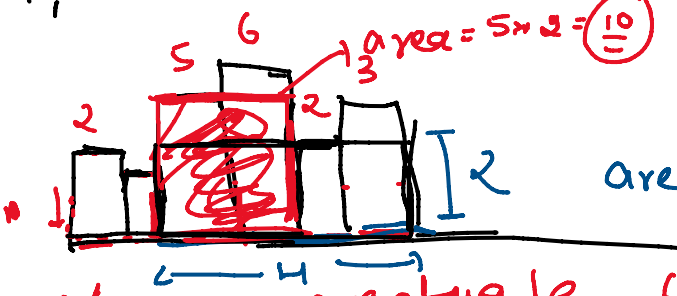
Class - 28

optimal approach — Stack

2, 1, 5, 6, 2, 3 — Largest Rectangle in Histogram

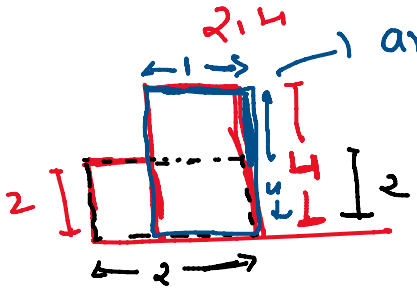


area = 5 * 2 = 10

Max area Rectangle = 10

area = 8

Max area rectangle = 6 * 1 = 6



area = 2 * 2 = 4

4 − 0 − 1

Max Area Rectangle = 4

Approach

Hint: Height ⇒ expand → left and right

for each bar find left smaller and right smaller

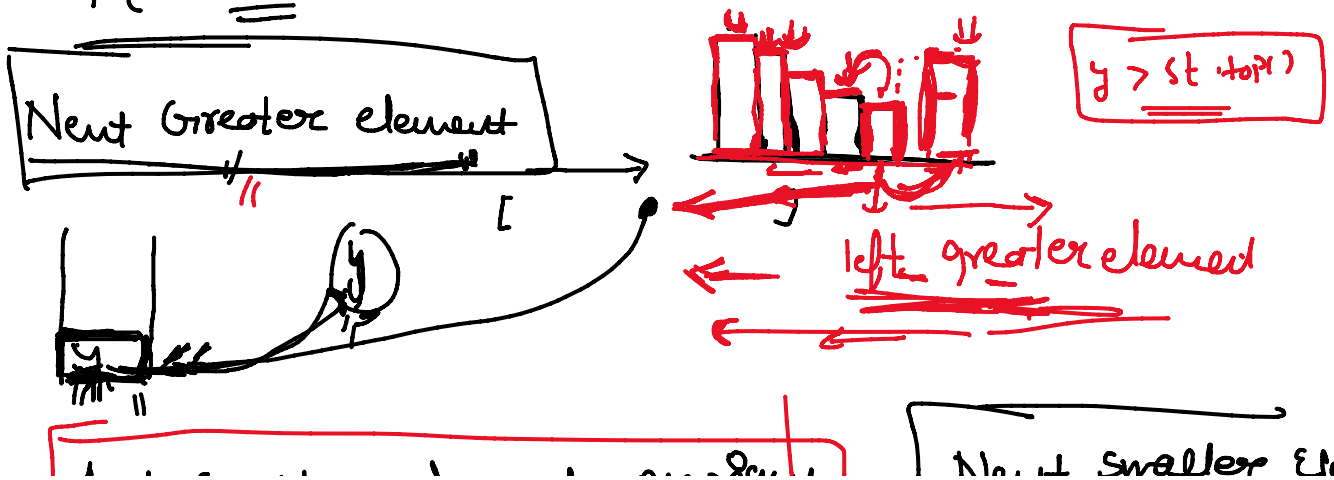
Brute Force

area of Rectangle

```
for (int i = 0; i < n; i++) // Traversal on all the bars
```

