Todays Content:

- 1^{st} smaller on left → 1^{st} smaller on right
- 1^{st} larger on left → 1^{st} larger on right
- Area of Histogram
- Get Min()

1st smaller element on left side

Given ar[N], for every ele ar[i] find nearest smaller element on leftside.

Nearest ele : ar[i], distance between indices.

	0	1	2	3	4	5
Ex1: ar[6] =	4	5	2	10	3	12
ans[6] =	-1	4	-1	2	2	3

	0	1	2	3	4	5	6	7
Ex2: ar[8] =	4	6	10	11	7	8	3	5
ans[8] =	-1	4	6	10	6	7	-1	3

Ideal: for every ar[i]: iterate on left & get 1st smaller on leftside:

TC: O(N) * O(N) = O(N^2) SC: O(1)

```
int[] smallerleft(int ar[]) {   TC: O(N^2)  SC: O(1)
    int N = ar.length;
    int ar[N]; Initialize entire ar[] with -1;
    for(int i = 1; i < N; i++) {
        // for ar[i] get nearest smaller on left
        for(int j = i-1; j >= 0; j--) {
            if(ar[j] < ar[i]) {
                ans[i] = ar[j]; break;
            }
        }
    }
    return ans;
}
```

Eg1:

	0	1	2	3	4	5	6	7	8	9	10	11
ar[12] =	5	8	11	14	7	10	13	6	9	10	2	5
ans[] =	-1	5	8	11	5	7	10	5	6	9	-1	2

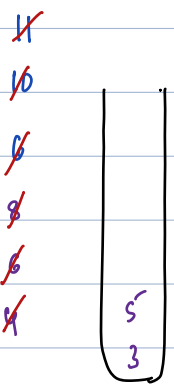
→ Operations:

1. push & pop are same time
2. peek & size() : Is Empty

Dabba : Stack

Eg2: ar[8] =

	0	1	2	3	4	5	6	7
	4	6	10	11	6	8	3	5
ans[] =	-1	4	6	10	4	6	-1	3



Idea:

```
while(st.sizeC() > 0 && st.peekC() >= ar[i]){
    st.popC()
}
if(st.sizeC() > 0){
    ans[i] = st.peekC();
}
else { ans[i] = -1; }
st.push(ar[i])
```

```
int[] 1st smaller(int ar[N]) {   TC: O(N)  SC: O(N) // stack
    int ans[N];                  Total N insertion + N deletions = 2N
    stack<Integer> st;
    for(int i=0; i<N; i++){
        // Get 1st smaller ele for ar[i] on left
        while(st.sizeC() > 0 && st.peekC() >= ar[i]){
            st.popC()
        }
        if(st.sizeC() > 0) {
            ans[i] = st.peekC();
        else { ans[i] = -1; }
        st.push(ar[i])
    return ans;
}
```

Nearest smaller Index on left:

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

stack: 0 1 2 . . .
(4 6)

```
int[] 1st smaller Index left(int ar[]){
    int ans[N];
    stack<Integer> st; // In stack we push index
    for(int i=0; i<N; i++){
        // Get 1st smaller ele for ar[i] on left
        while(st.sizeC() > 0 && ar[st.peekC()] >= ar[i]){
            st.popC()                    ↳ index
        }
        if(st.sizeC() > 0) {
            ans[i] = st.peekC(); // we storing index
        else { ans[i] = -1; }
        st.push(i) // only index
    return ans;
}
```

TODO: 1st smaller index on right
Iterate from i = {N-1, 0}

2) Nearest Greater on left side

	0	1	2	3	4	5	6	7
Inp: ar[8] =	11	8	6	2	10	9	4	1
ans[] =	-1	11	8	6	11	10	7	4


```
int[] 1st greater(int ar[N]) {  TC = O(N) SC: O(1)
    int ans[N];
    stack<Integer> st;
    for(int i=0; i<N; i++){
        // Get 1st smaller ele for ar[i] on left
        while(st.size()>0 && st.peek() <= ar[i]){
            st.pop()
        }
        if(st.size()>0){
            ans[i] = st.peek();
        }
        else { ans[i] = -1; }
        st.push(ar[i]) // st.push(i)
    }
    return ans;
}
```

ar[st.peek()] → 1st greater index on left

Note: Nearest greater index right:

