Agenda

i) Design min Stack

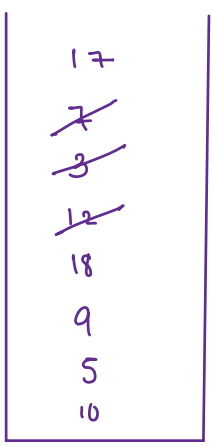
ii) Nearest Smaller **
- ↳ on left
- ↳ on right

iii) Largest Area histogram

Q.1 Design a Stack that supports push, pop, top and min functions in O(1) time.

10 5 9 18 12 3 7 17

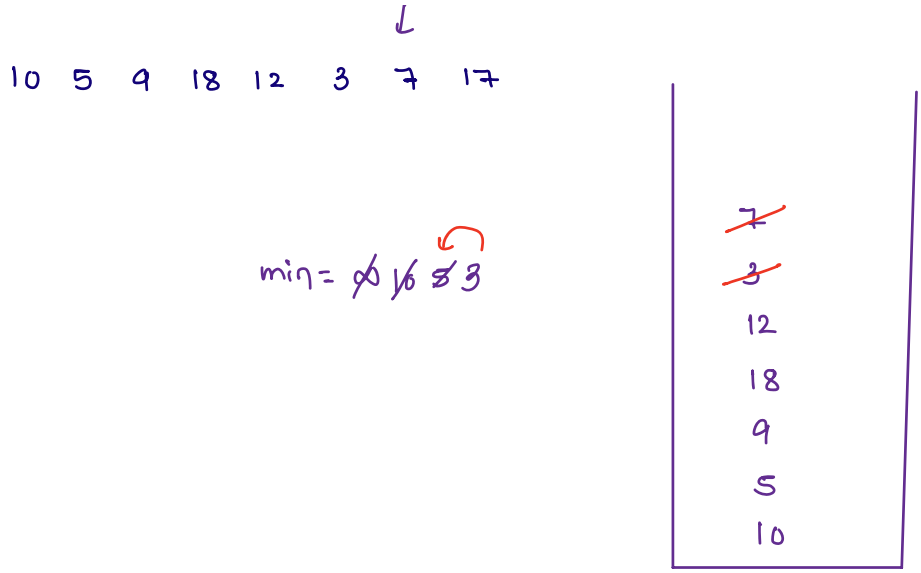
```
Minstack st = new Minstack();
st.push(10)
st.push(5)
st.push(9)
st.push(18)
st.push(12)
st.push(3)
st.push(7)
st.min() ⟶ 3
st.pop()
st.pop()
st.min() ⟶ 5
st.top() ⟶ 12
st.pop()
st.push(17)
st.top() ⟶ 17
```



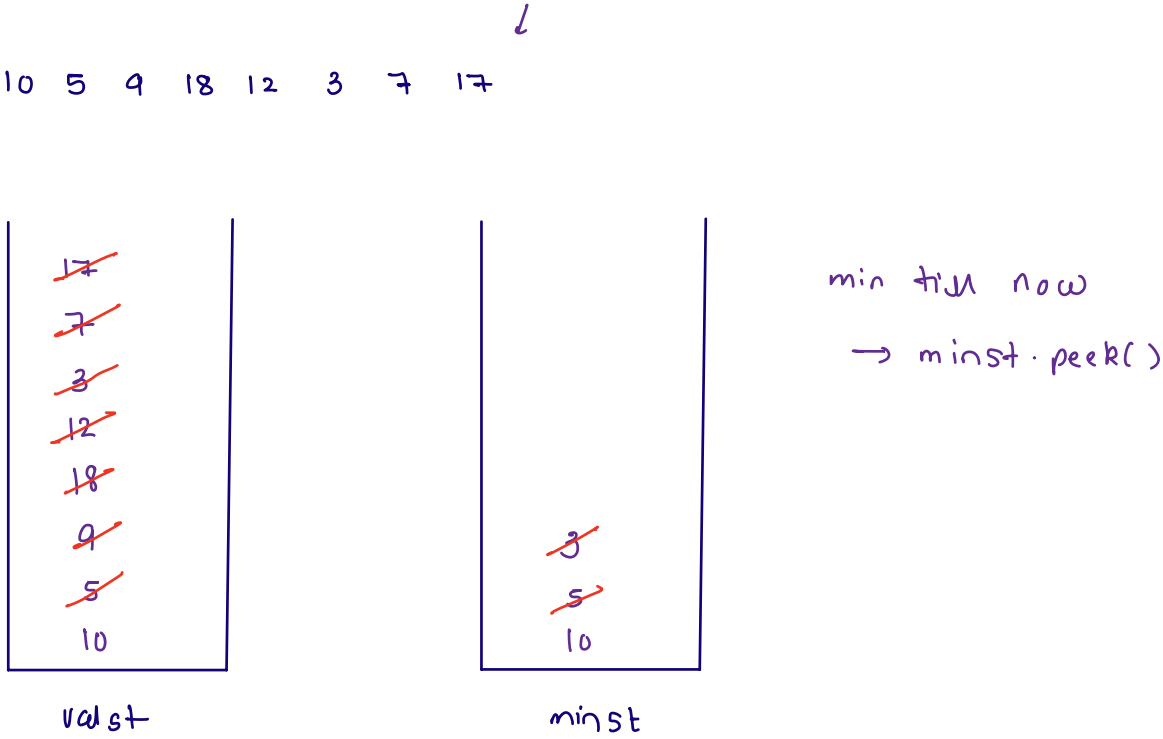

```
class Minstack {
    void push(int x)
    void pop()
    int top()
    int min()
}
```