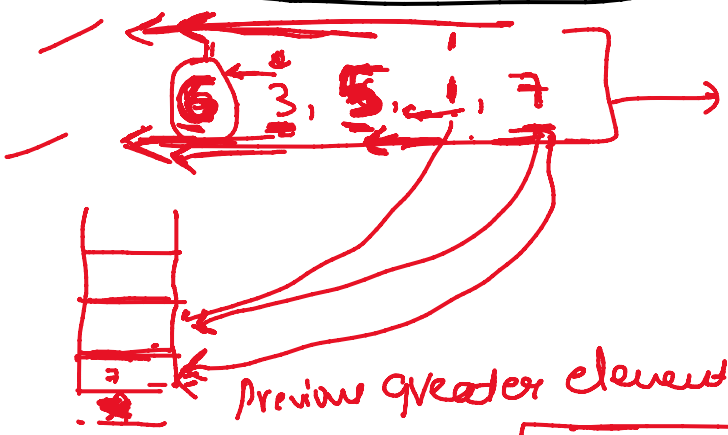
```
for ( i=0 ; i<n ; i++ )  // Traversal on all the bars
{
    // find left minimum; int leftSmaller = -1;
    for ( j = i-1 ; j>=0 ; j-- )
    {
        if (nums[j] < nums[i])
        {
            leftSmaller = j ; break ; }
    }
    int rightSmaller = -1 ;

    For ( j = i+1 ; j<n ; j++ )
    {
        if (nums[j] < nums[i])
        {
            rightSmaller = j ; break ; }
    }
    // we have leftSmaller and rightSmaller
       so we can find area
}
```

T.C = $O(n^2)$



Next Smaller Element

Just smaller element on right



Previous greater element

if (3 > st.top()) ✗

st.top() > 3

Next Smaller Element on right

Next Greater element on right side

while (nums[i] > st.top())
st.pop()

nums[i] < st.top()

Max Area Rectangle in a Histogram

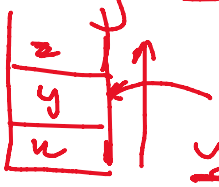
↳ For every element find → Next smaller and Previous Smaller

Next Greater and Previous Greater element

Next Smaller element

Next Greater Element using stack

S.push(0)

