Todays Content :

→ 1st smaller on left → 1st smaller on right

→ 1st larger on left → 1st larger on right

→ Area of Histogram

→ Sum of Max of every subarray

1st Smaller element on leftside

Given ar[N], for every index ar[i] find 1st smaller element on leftside.
nearest smaller element

Ex1:

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

Idea1: For every ar[i] iterate on left & get 1st small element

```
int[] smallerleft(int ar[n]){        TC: O(N^2)  SC: O(1)
    int ans[n] = -1
    i = 0; i < n; i++){
        // for ar[i] get 1st smaller
        j = i-1; j >= 0; j--){
            if(ar[j] < ar[i]){
                ans[i] = ar[j]
                break
            }
        }
    }
    return ans[]
}
```

↳ nlogn : sorting / BS
*   *
↳ n ?

Ex1:

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

container

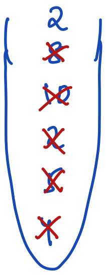
~~5~~ ~~2~~ ~~8~~ ~~10~~ ~~6~~ 1

→ insert
→ delete
→ access

} all operations same side
Container stack

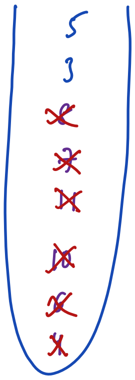
Ex2:

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



Ex3:

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



```
int[]   1st smaller( int ar[N] ) {   TC: O(N)  SC: O(N)
    int ans[N] = -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[]
}
```

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

~~0~~	~~1~~	~~2~~	~~3~~	~~4~~	~~5~~	6	7	

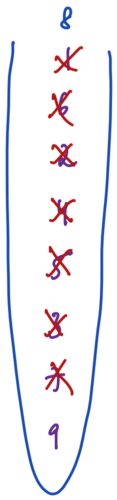
```
int[]   1st smaller Index ( int ar[N] ) {   TC: O(N)  SC: O(N)
    int ans[N] = -1
    stack<int> st
    i= 0; i< n; i++){
        while( st.size()>0 && ar[st.top()] >= ar[i] ){
            st.pop()
        }
        if ( st.size()>0){
            ans[i] = st.top()
        }
        st.push (i)
    }
    return  ans[]
}
```

Note: nearest smaller in right, iterate from n-1 → 0

2) Nearest Greater on left side

	0	1	2	3	4
Ex1: ar[5] =	3	6	5	8	2
ans[5] =	-1	-1	6	-1	8

	0	1	2	3	4	5	6	7	8
Ex2: ar[9] =	9	7	3	5	4	2	6	1	8
ans[9] =	-1	9	7	7	5	4	7	6	9



```
int[]  1st greater left( int ar[N] ) {
    int ans[N] = -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])      ──→  st.push(i)
    }
    return  ans[]
}
```

st.top() <= ar[i] → ar[st.top] <= ar[i]

if we need index for 1st greater left

st.push(ar[i]) → st.push(i)

Note: nevarest greater in right, iterate from n-1 → 0

8:05am $\xrightarrow{10mins}$ 8:15am

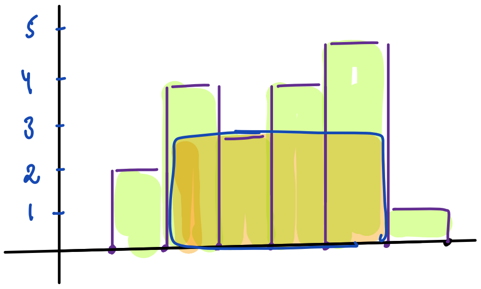
Histogram area:

Given Continous block of Histogram find max Rectangular area which can be present with in histograms

