Today's Content :

→ Nearest Smaller in left side

→ Area of Histogram

→ Max Rectangular area

→ Max - Min In Every Subarray

Q1) Nearest Smaller Element on left.

Given an array of +ve Integers

For every index i, find the nearest element on left side of i, which is smaller than A[i]

ar[] =	4	5	2	10	8	2
	-1	4	-1	2	2	-1

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

Idea: // ans[N] = {-1}; → We are assigning all value as -1

```
i = 0; i < N; i++) {
  // j = i-1; j >= 0; j--) {
       if( ar[j] < ar[i]) { ans[i] = ar[j];
                            break }
  }
}
```

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

arr[ ] = 5 2 8 10 12 6 1

-1 -1 2 8 10 2 -1

ans space

~~5~~, ~~2~~, ~~8~~, ~~10~~, ~~12~~, ~~6~~, 1

arr[ ] = 4 6 10 11 7 8 3 5

-1 4 6 10 6 7 -1 3

ans space

~~4~~, ~~6~~, ~~10~~, ~~11~~, ~~7~~, ~~8~~, 3, 5

Stack :

- We are adding later
- We are deleting later

// arr[i] > st.top()

ans[i] = st.top()

arr[i] <= st.top()

st.pop()

Pseudocode

```
stack<int> st;
int ans[N] = {-1} // (Initialization)
for(int i = 0; i < N; i++){
    while(st.size() > 0 &&
              arr[i] <= st.top()){
        st.pop();
    }
    if (st.size > 0){
        ans[i] = st.top()
    }
    else { ans[i] = -1 }
    st.push(arr[i])
}
```

TC: Every element will go inside stack & come out once.

$\hookrightarrow$ TC: O(N)   SC: O(N)

3 Find the index of nearest smaller on left side

// push index in stack.

```
stack<int> st;
int ans[N] = {-1} //(Initialization
for(int i=0; i<N; i++){
    while(st.size()>0 && ar[i] <= ar[st.top()])
        st.pop();
    }
    if(st.size()>0){
        ans[i] = st.top()
    }
    else { ans[i] = -1 } } // Even if we neglect else, That Okay
    st.push(i)
}
```

st.top() → you are only storing indices

Q3) Get distance of nearest smaller element on left side

Q4) Find the nearest smaller on the right side?

i = N-1; i>=0; i--){

Q5) Find the nearest greater on left side ?

→ conditions will change ✓

Q6) Find the nearest greater on right side

→ i = N-1 ; i >= 0 ; i--) {

conditions will change

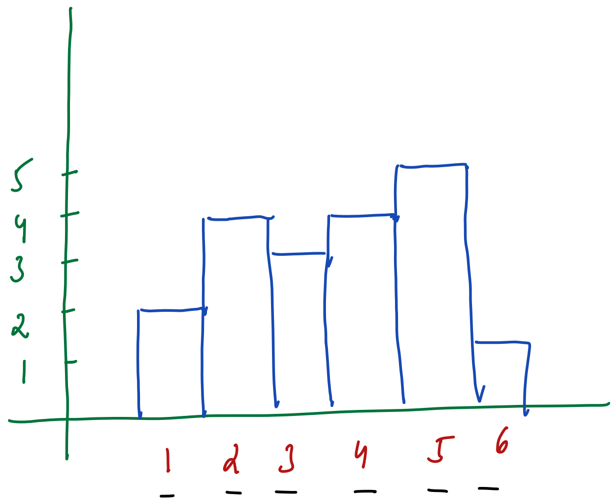
}

Q) Given Continous Blocks of histogram find max Rectangular area ?

