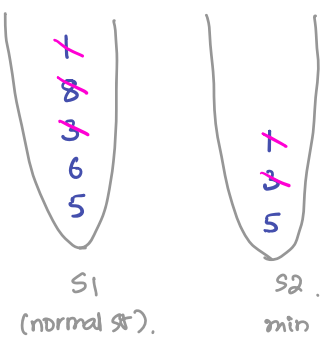


hint: Try using another stack to keep track of min.

(i) push(ele)
* Always push the ele to S1.
* Push the ele to S2 if ele ≤ S2.top.

(ii) pop()
* Pop out from S2 if S1.top = S2.top.
* Always pop out from S1.

(iii) getMin()
* Simply read S2.top.



Note: Equal elements must also be pushed to both S1 & S2 if the cond^n is satisfied.

```
class SpecialStack
    Stack s1 // normal stack
    Stack s2 // for min ele.

    void push(int ele)
        if(s1.size() == 0)
            s1.push(ele), s2.push(ele)
        else
            s1.push(ele)
            if(s2.size>0 && ele ≤ s2.top())
                s2.push(ele)

    void pop()
        if(s1.top == s2.top)
            s2.pop()
        s1.pop()

    int getMin()
        return s2.top.
```

[Possible to implement with O(1) space, TODO] → Optional.