Agenda

1) Nearest smaller on left
2) Other variations
3) Largest area histogram *
4) Sum of Max-min (tough) {remaining}

Q.1 Given A[], find nearest smaller on left for every element.

↳ distance

A =	[8	2	4	9	10	5	3	12]
	0	1	2	3	4	5	6	7
ans =	[-1	-1	2	4	9	4	2	3]

A =	[8	2	9	4	0	5	3]
	0	1	2	3	4	5	6
ans =	[-1	-1	2	2	-1	0	0]

i) Brute force idea: for every element, travel left side unless we get smaller than A[i].

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

```
ans = new int[n];
Arrays.fill(ans, -1);

for (i=0; i<n; i++) {
    for(j=i-1; j>=0; j--) {
        if (A[j] < A[i]) {
            ans[i] = A[j];
            break;
        }
    }
}
```

A = [2 4 1 7]
0 1 2 3

ans = [-1 2 -1 1]

Expected TC: O(n)

A = [8 2 4 9 1 5 3 12]
0 1 2 3 4 5 6 7

ans = [-1 -1 2 4 -1 1 1 3]

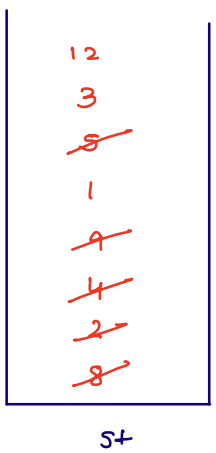
for every ele:

1) pop value >= A[i]

2) if st.size() == 0
   ans[i] = -1

   else
   ans[i] = st.peek()

3) st.push(A[i])



```
int[]  nextSmallerOnLeft ( int[] A) {
    int n= A.length;
    Stack<Integer> st = new Stack<>();
    int[] nsol = new int[n];  // nsol: next smaller on left

    for (int i=0; i<n; i++) {
        while (st.size()>0 && st.peek() >= A[i]) {
            st.pop();
        }
        if ( st.size()==0) {
            nsol[i] = -1;
        }
        else {
            nsol[i]= st.peek();
        }
        st.push(A[i]);
    }

    return nsol;
}
```

n pushes
n pop

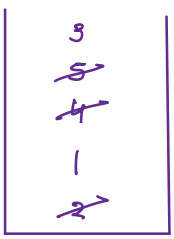
TC: O(n)

SC: O(n)

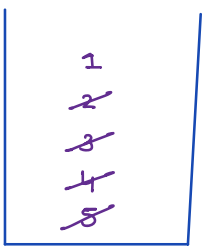
```
for (int i=0; i<n; i++) {
    while (st.size()>0 && st.peek() >= A[i]) {
        st.pop();
    }
    if (st.size()==0) {
        nsol[i] = -1;
    }
    else {
        nsol[i]= st.peek();
    }
    st.push(A[i]);
}
```

A = [2 1 4 5 3]
0 1 2 3 4

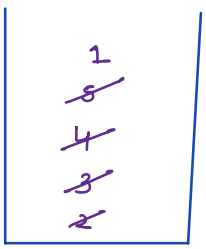
nsol = [-1 -1 1 4 1]



	5	4	3	2	1
nsol:	-1	-1	-1	-1	-1
itr:	1	1	1	1	1
		1	1	1	1



	2	3	4	5	1
nsol:	-1	2	3	4	-1
itr:	1	1	1	1	4
					1



todo: some more eg.

Q.2 Given A[], find index of nearest smaller on left for every element.