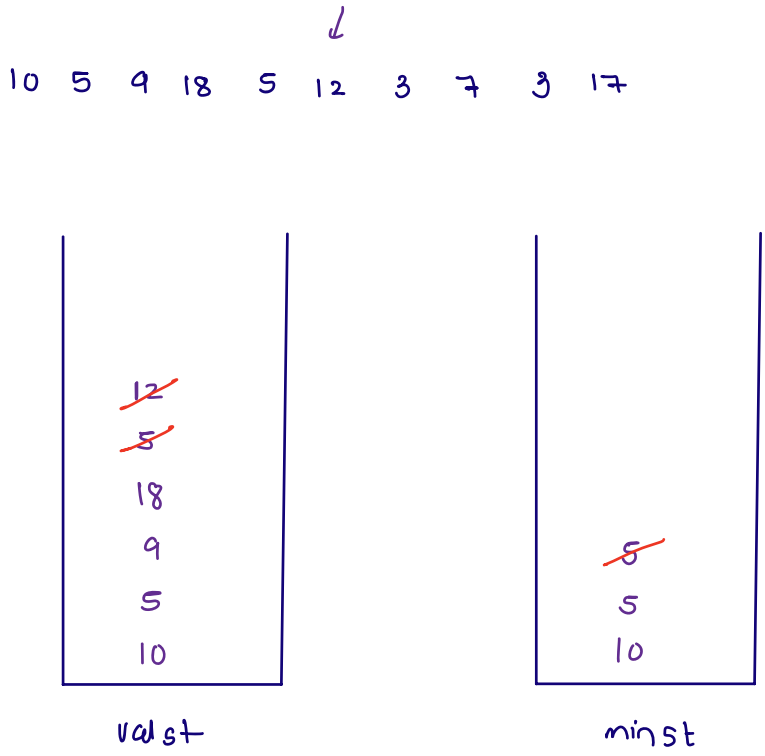
O(1)

5



Idea: Manage min by creating a seperate stack.




10 5 9 18 5 12 3 7 3 17
12
5
18
9
5
10
valst
5
5
10
minst

class MinStack {

→ two stacks { valst, minst }

void push (int x) {

→ maintain size o

→ add x to valst

→ add x to minst (x <= minst.peek())

}

void pop () {

→ maintain size o

→ pop from valst : (temp)

→ pop from minst (minst.peek() == temp)

}

int top () {

→ maintain size o

→ return valst.peek()

}

int min () {

→ maintain size o

→ return minst.peek()

}

}

```java
class Solution {

    Stack<Integer>valst = new Stack<>();
    Stack<Integer>minst = new Stack<>();

    public void push(int x) {
        if(valst.size() == 0) {
            valst.push(x);
            minst.push(x);
        }
        else {
            valst.push(x);

            //should I add x in minst as well
            if(x <= minst.peek()) {
                minst.push(x);
            }
        }
    }

    public void pop() {
        if(valst.size() == 0) {
            //nothing
        }
        else {
            int temp = valst.pop();

            //should I pop from minst as well
            if(temp == minst.peek()) {
                minst.pop();
            }
        }
    }

    public int top() {
        if(valst.size() == 0) {
            return -1;
        }
        else {
            return valst.peek();
        }
    }

    public int getMin() {
        if(valst.size() == 0) {
            return -1;
        }
        else {
            return minst.peek();
        }
    }
}
```

