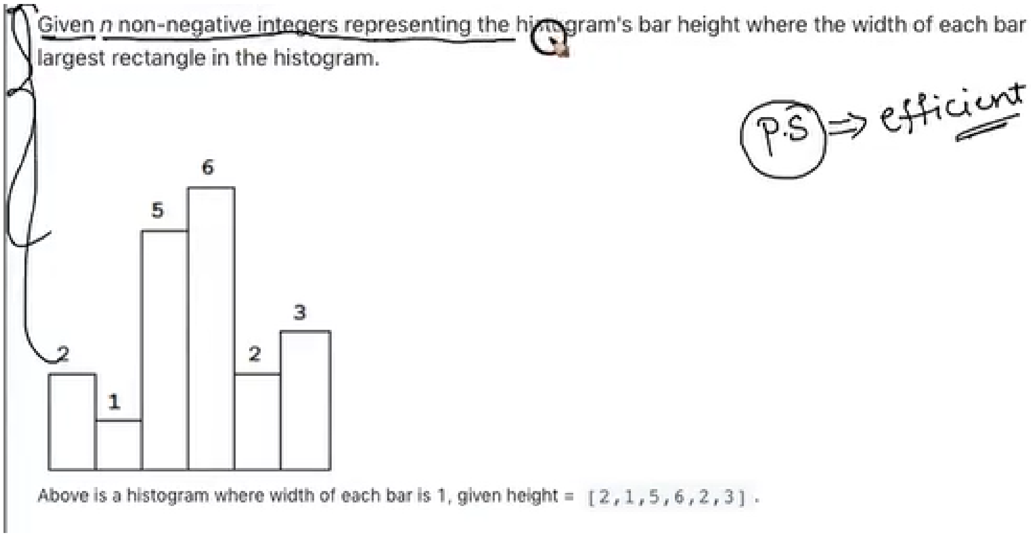
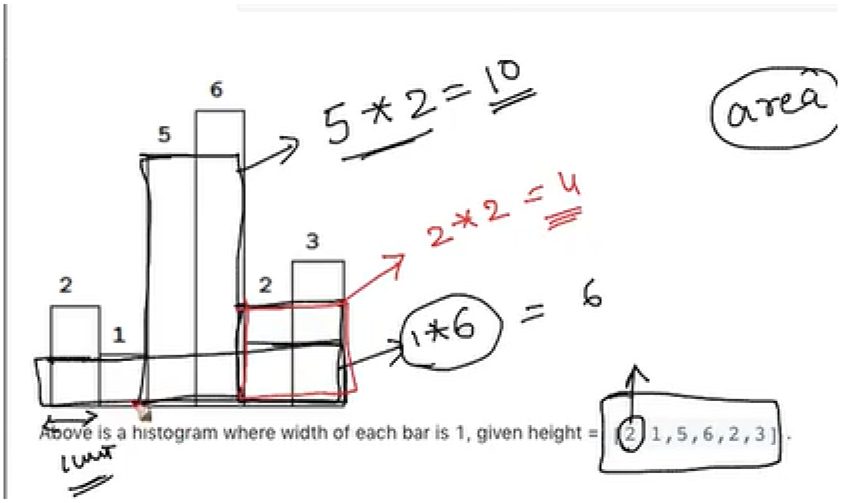
84. Largest Rectangle in Histogram



Given *n* non-negative integers representing the histogram's bar height where the width of each bar is 1, find the area of largest rectangle in the histogram.

Above is a histogram where width of each bar is 1, given height = `[2,1,5,6,2,3]`.