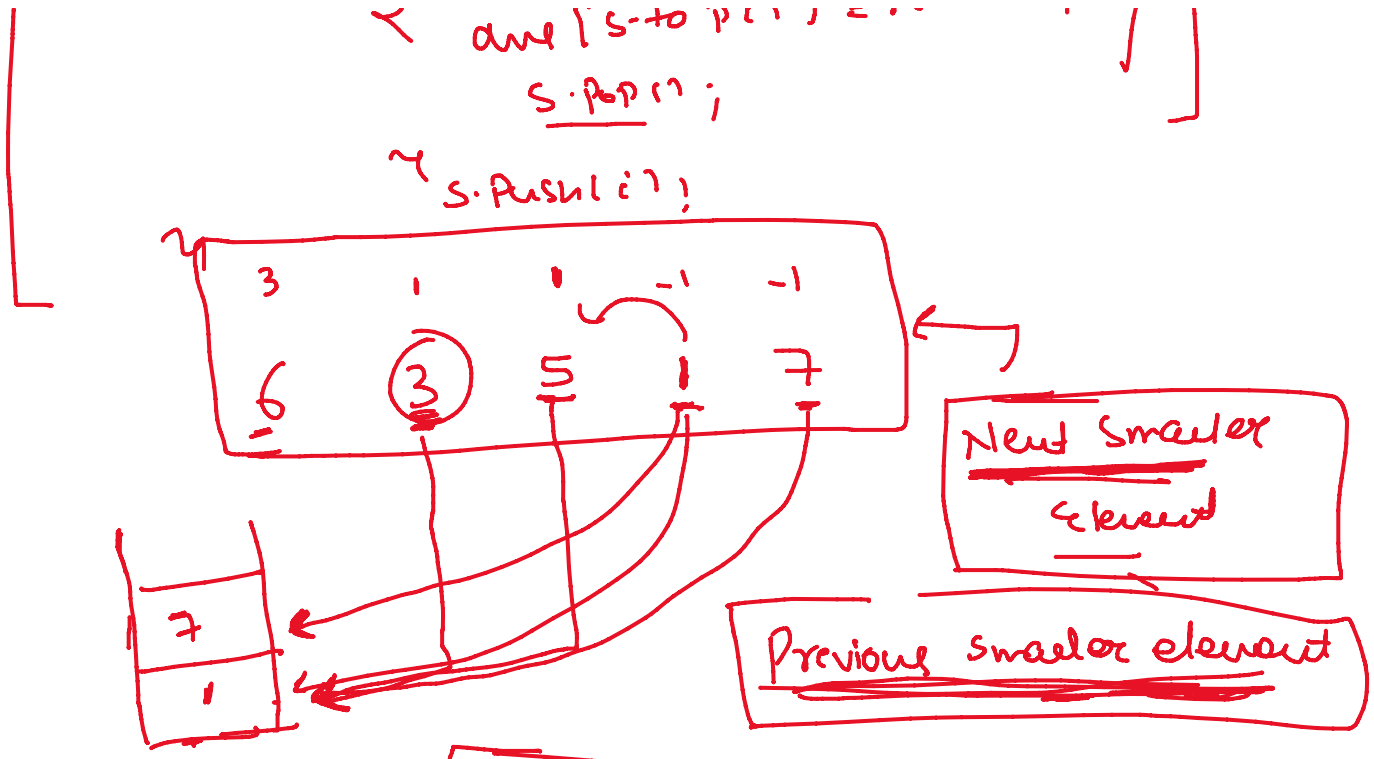


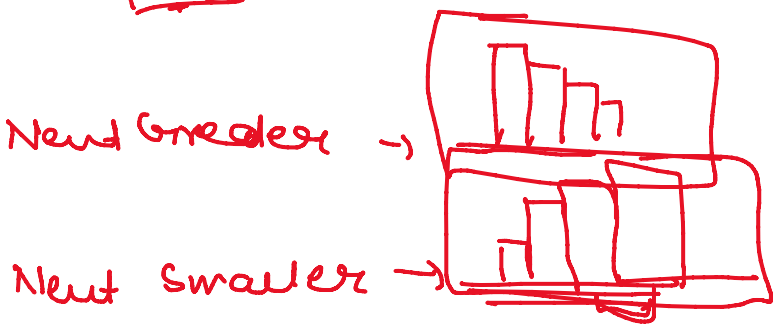
y > x

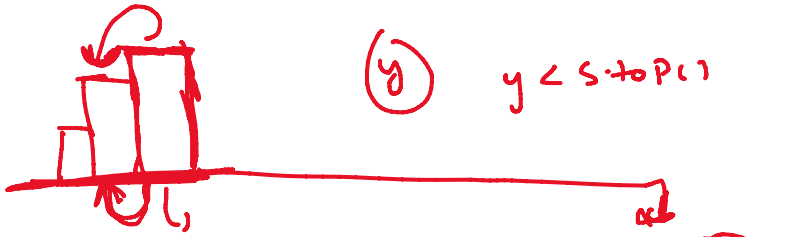
for (i=1 ; i<n ; i++)

{ while (!s.empty() and nums[i] < nums[s.top()])

{ arr[s.top()] = nums[i] ; s.pop() ; }







Next smaller element = y

Previous smaller element

area = (Bar Height) * (Next small (index) − Prev small (index))

Next smaller element → area

Stack <int> st;

s.push(0);

For (i=1 ; i<n ; i++)

{ while (!s.empty() and nums[i] < nums[s.top()])

{ int tp = s.top();

s.pop();

int area = nums[tp] * (s.empty() ? i : (i - s.top() - 1));

ans = max(ans, area);

if No Previous element

smaller

Next smaller idx

Prev smaller idx

6 3 5 1 →

s.push(i);

}

s.push(i);

}

Previous smaller Element

pop()

Rectangle Height

width: next smaller idx - prev small idx - 1

6 5 3 4 !