Iterate from i = {N-1, 0}

Histogram area:

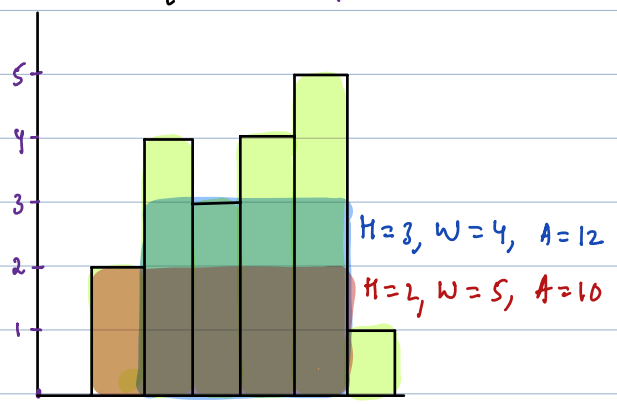
Given continous block of histogram find max Rectangular Area.

Rectangular area, should be with in histograms:

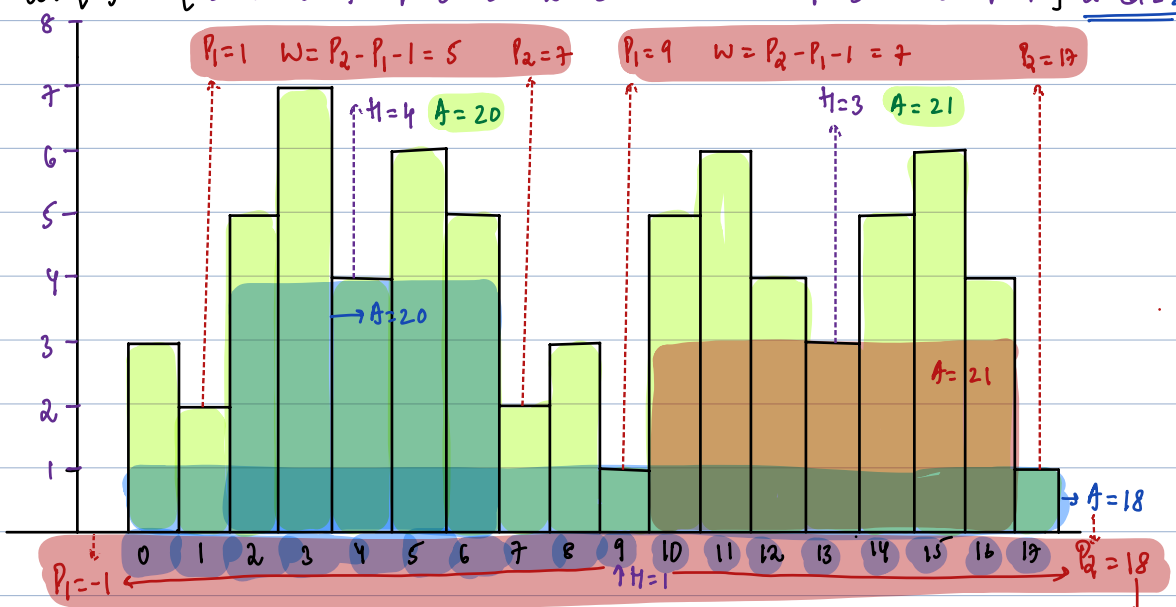
Note: Width of each histogram is 1 Note: Every Square is Rectangle

Every Rectangle is Not Square

Ex1: ar[6] = { 2 4 3 4 5 1 } area = 12



Ex2: ar[ ] = { 3 2 5 7 4 6 5 2 3 1 5 6 4 3 5 6 4 1 } area = 21



W = 18-(-1)-1

W = 18

Obs1: Height of any rectangle it should be = one of histogram heights

Obs2: For every histogram as ht of rectangle

Calculate P_1 on left = 1st smaller index on left = By default initialize $P_1 = -1$

Calculate P_2 on right = 1st smaller index on right = By default initialize $P_2 = N$