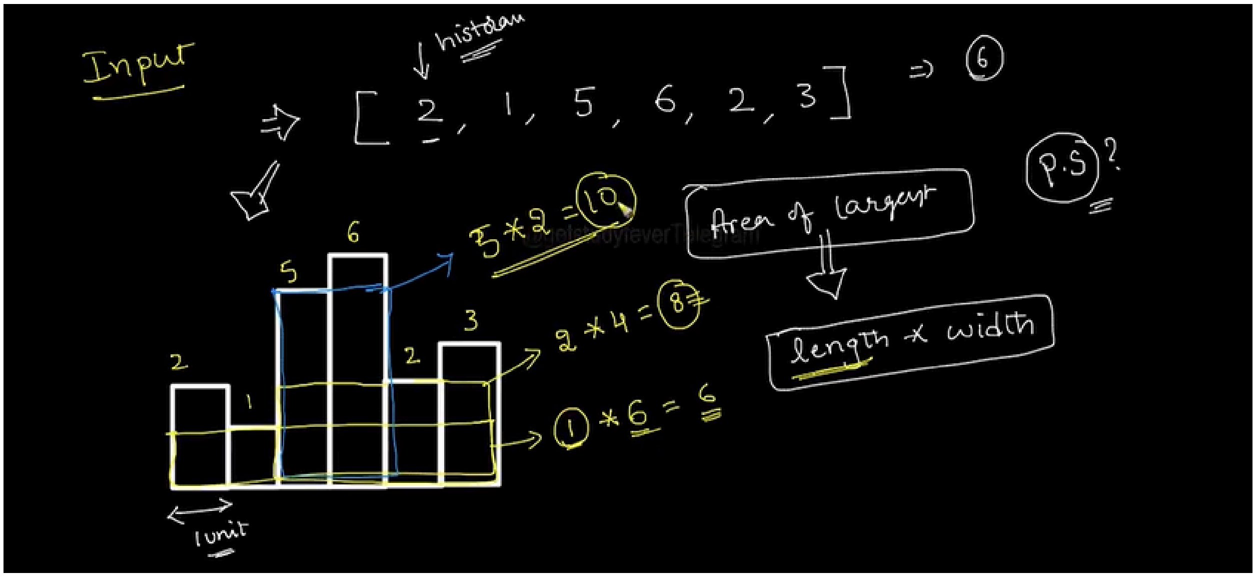
1. Here the area of rectangle is Length * Breadth
2. We need to find area of largest rectangle
3. Here we have 3 possible rectangles



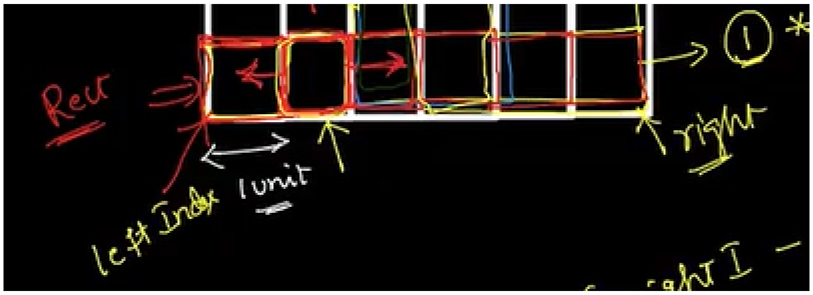
Above is a histogram where width of each bar is 1, given height = `[2,1,5,6,2,3]`.

Solution Approach 1:

1. Here the possible rectangles from our histogram

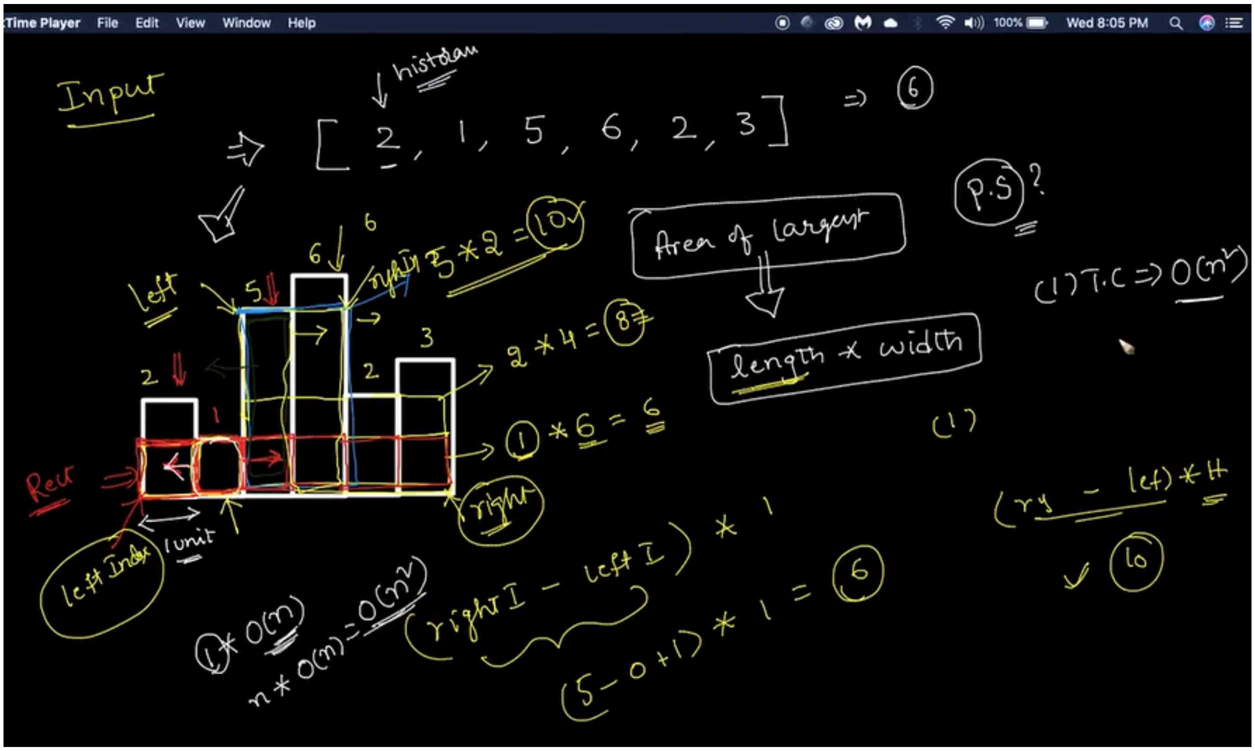


2. In this approach what we will do is …we will find the leftIndex and rightIndex for each index
3. Example here for our first rectangle



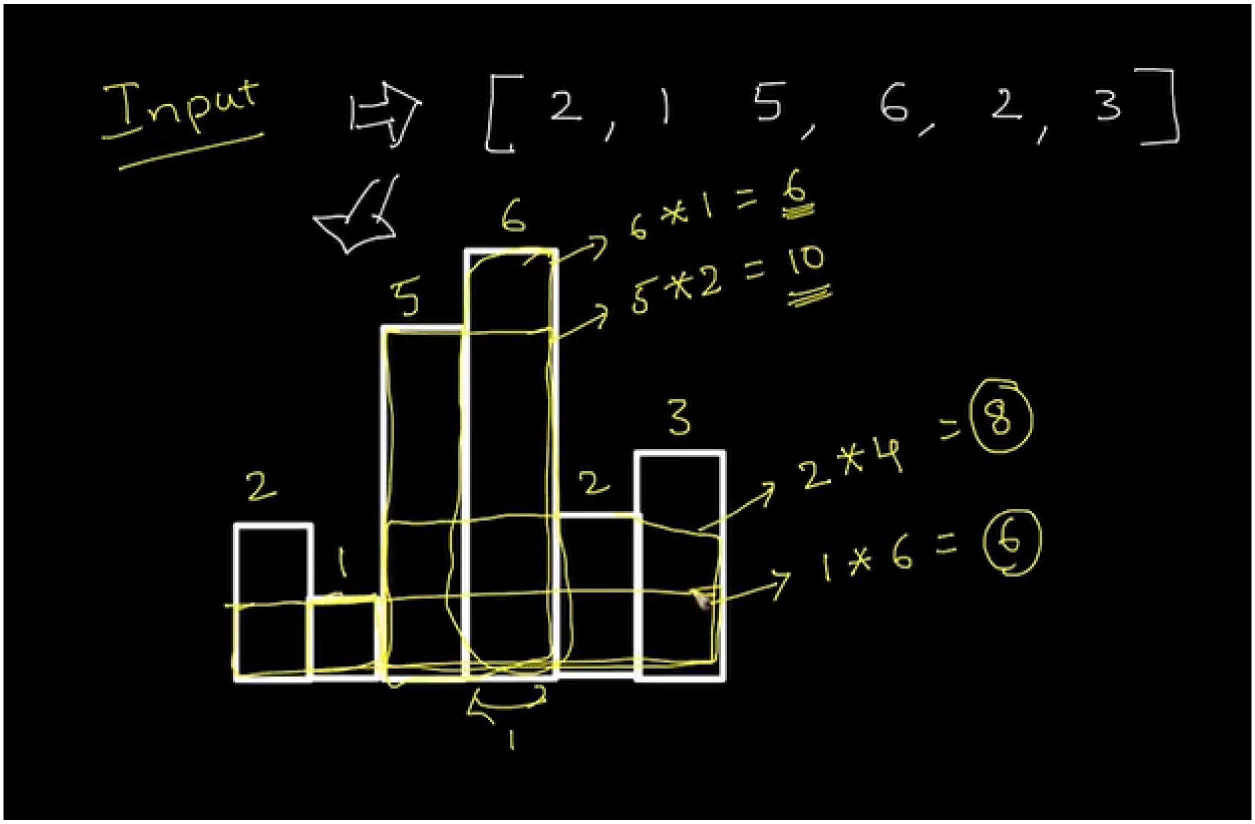
for index1 ..max possible leftIndex is 0 and rightIndex is 5
4. So rectangle at index 1 is given by (Rindex-LIndex+1)*height = (5-0+1)*1 = 6
5. Similarly for index 2 …max possible Leftindex is itself(as every height in left is less than height at index 2) and max possible RightIndex is 3(as height is greater than 5)
6. So area of rectangle for index 2 = (3-2+1)*5 =10

