Monotonic Increasing Stack — Coding Interview Notes (Light Theme)

General Pattern Template

```
def fn(arr):
    stack = []
    ans = 0

    for num in arr:
        # for monotonic decreasing, just flip the > to <
        while stack and stack[-1] > num:
            # do logic (process stack[-1] knowing current num breaks monotonicity)
            stack.pop()
        stack.append(num)

    return ans
```

Concept:
A **monotonic stack** maintains elements in non-decreasing (increasing) or non-increasing (decreasing) order as you scan. When a new element violates the order, pop and process. This enables O(n) solutions for "next/previous greater/smaller" and structure-based problems (histogram area, stock span).

Time Complexity: O(n) amortized — each element is pushed and popped at most once.

Key Ideas

1. Choose stack direction by comparison: increasing (pop while top > curr), decreasing (pop while top < curr).
2. Store *indices* when you need positions (spans/areas), not just values.
3. On pop, the new top is the previous element that *maintains* monotonicity — often the 'previous smaller/greater'.
4. Use sentinels (e.g., extra 0) to flush the stack at the end when needed (histogram problems).

Example 1: Next Greater Element (NGE)

Goal: For each element, find the next element to its right that is greater.
Approach: Maintain a *decreasing* stack of indices; when current is larger, pop and set NGE.

```
def next_greater_elements(nums):
    n = len(nums)
    ans = [-1] * n
    stack = []  # store indices; values decreasing
```

```
    for i, x in enumerate(nums):
        while stack and nums[stack[-1]] < x:
            j = stack.pop()
            ans[j] = x
        stack.append(i)

    return ans
```

Example 2: Largest Rectangle in Histogram

Goal: Given bar heights, find the largest rectangle area.
Approach: Maintain an *increasing* stack of indices. When current height is smaller than the top, pop and compute areas with popped bar as the limiting height.

```
def largest_rectangle_area(heights):
    stack = []  # indices of increasing heights
    max_area = 0
    heights.append(0)  # sentinel to flush

    for i, h in enumerate(heights):
        while stack and heights[stack[-1]] > h:
            height = heights[stack.pop()]
            left = stack[-1] + 1 if stack else 0
            width = i - left
            max_area = max(max_area, height * width)
        stack.append(i)

    heights.pop()
    return max_area
```

Example 3: Stock Span

Goal: For each day, find the number of consecutive days before it with price ≤ today's price.
Approach: Maintain a *decreasing* stack of (price, span). While top.price ≤ curr, merge spans.

```
def stock_span(prices):
    stack = []  # (price, span), prices decreasing
    ans = []
    for p in prices:
        span = 1
        while stack and stack[-1][0] <= p:
            span += stack.pop()[1]
        stack.append((p, span))
        ans.append(span)
    return ans
```

Summary Table

ProblemStack TypeOn Pop MeaningComplexity Next Greater ElementDecreasing (values)Found next greater for popped indexO(n) Largest RectangleIncreasing (indices)Compute area with popped as heightO(n) Stock SpanDecreasing (price,span)Merge spans while ≤ currentO(n)