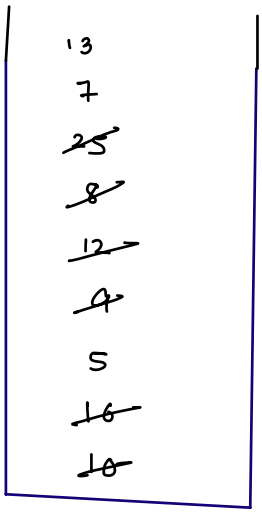
Q.2 Nearest / next smaller on Left.

A =	[	10	16	5	9	12	8	25	7	13]
nsol		-1	10	-1	5	9	5	8	5	7

A =	[	18	3	13	19	5	24	4]
nsol		-1	-1	3	13	3	5	3

Expected TC: O(n)

A =	[	10	16	5	9	12	8	25	7	13]
ans		-1	10	-1	5	9	5	8	5	7




```
for ( i → 1 to A.length-1 ) {
    while (st.size() > 0 && st.peek() >= A[i]) {
        st.pop();
    }
    if ( st.size() == 0 ) {
        ans[i] = -1;
    }
    else {
        ans[i] = st.peek();
    }
    st.push(A[i]);
}
```

TC: O(n)

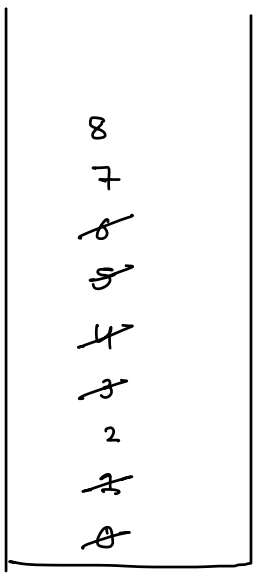
SC: O(n)

Nearest / next smaller on left (Index based)

for every element find its nearest smaller on left's index.

	0	1	2	3	4	5	6	7	8
A =	[10	16	5	9	12	8	25	7	13]
ans	-1	0	-1	2	3	2	5	2	7

```
for ( i → 1 to A.length-1 ) {
    while (st.size() > 0 && A[st.peek()] >= A[i]) {
        st.pop();
    }
    if( st.size() == 0) {
        ans[i] = -1;
    }
    else {
        ans[i] = st.peek();
    }
    st.push(i);
}
```



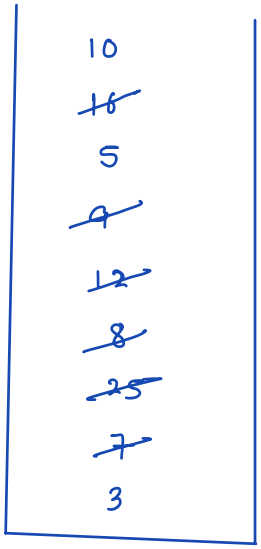
Q.3 Nearest / next smaller on right.

for every element find the value of nearest / next smaller on right

exact same logic but travel from right to left

A = [10 16 5 9 12 8 25 7 3]

ans 5 5 3 8 8 7 7 3 -1

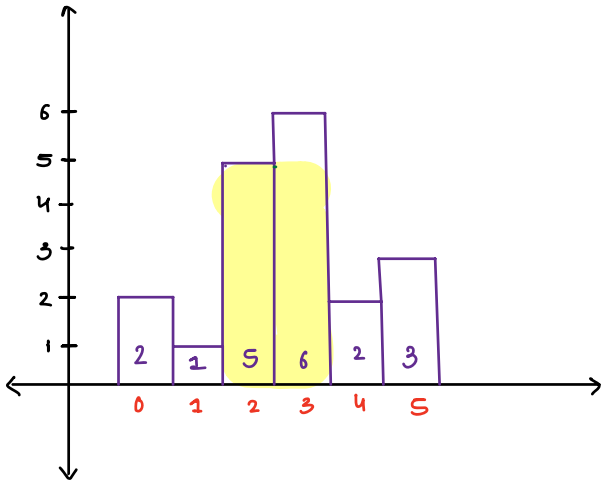


Nearest / next smaller on right (Index based)