7. The time complexity of this approach is O(n^2)

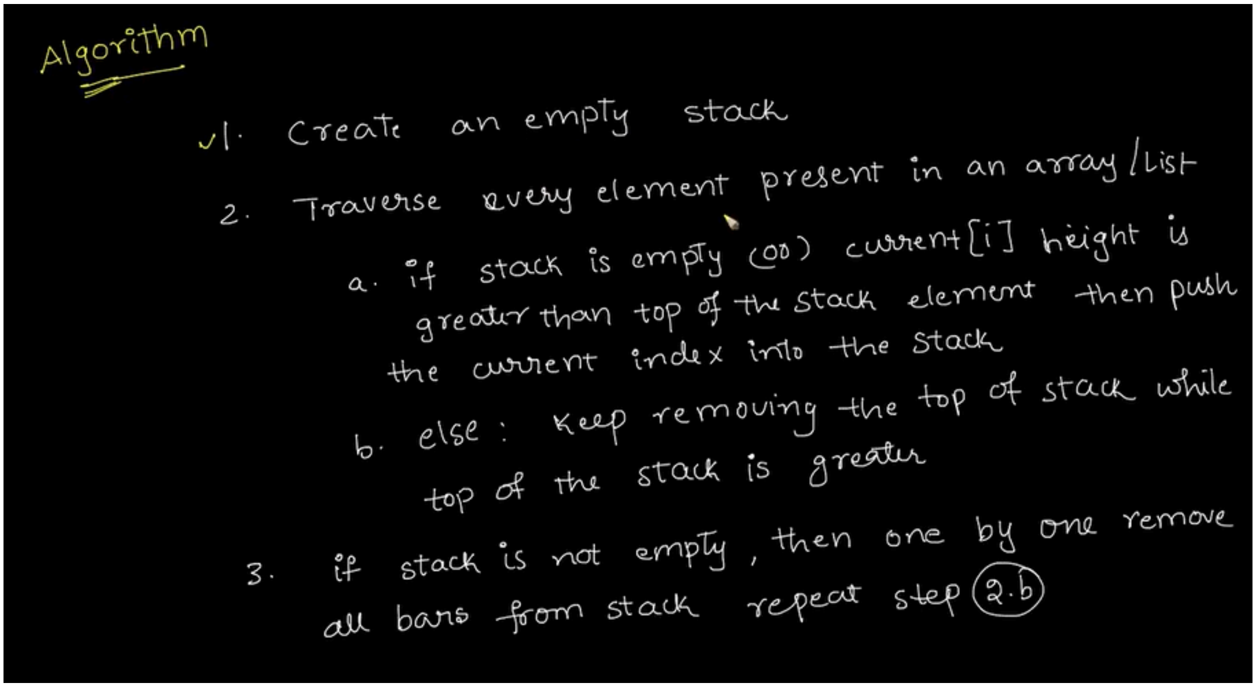


for calculating area of rectangle for one index is O(n) and for n indexes = n*O(n)