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

nsol = [-1 -1 1 2 2 2 1 6]

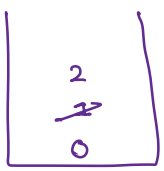
```
int[] nextSmallerOnLeft(int[] A) {
    int n = A.length;
    Stack<Integer> st = new Stack<>();
    int[] nsol = new int[n]; // nsol: next smaller on left

    for(int i=0; i<n; i++) {
        while(st.size()>0 && A[st.peek()] >= A[i]) {
            st.pop();
        }
        if(st.size()==0) {
            nsol[i] = -1;
        }
        else {
            nsol[i] = st.peek();
        }
        st.push(i);
    }

    return nsol;
}
```

A = [2 4 3]
0 1 2

nsol -1 0 0



Other variations

→ Nearest Smaller on left ✓

→ Index of nearest Smaller on left ✓

→ Nearest Smaller on right → run from i : n-1 to 0

→ Index of nearest Smaller on right → run from i : n-1 to 0

→ Nearest greater on left

→ Index of nearest greater on left

→ Nearest greater on right

→ Index of nearest greater on right

pop till stack has values <= A[i]

n-1 to 0 ↑

left / right → travel direction

↓ 0 to n-1

pop content on stack <= A[i] ↑

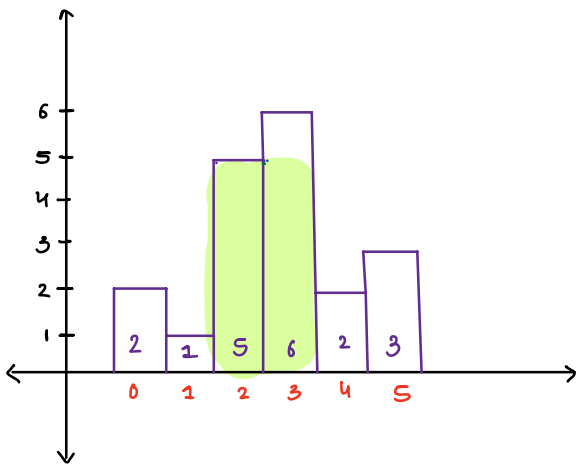
Smaller / greater → condition

↓ pop content on stack >= A[i]

Q.3 Largest area histogram *

Return the area of largest rectangle possible?

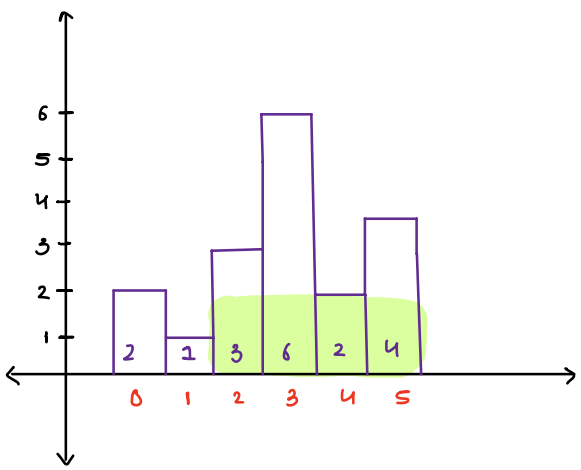
A = [2 1 5 6 2 3]



area = h * w

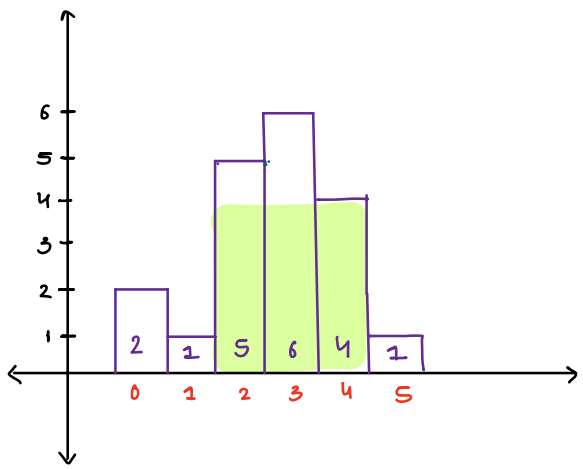
ans = 10 (5*2)

A = [2 1 3 6 2 4]