Q.4 Largest area histogram

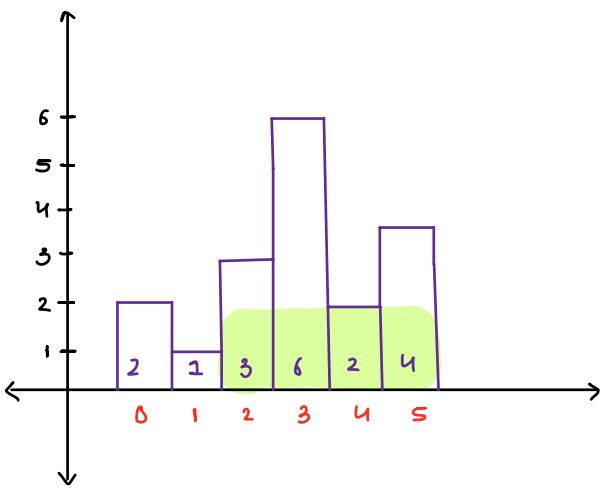
A = [2 1 5 6 2 3]

Return the area of largest rectangle possible?

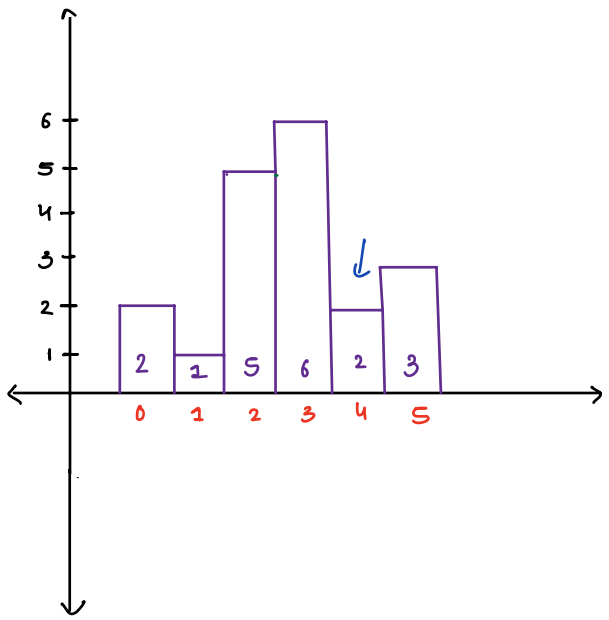


ans = 5 * 2 = 10

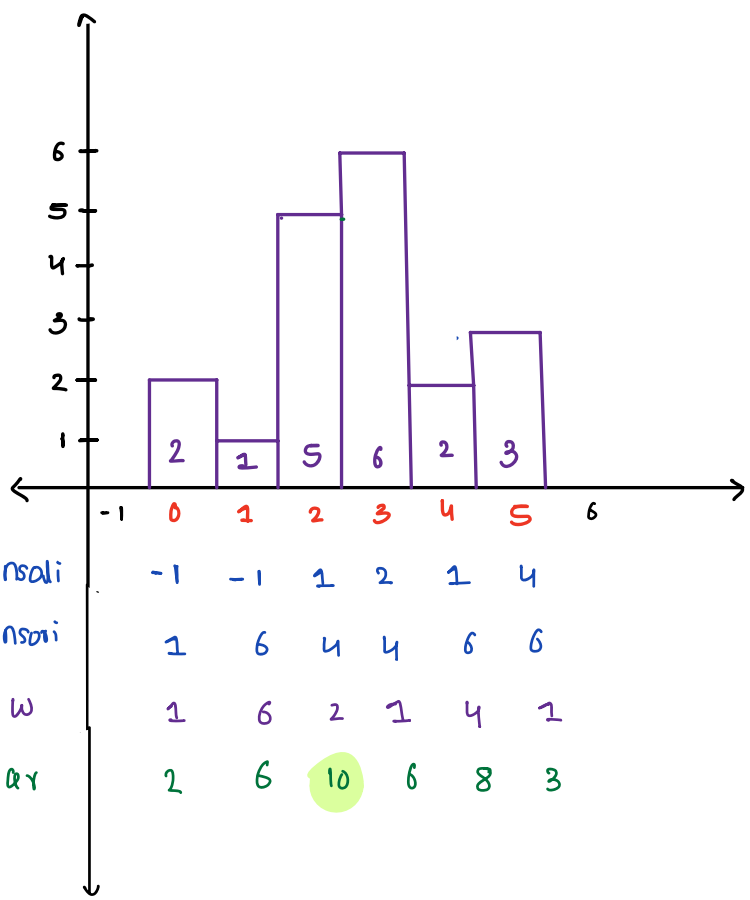
A = [2 1 3 6 2 4]



