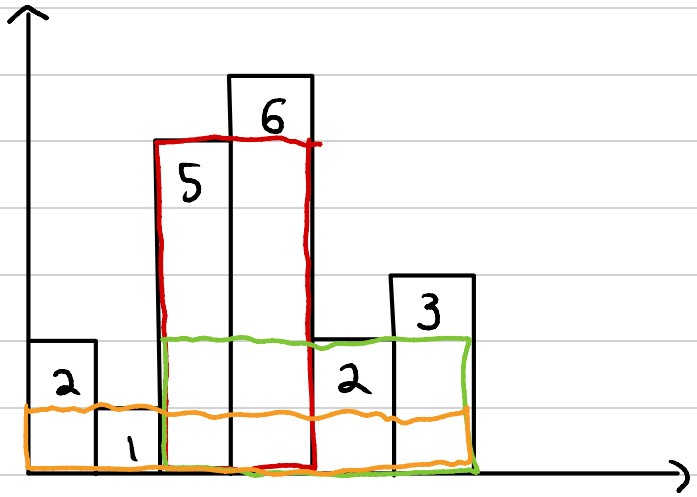
LARGEST RECTANGLE IN GIVEN HISTOGRAM

Given an array of integers, where each integer represents height of a histogram, our job is to return the area of largest rectangle formed.

Brute force solution is to iterate through the array and check the biggest rectangle each histo can form

For every block, the largest rectangle continues as long as next and previous element is encountered.

Eg:



∴ Concept of previous smaller and next smaller elemenent is used.

Iterate through the array while

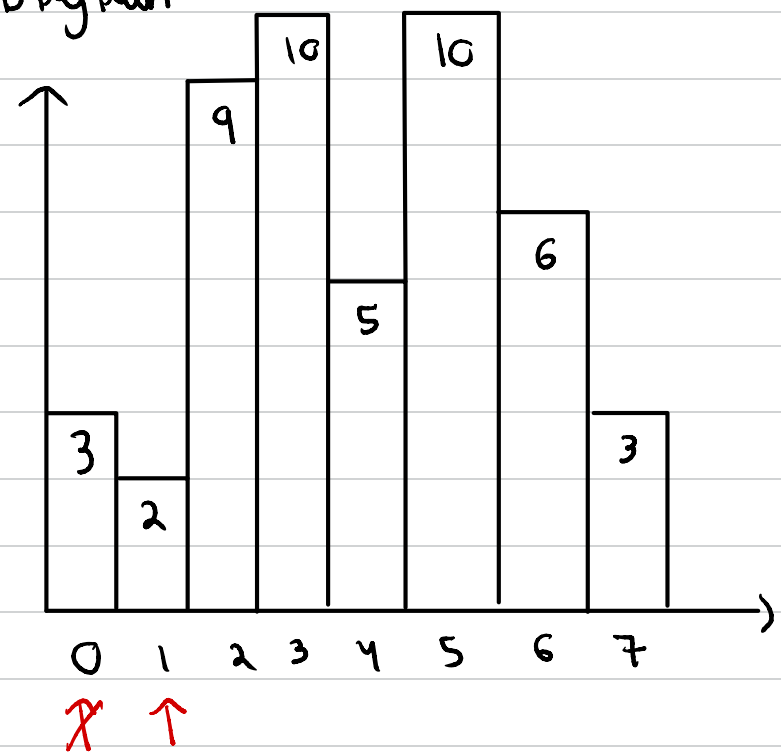
maintaining a max area value.

Time Complexity is $O(5N)$ and space complexity is $O(4N)$.

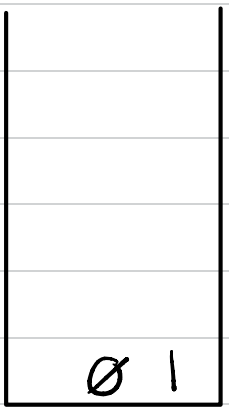
Optimal Solution involves computing previous smaller element on the way as we iterate.

When we are at i^{th} iteration, we are computing area for $(i-1)^{th}$ bar. Hence its pse and nse is known.