0 1 2 3 4

4

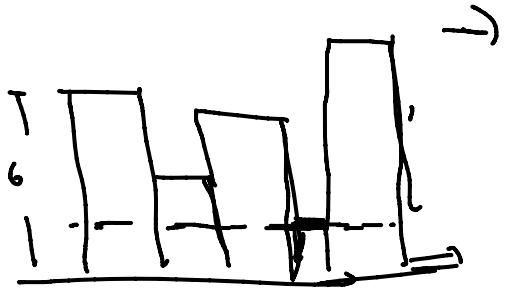
4 - (1) - 1 = 4

=> 4 * 1 = 4

6 3 5 1 7 =>

7

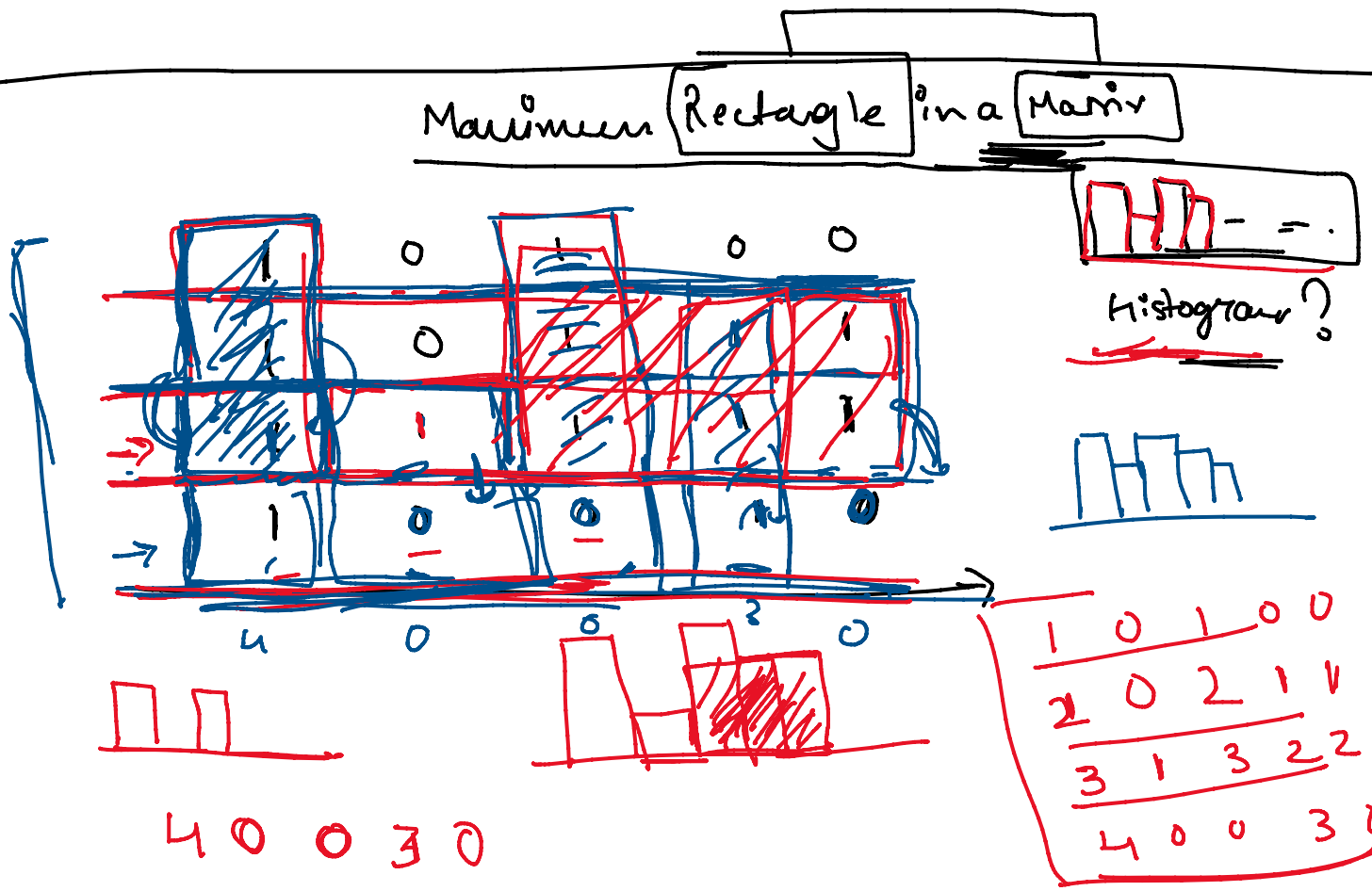
4



prev smaller = -1

```
while( !s.empty() )
{  int tp = s.top();
   int curarea = num[tp] * (s.empty() ? n : n - s.top() - 1)
   ar = max(ar, curarea);
}
```

Maximum Rectangle in a Matrix




```
Vector<int> hist(m,0);
```

```
For (i=0; i<n; i++)
{   For (j=0; j<m; j++)
    {   if (matrix[i][j] == 1)
        {   hist[j]++;
        }
        else
            hist[j]=0;
    }
Ind area = maxarea(hist);
```