ans = 2 * 4 = 8



i	h	w	ans
0	2	1	2
1	1	6	6
2	5	2	10
3	6	1	6
4	2	4	8
5	3	1	3



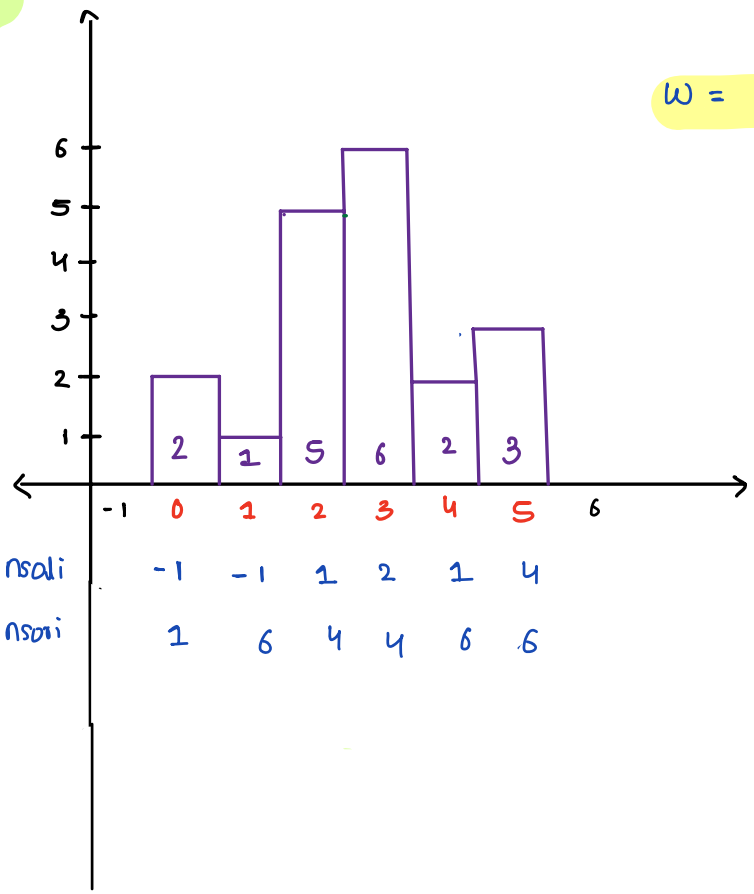

```
w = nsori[i] - nsoli[i] - 1
ar = w * A[i]
max = Math.max(ar, max)
```

TC : O(n)

SC : O(n)

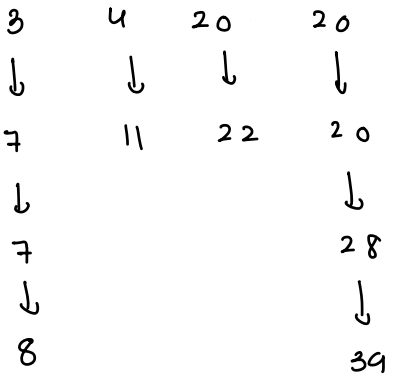
Doubts

$w = nsori[i] - nsdi[i] - 1$



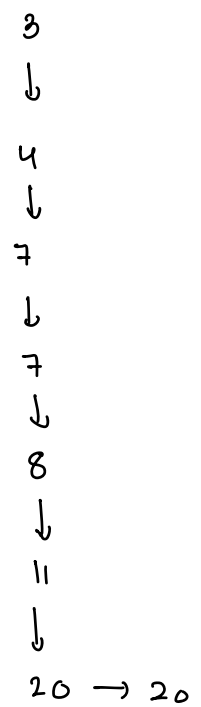
nsdi	-1	-1	1	2	1	4
nsori	1	6	4	4	6	6

temp
tn




```
ListNode {
    int val;
    ListNode next;
    ListNode down;
}
```

mh



20 → 20 → 20 → 22 → 28 → 39

```
ListNode mh = null;
ListNode temp = head;
while (temp != null) {
    ListNode tn = temp.right;
    temp.next = null;
    mh = mergedTwoSortedLL(mh, temp);
    temp = tn;
}
```