Documentation for "Largest Rectangle in Histogram" Solution

Problem Description

Given an array of integers heights representing the histogram's bar heights where the width of each bar is 1, return the area of the largest rectangle in the histogram.

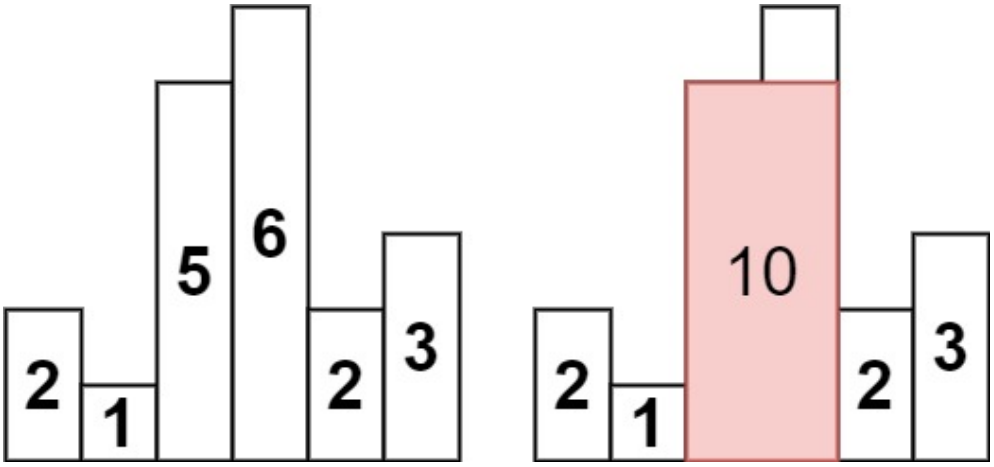
Example 1:

Input: heights = [2, 1, 5, 6, 2, 3]

Output: 10

Explanation:

The histogram is as follows:



The largest rectangle is shown in the red area, which has an area = 10 units.

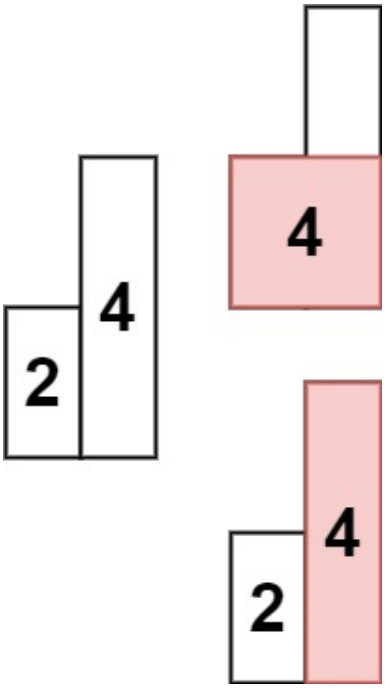
Example 2:

Input: heights = [2, 4]

Output: 4

Explanation:

The histogram is as follows:



The largest rectangle has an area = 4 units.

Constraints

- 1 <= heights.length <= 105
- 0 <= heights[i] <= 104

Solution Explanation

The provided solution uses a stack to efficiently compute the area of the largest rectangle in a histogram. Here's a step-by-step explanation of the approach:

Approach

1. **Initialize an empty stack:** This stack will be used to store the indices of the bars in the histogram.

2. **Initialize max_area to 0:** This variable will keep track of the maximum rectangular area found so far.

3. **Traverse the histogram:**

- For each bar in the histogram, do the following:
- If the stack is empty or the current bar is higher than or equal to the bar at the top of the stack, push the current bar's index onto the stack.
- Otherwise, pop the top of the stack and calculate the area with the popped bar as the smallest (or minimum height) bar. Update max_area if the calculated area is larger.

4. **Pop remaining bars:** After the traversal, there might be some bars left in the stack. Pop each bar and calculate the area as described above.

5. **Return max_area:** The largest rectangular area found in the histogram.

Explanation of Code

Initialization:

- stack is used to keep track of the indices of the bars in the histogram.
- max_area is used to store the maximum area found so far.
- index is used to traverse the histogram.

While Loop:

- The loop continues until all bars are processed.
- If the current bar is higher than the top of the stack, its index is pushed onto the stack.
- If the current bar is lower than the top of the stack, the top of the stack is popped, and the area is calculated considering the popped bar as the smallest bar. The maximum area is updated if necessary.

Final Processing:

After the main loop, any remaining bars in the stack are popped and processed in a similar manner to calculate the area and update the maximum area.

This approach ensures that each bar is pushed and popped from the stack only once, making the time complexity O(n), where n is the number of bars in the histogram.