im row → base row

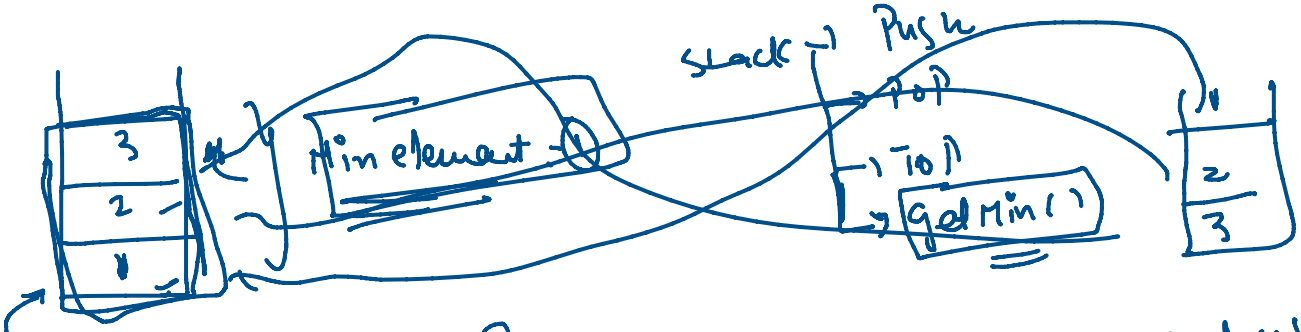
→

```
for
  for
    for
```

For each element i → $2n$

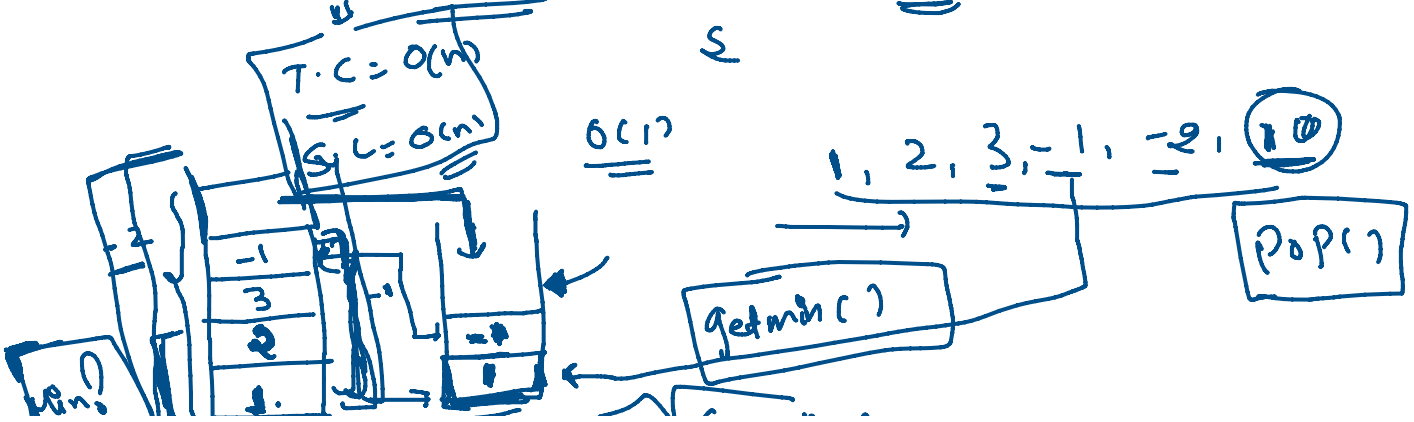
Total elements = n ⇒

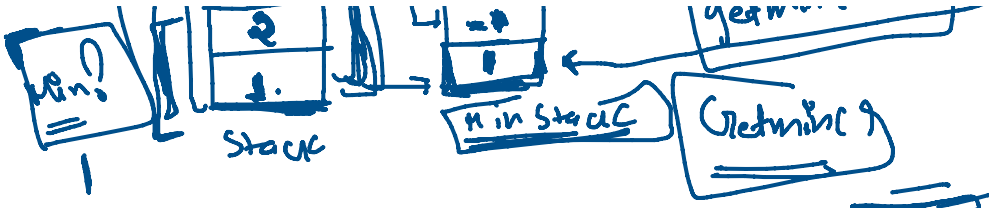
Total complexity = $2n^2 \sim O(n^2)$



Brute force?

```
while( !s.empty())
{
```

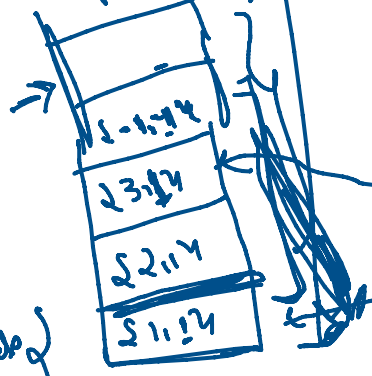




T.C = O(1)

i = 0 (n) → optimize ? O(1) ?

Min value of Stack



1, 2, 3, -1, -2, 10

Pair<int, vector<int>>

Pair<int, int>

Min value

2 → Stack, String (2-3)

String → PatternMatching

KMP, LPS, Rabin Karp