→ Find max Rectangular Block, which cannot exceed histogram

width of all hist = 1

Ex:

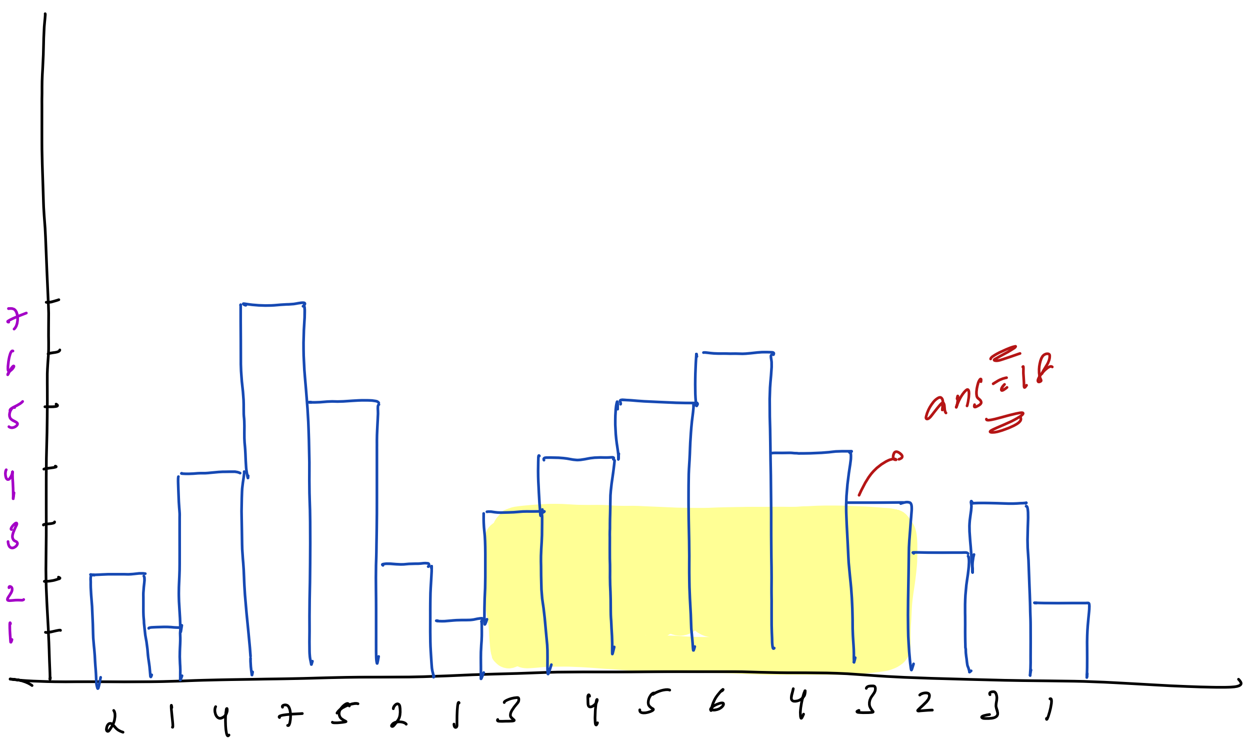


for all pairs

s	e	H	w	area
1	1	2	1	2
1	2	2	2	4
1	3	2	3	6
1	4	2	4	8
1	5	2	5	10
1	6	1	6	6

Ex:

arr[] : 2 1 4 7 5 2 1 3 4 5 6 4 3 2 3 1



Idea:

Get all pairs for (s, e)

// Say N Buildings

s = 0 ; s < N ; s++) {

e = s ; e < N ; e++) {

[s e] (iterate & get)

get minval = H

W = e - s + 1

ans = max (ans, H * W)

}

}

// Carry forward minval & maxval

x = min [i j]

min [i, j+1] = min (x, ar[j+1])

min [i j]