2nd Approach

1. 

   here we can notice for every min bar..we have calculated the area of rectangle
2. Here what we have done is



Algorithm

✓1. Create an empty stack

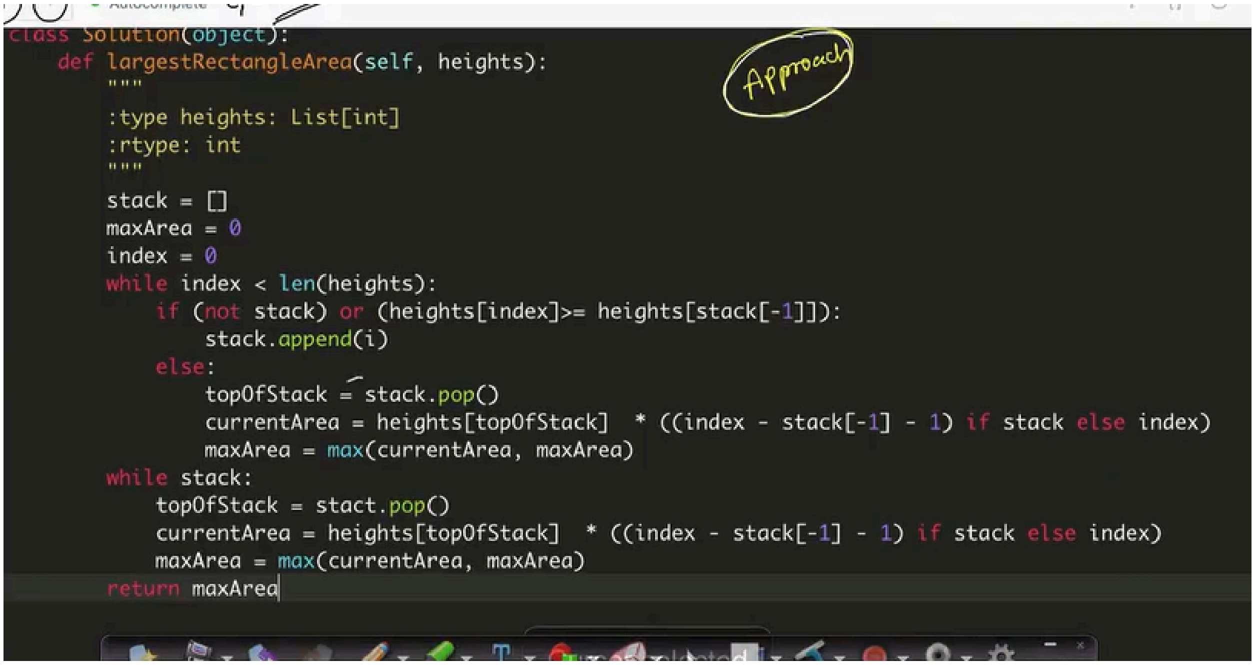
2. Traverse every element present in an array/List

   a. if stack is empty (or) current[i] height is greater than top of the stack element then push the current index into the stack

   b. else : Keep removing the top of stack while top of the stack is greater

3. if stack is not empty, then one by one remove all bars from stack repeat step $\textcircled{2.b}$

3. Python code:



```
class Solution(object):
    def largestRectangleArea(self, heights):
        """
        :type heights: List[int]
        :rtype: int
        """
        stack = []
        maxArea = 0
        index = 0
        while index < len(heights):
            if (not stack) or (heights[index]>= heights[stack[-1]]):
                stack.append(i)
            else:
                topOfStack = stack.pop()
                currentArea = heights[topOfStack]  * ((index - stack[-1] - 1) if stack else index)
                maxArea = max(currentArea, maxArea)
        while stack:
            topOfStack = stact.pop()
            currentArea = heights[topOfStack]  * ((index - stack[-1] - 1) if stack else index)
            maxArea = max(currentArea, maxArea)
        return maxArea
```