Dry run



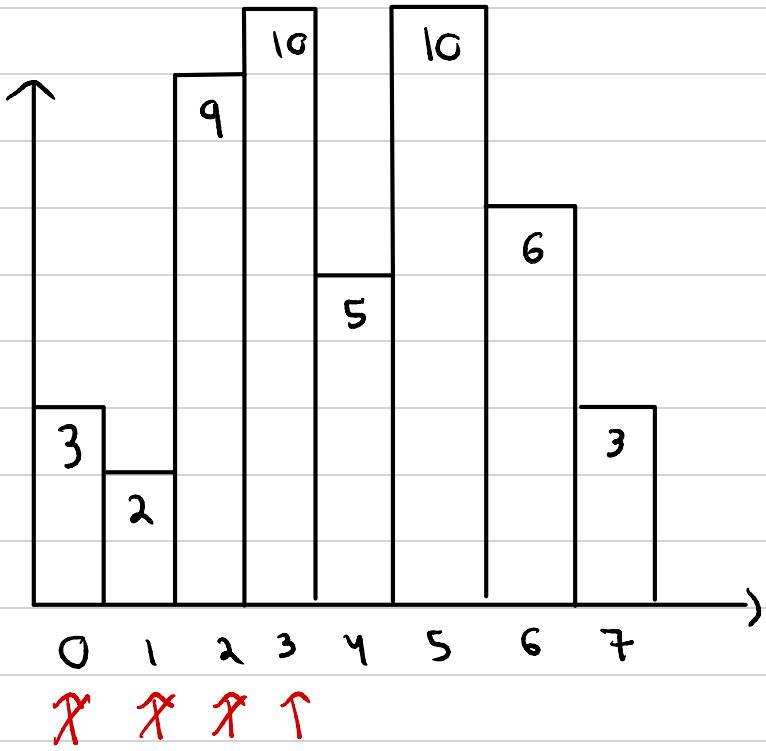
maxArea = 0



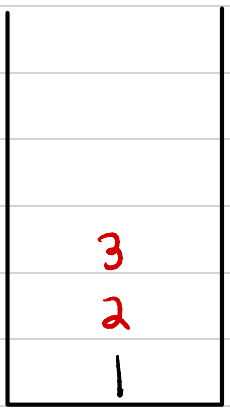
Stack
↓
indices

We start at $i = 0$, $arr[i] = 3$. Stack is empty so we just push it.
Now $i = 1$, $arr[i] = 2$. This is smaller than 3 (top of stack) so we pop it and push 2. Now, 2 is also actually nse for 3. Hence while sitting on

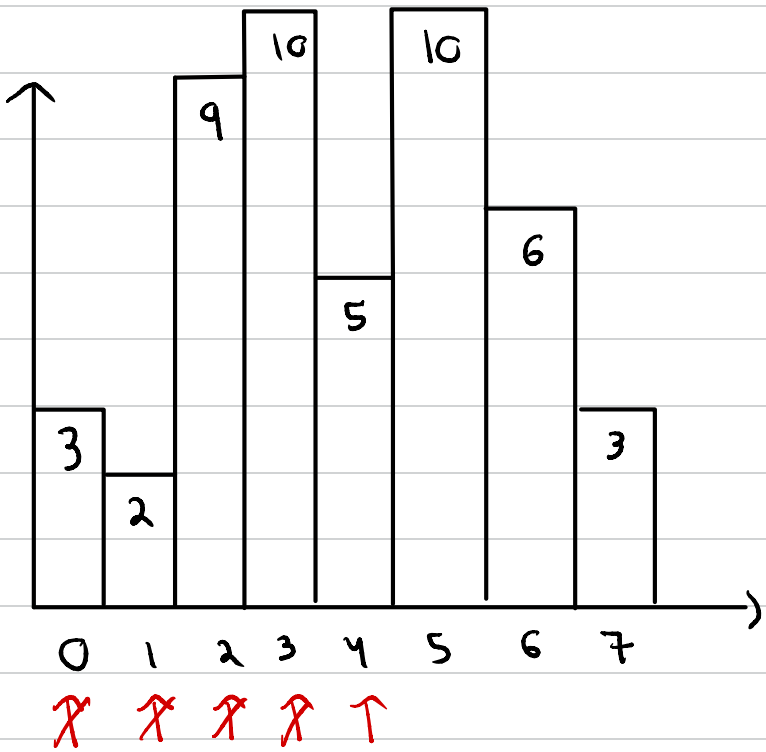
i = 1, we do area calculation for i = 0.



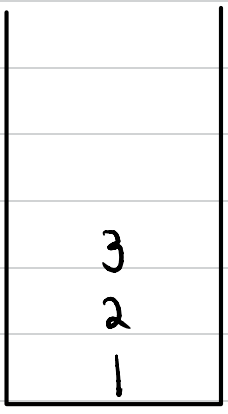
maxArea = ∅ 3



Area for i = 0 ⇒ arr[i] × (nse − pse − 1)
= 3 × (1 − (−1) − 1)
= 3



maxArea = ∅ ~~3~~ 10



For i = 4, we need to pop from stack. While popping, we realize that arr[i] = 5 is NSE for my

previous i. Hence we compute its area.

Area for arr[i] = 10 ⇒ 10 × (4 − 2 − 1)
= 10

Max area changes and we pop from stack Now i = 2 is still larger than 5 hence we need to pop But before we do that we have to find its area (and so on...)

Pseudocode

```
LargestRectangle(arr, N) {
  stack<int> st;
  maxArea = 0;
  for(i = 0 → N − 1) {
     while(!st.empty() && arr[st.top()]
                                > arr[i]){
        element = st.top();
        st.pop();
        nse = i;
        pse = st.empty() : -1 : st.top();
        area = arr[element] × (nse − pse − 1);
        maxArea = max(maxArea, max);
     }
     st.push(i);
  }
  while(!st.empty()) {
     nse = N;
     el = st.top();
     st.pop();
     pse = st.empty() ? -1 : st.top();
     area = arr[el] × (nse − pse − 1);
```

```
            maxArea = max(area, maxArea);
        }
    return maxArea;
}
```

Time Complexity is O(2N) and Space Complexity is O(N).