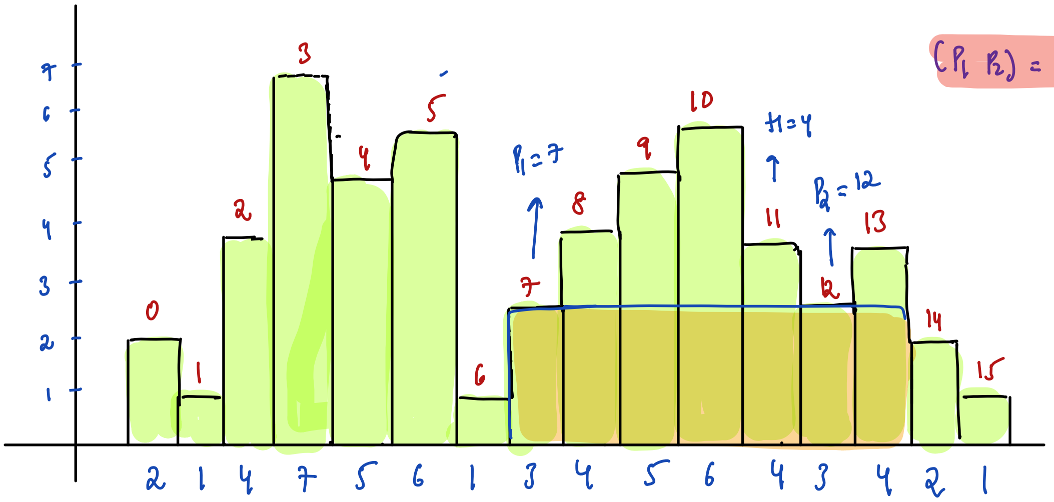
Note: Every histogram is of width = 1

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

$$\begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \\ 2 & 4 & 3 & 4 & 5 & 1 \end{array}$$



Ex2: ar[] = 2 1 4 7 5 6 1 3 4 5 6 4 3 4 2 1



$(P_1\ P_2) = P_2 - P_1 - 1$

Obs1: Our Rectangle height will match with a histogram height

↳ Obs2: Say we fix our rectangle height, repeat for every histogram

a) Keep extending to left until we find index with height < rectangle height = P_1

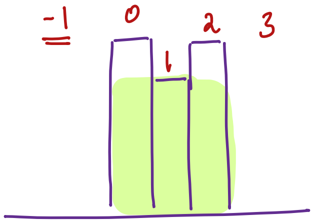
b) Keep extending to right until we find index with height < rectangle height = P_2

c) Rectangular area = $[P_2 - P_1 - 1] * \{ height \}$

```
int  Rectarea(int hist[n]){   TC: O(N+N+N) = O(N)   SC: O(N+N) = O(N)
  //Obs: For every histogram height we need to get 1st smaller
         inden on left & on right
   int left[ ] =  smaller index left (hist)
   int right[ ] =  smaller index right (hist)
   ans = 0;              Note: 1st smaller inden on right default
                               value should be  N
   i = 0; i < N; i++){
      // Considering ith histogram with height = hist[i]
      // 1st smaller inden on left, 1st smaller on right
          l = left[i] , r = right[i]
      ans = man( [r-l-1]* hist[i], ans)
   }
   return ans;
}
```

Edge Case:

Ex: ar[3] = 4 3 4   ans = 9



left[ ] =	-1	-1	1
right[ ] =	1	3	3
r-l-1 =	1	3	1

3Q) Given ar[N] distinct elements sum of max of every subarray

Ex: ar[3] = { 4 2 3 } ans=20

	max
{4} →	4
{4 2} →	4
{4 2 3} →	4
{2} →	2
{2 3} →	3
{3} →	3

Contribution technique :

→ For every ar[i] we need to calculate in how many subarrays it's max = Ci

```
ans = 0;
i = 0; i < N; i++) {
    // Ci = number of subarrays in
       in which ar[i] is max
    ans = ans + ar[i] * Ci
}
```

Ex1: ar[] = { 1 3 10 1 4 2 8 6 4 14 2 19 }

0	1	2	3	4	5	6	7	8	9	10	11
1	3	10	1	4	2	8	6	4	14	2	19
		x ↑ P_1	✓	✓	✓	✓✓ ↑ i	✓	✓	x ↑ P_2		

$s = i - P_1$ i $e = P_2 - i$

obs: P_1 = 1st greater index on leftside

P_2 = 1st greater index on rightside

start	end	subarrays
3	6	4 x 3 = 12
4	7	
5	8	
6		

```
int maxSub(int ar[N]){ TC: O(N)  SC: O(N)
    int left[ ] =    greater index left (hist)
    int right[] =    greater index right (hist)
                     Note: 1st greater index on right default
                           value should be N
    ans = 0;
    i = 0; i < n; i++){
        // 1st greater index on left, 1st greater on right
            p1 = left[i]  p2 = right[i]
            c  = (i - p1) * (p2 - i) // no: of subarrays in with ar[i] is max
            ans = ans + (c)* ar[i]
    }
    return ans;
}
```

→ sum of min of every subarray : TODO

→ If elements are not distinct:

- → sum of max of every subarray : TODO
- → sum of min of every subarray : TODO

→

$$\underline{Ex1}: ar[\,] = \begin{array}{ccccc} 0 & 1 & 2 & 3 & 4 \\ \{4 & 3 & 10 & 2 & 5\} \end{array}$$

$$\hookrightarrow = \{10 \;\; 5 \;\; 4 \;\; 3 \;\; 2\}$$