Width = $P_2 - P_1 - 1$ Area = width × ht & get overall max.

```
int Rectangular(int ar[N])  TC:O(N) SC:O(N)
    int left[N];
    stack<Integer> st; // In stack we push index
    for(int i=0; i<N; i++){
        // Get it's smaller ele for ar[i] on left
        while(st.size()>0 && ar[st.peek()]>=ar[i]){
            st.pop()                    ↳ index
        }
        if(st.size()>0){
            left[i] = st.peek(); // we storing index
        }else{left[i] = -1;}
        st.push(i) //only index
    }

    int right[N];
    stack<Integer> st; // In stack we push index
    for(int i=N-1; i>=0; i--){
        // Get it's smaller ele for ar[i] on left
        while(st.size()>0 && ar[st.peek()]>=ar[i]){
            st.pop()                    ↳ index
        }
        if(st.size()>0){
            right[i] = st.peek(); // we storing index
        }else{right[i] = N}
        st.push(i) //only index
    }

    int ans=0;
    for(int i=0; i<N; i++){ // for each histogram ar[i]
        int p1 = left[i], p2 = right[i];
    } ans = max(ans, ar[i]*(p2-p1-1))
    return ans;
}
```

	0	1	2	3	4	5
ar[6] =	2	4	3	4	5	1
left[] =	-1	0	0	2	3	-1
right[] =	5	2	5	5	5	6
Width =	5	1	4	2	1	6
Area =	10	4	12	8	5	6

Stack:

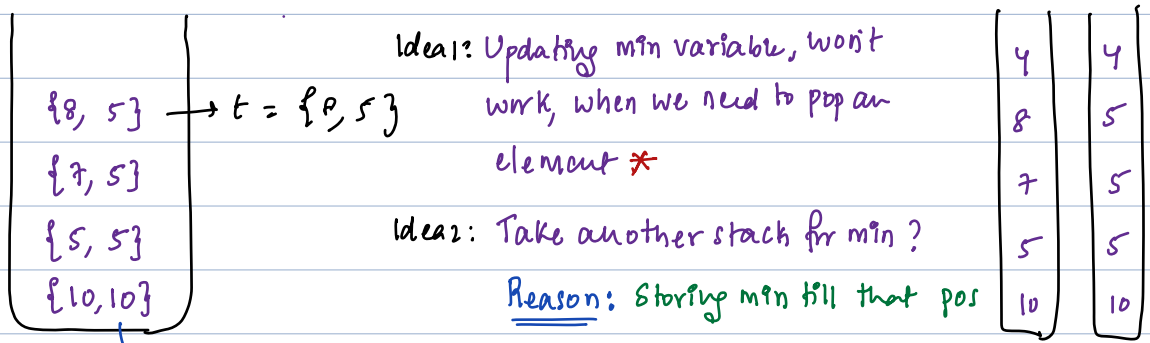
Create a stack with 5 following operations

a. push() b. peek() c. getMin() : return min of entire stack.

d. pop() e. size() Expected TC: O(1)

Eg: 10 5 7 8 4 getMin() pop() getMin() peek() 2

4 {4, 4} 5 8



Idea1: Updating min variable, won't work, when we need to pop an element *

Idea2: Take another stack for min ?

Reason: Storing min till that pos

```
stack<Pair> st = new stack<>();
```

```
class Pair{
    int ele;
    int m;
    Pair(int a, int b){ // Constructor
    } ele=a, m=b
}
```

```
stack<Pair> st = new stack<Pair>(); // Each ele is of Pair Object
```

```
void own_push(int x){
    if(st.size()==0){
        Pair p = new Pair(x, x)
    } st.push(p)
    else{
        Pair t = st.Peek();
        int val = min(x, t.min)
        Pair p = new Pair(x, val)
    } st.push(p)
}
```

```
void own_pop(){
    if(st.size()==0)
    { return }
    st.pop();
}
```

```
int own_getMin(){
    if(st.size()==0){
    } return
    Pair t = st.Peek();
} return t.min;
```

```
int size(){
    return st.size();
}
```

```
int own_peek(){
    if(st.size()==0){
    } return
    Pair t = st.Peek();
} return t.ele;
```