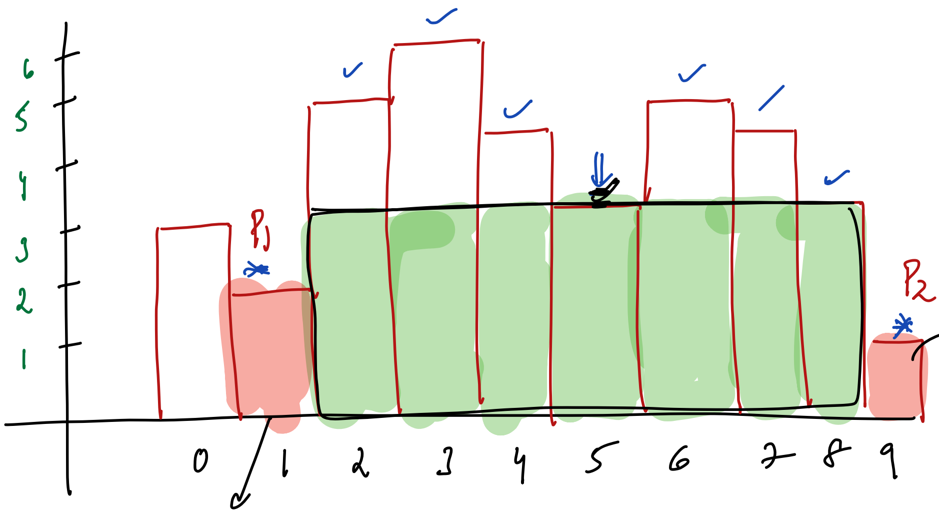
TC: $O(N^2 * N) \Rightarrow O(N^3)$

SC: $O(1)$

TC: $O(N^2) \Rightarrow$ By carry forward

SC: $O(1)$

Ex:



find small < 3 in right

ind = 9

find smale < 3 in left

ind = 1

→ len = 9 - 1 - 1 = 7 * 3 = 21

// Idea: Take every building:

{

P_2 : get first smaller in right — find smaller index on right

P_1 : get first small in left — find smaller index on left

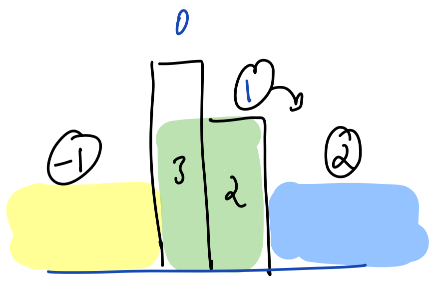
width = $(P_2 - P_1 - 1)$

H = arr[i]

ans = max(ans, H*W)

}

Ex:



find small index left : $P_1[\,]$: -1   -1 → $P_2 - P_1 - 1 = (-1)$

find small index right : $P_2[\,]$: 2 $\cancel{1}$ 2 → $2-(-1)-1 = 2$

$P_2 - P_1 - 1$: 1 $(2-(-1)-1) \Rightarrow 2$

// find small in left $P_1[N] = \{-1\}$ ⇒ // Calculate ⇒ $O(N)$

// find small in right $P_2[N] = \{N\}$ ⇒ // Calculate ⇒ $O(N)$

```
for(i=0; i<N; i++){
    ans = max(ans, (P2[i]-P1[i]-1)*(ar[i]))
}
```

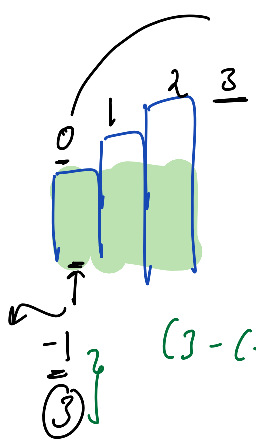
→ $O(N)$

width

TC: $O(N+N+N) \Rightarrow O(N)$

SC: $O(N)$

11:pm break



$(3-(-1)-1) = 4-1 = 3$

30) Given a matrix, which contains 1 & 0, find max rectangular area, which contains all 1's