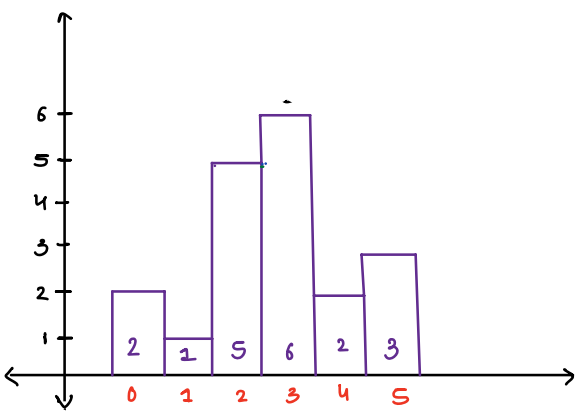


ans = 8

A = [2 1 5 6 4 1]



A = [2 1 5 6 2 3]



nsol	-1	-1	1	2	1	4
nsor	1	6	4	4	6	6
h	2	1	5	6	2	3
w	1	6	2	1	4	1
area	2	6	10	6	8	3

lb : nearest smaller on left.

rb : nearest smaller on right.

if nsol does not exists → -1

if nsor does not exists → n

h = A[i]

w = nsor[i] - nsol[i] - 1

Code :

```java
public class Solution {

    int[] nextSmallerLeft(int[]A) {
        int n = A.length;
        int[]nsol = new int[n];
        Stack<Integer>st = new Stack<>();

        for(int i=0; i < n;i++) {
            while(st.size() > 0 && A[st.peek()] >= A[i]) {
                st.pop();
            }

            if(st.size() == 0) {
                nsol[i] = -1;
            }
            else {
                nsol[i] = st.peek();
            }

            st.push(i);
        }

        return nsol;
    }

    int[] nextSmallerRight(int[]A) {
        int n = A.length;
        int[]nsor = new int[n];
        Stack<Integer>st = new Stack<>();

        for(int i=n-1; i >= 0;i--) {
            while(st.size() > 0 && A[st.peek()] >= A[i]) {
                st.pop();
            }

            if(st.size() == 0) {
                nsor[i] = n;
            }
            else {
                nsor[i] = st.peek();
            }

            st.push(i);
        }

        return nsor;
    }

    public int largestRectangleArea(int[] A) {
        int[]nsol = nextSmallerLeft(A);
        int[]nsor = nextSmallerRight(A);

        int ans = 0;

        for(int i=0 ; i <  A.length;i++) {
            int h = A[i];
            int w = nsor[i] - nsol[i] - 1;
            int area = h*w;

            ans = Math.max(ans,area);
        }

        return ans;
    }
}
```


itr : 3n

TC : O(n)

total space : 2n

SC : O(n)

Doubts

```
char ch1 = 'a';
char ch2 = '$';
```

ch1 == ch2

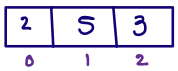
```
String s1 = "Nasrin";
String s2 = "Manisha";
```

s1 == s2 ×

↳ s1.equals(s2)

Q.u Given A[], find sum of all (max - min) for all subarrays.

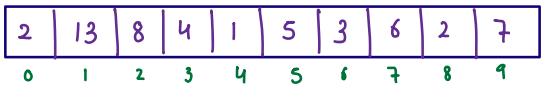
A =



s	e	subarray	(max - min)
0	0	[2]	
0	1	[2 5]	
0	2	[2 5 3]	
1	1	[5]	
1	2	[5 3]	
2	2	[3]	

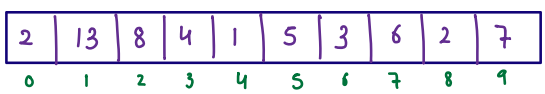
→ In how many subarrays 5 is max?

A =



→ In how many subarrays 6 is max?

A =



→ In how many subarrays 2 is min ?

A =