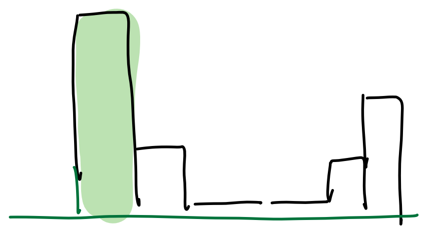
	0	1	2	3	4	5
0	1	1	0	1	0	1
1	0	1	1	1	1	1
2	1	1	1	1	1	0
3	1	0	1	0	0	1
4	1	1	0	0	1	1

Bars

1	1	0	1	0	1	→ 2
0	2	1	2	1	2	→ 5
1	3	2	3	2	0	→ 8
2	0	3	0	0	1	→ 3
3	1	0	0	1	2	→ 3

ans = 8

	0	1	2	3	4	5
0	1	1	0	1	0	1
1	0	2	1	2	1	2
2	1	3	2	3	2	0
3	2	0	3	0	0	1
4	3	1	0	0	1	2



Idea:

for every column take pfsum but if a val = 0, reset sum = 0

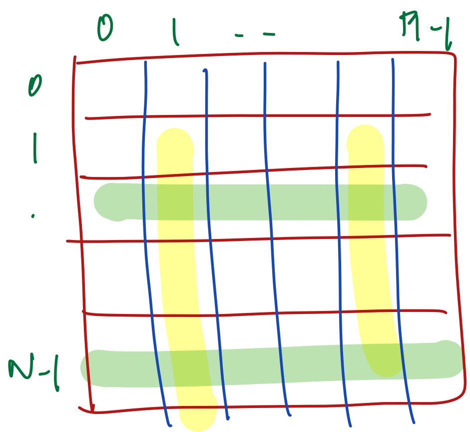
$N*M$

for any row Apply max rectangle area → $N*M$

TC: $O(N*M + N*M) \approx O(N*M)$ SC: $O(M)$

idea: for every submatrix get sum == sign of rectangle

Construct pf[][] matrix & get get every submatrix sum. ⇒ How many ways to fix start & end row ⇒ $\frac{(N)(N+1)}{2}$



How many ways to fix start & end col ⇒ $\frac{(M)(M+1)}{2}$

⇒ submatrix ⇒ $\frac{(N)(N+1)}{2} \frac{(M)(M+1)}{2}$

⇒ $O(N^2M^2)$

4Q) Given N Array elements, find sum of max of every subarray

Contribution Technique

TC: $O(N^3)$

for every subarray get max & get

TC: $O(N^2)$

Carry forward subarray max

TC:

$$\sum_{i=0}^{N-1} (\text{for every element } ar[i] \text{ in how many subarrays it is max}) * ar[i]$$

Ex: [1 4 3]

[1] : 1
[4] : 4
[3] : 3
[1 4] : 4
[1 4 3] : 4
[4 3] : 4

4*4 + 1*1 + 3*1

Total = 16 + 1 + 3

=> 20

0 1 2 3 4 5 6 7 8 9

i

Ex: 9 1 8 3 2 4 5 3 2 6 10

4

3

first greater in left (P_1)

$(i - P_1)$

$(P_2 - i)$

first greater in right (P_2)

4 * 3 = 12

How do initialize

$P_2 = N$

//

How do we initialize $P_1 = -1$

Note:

Take care of repetitions, slightly tricky edge case

//

// Sum of (Max - Min) in Every Subarray

first greater in left

first greater in right

first smaller in left

first small in right

Done

→ infix → postfix : (Incoming)

→ postfix - Evaluation { = }

max freq stack :

Ex: 3 4 ~~8~~ 5 4 3 7 8, 7, 4 7

1 : | 3 | 4 | ~~5~~ | 7 | 8

2 : | 4 | 3 | 7

3 : | ~~4~~ | ~~7~~ |

map

3 : ~~1~~ 2

4 : ~~1~~ ~~2~~ ~~3~~ 2

5 : ~~1~~ 0

7 : ~~1~~ ~~2~~ ~~3~~ 2

8 : 1

maxfreq = ~~3~~ 2