---

🔷 **WEEK 2 – DAY 9**

**Monotonic Stack, Span Problems, Security Counters**

Sony evaluates:

- Whether you **recognize monotonic patterns**
- Ability to reduce **O(n²) → O(n)**
- Comfort with **index-based stack logic**
- Applying DSA ideas to **real security features**

---

## 1. Monotonic Stack — Core Concept

**Spoken Explanation (What You Say Aloud)**

"A monotonic stack is a stack that maintains its elements in either increasing or decreasing order. Each element is pushed once and popped once, so even though there are nested loops, the overall time complexity remains linear.
This pattern is extremely useful for problems that ask for the next greater, next smaller, span, or nearest boundary."

---

**Types of Monotonic Stack**

| Type | Order (Bottom → Top) | Common Use |
|---|---|---|
| Monotonic Increasing | Increasing | Next Smaller Element |
| Monotonic Decreasing | Decreasing | Next Greater Element, Spans |

---

**Problem Example: Next Greater Element**

**Key Insight (Say This Clearly)**

"Maintain a decreasing stack; resolve elements when a larger value appears."

---

**Python Solution (WITH INPUT)**

```
# File: monotonic_stack_next_greater.py


def nextGreater(nums):
    stack = []
    res = [-1] * len(nums)
```

```
    for i in range(len(nums)):
        while stack and nums[i] > nums[stack[-1]]:
            idx = stack.pop()
            res[idx] = nums[i]
        stack.append(i)

    return res



# INPUT
nums = [4, 5, 2, 25]
print(nextGreater(nums))  # Output: [5, 25, 25, -1]
```

---

**JavaScript Solution (WITH INPUT)**

```
// File: monotonicStackNextGreater.js

function nextGreater(nums) {
    const stack = [];
    const res = new Array(nums.length).fill(-1);

    for (let i = 0; i < nums.length; i++) {
        while (stack.length && nums[i] > nums[stack[stack.length - 1]]) {
            const idx = stack.pop();
            res[idx] = nums[i];
        }
        stack.push(i);
    }
    return res;
}
```

// INPUT

console.log(nextGreater([4, 5, 2, 25])); // [5, 25, 25, -1]

---

**Complexity (Say This in Interview)**

- **Time Complexity:** O(n)
- **Space Complexity:** O(n)

---

**Warm-Up Interview Questions**

1. Why is monotonic stack O(n)?
2. Why store indices instead of values?
3. When does this pattern fail?
4. Difference vs two pointers?
5. Real-world analogy?

---

**Warm-Up Interview Questions (Answered Clearly)**

**1 Why is monotonic stack O(n)?**

**Spoken Answer:**

"Each element is pushed once and popped once.
Even though we use a while loop, total operations are linear."

---

**2 Why store indices instead of values?**

**Spoken Answer:**

"Indices allow us to compute distances, handle duplicates, and place results directly in the output array."

---

**3 When does this pattern fail?**

**Spoken Answer:**

"It fails when the problem requires comparing non-adjacent elements in both directions or when order does not matter."

---

**4 Difference between monotonic stack and two pointers?**

**Spoken Answer:**

“Two pointers move deterministically, usually on sorted data.
Monotonic stack reacts dynamically to future elements and resolves pending comparisons.”

---

### 5️⃣ Real-World Analogy?

**Spoken Answer:**

“Stock prices waiting for a higher future price or days waiting for a warmer temperature.”

---

**Sony One-Line Summary (Memorize)**

“Monotonic stacks efficiently solve nearest greater or smaller problems in linear time by maintaining ordered state.”

---

## 2. Problem 1: Stock Span Problem

**Spoken Explanation (What You Say Aloud)**

“The stock span for a day is the number of consecutive previous days, including today, for which the stock price was less than or equal to today’s price.
A brute-force approach checks backward for each day, which is quadratic.
Using a monotonic decreasing stack, we remove all previous prices that are smaller or equal to today because today dominates them.
This allows us to compute spans in linear time.”

---

**Key Insight (Say This Clearly)**

“I pop all previous prices that are smaller or equal to today’s price, because they are covered by today.”

---

**Recommended Program File Names**

- **Python:** stock_span_monotonic_stack.py
- **JavaScript:** stockSpanMonotonicStack.js

---

**Python Solution (WITH INPUT)**

```
# File: stock_span_monotonic_stack.py


def stockSpan(prices):
    stack = []  # stores indices
    span = [0] * len(prices)
```

```
    for i in range(len(prices)):
        while stack and prices[stack[-1]] <= prices[i]:
            stack.pop()

        span[i] = i + 1 if not stack else i - stack[-1]
        stack.append(i)

    return span



# INPUT
prices = [100, 80, 60, 70, 60, 75, 85]
print(stockSpan(prices))
# Output: [1, 1, 1, 2, 1, 4, 6]
```

---

**JavaScript Solution (WITH INPUT)**

```
// File: stockSpanMonotonicStack.js

function stockSpan(prices) {
    const stack = [];
    const span = new Array(prices.length);

    for (let i = 0; i < prices.length; i++) {
        while (stack.length && prices[stack[stack.length - 1]] <= prices[i]) {
            stack.pop();
        }

        span[i] = stack.length === 0 ? i + 1 : i - stack[stack.length - 1];
        stack.push(i);
    }
```

```
    return span;
}

// INPUT
console.log(stockSpan([100, 80, 60, 70, 60, 75, 85]));
// Output: [1, 1, 1, 2, 1, 4, 6]
```

---

**Complexity (Say This Confidently)**

- **Time Complexity:** O(n)
- **Space Complexity:** O(n)

---

**Sony Follow-Ups**

1. Why use indices?
2. What if prices stream in real time?
3. Can this be done online?
4. Next smaller to left variation?
5. Financial system use case?

---

**1 Why use indices instead of values?**

**Spoken Answer:**

"Indices allow me to calculate span directly as a distance.
They also help handle duplicate values and map results to correct positions."

---

**2 What if prices stream in real time?**

**Spoken Answer:**

"This algorithm is naturally online.
Each new price can be processed immediately using the existing stack without reprocessing past data."

---

**3 Can this be done online?**

**Spoken Answer:**

“Yes. We only depend on past data.
Each incoming price updates the stack and computes span instantly.”

---

### 4 Next smaller to left variation?

**Spoken Answer:**

“Change the comparison from <= to >=.
This converts the stack logic to track the nearest smaller element on the left.”

---

### 5 Financial system use case?

**Spoken Answer:**

“Used in stock analytics platforms to measure momentum, trend strength, and resistance levels in real time.”

---

**Sony One-Line Summary (Memorize This)**

“The stock span problem is efficiently solved using a monotonic decreasing stack that removes dominated prices and computes spans in linear time.”

---

**Interviewer Confidence Tip**

If Sony asks **“Why is this O(n)?”**, say:

“Because each index is pushed once and popped once, so total operations are linear.”

---

## 3. Problem 2: Largest Rectangle in Histogram

**Spoken Explanation (What You Say Aloud)**

“Each bar can potentially be the height of the largest rectangle.
To compute its maximum width, I need to know how far it can extend to the left and right until a smaller bar blocks it.
A monotonic increasing stack helps me find these boundaries efficiently in linear time.”

---

**Why Sony Loves This Problem**

“This problem tests whether you truly understand monotonic stacks, boundary calculation, and amortized analysis — not just memorized patterns.”

---

**Key Insight (Say This Clearly)**

“For each bar, I compute the largest rectangle where that bar is the smallest height by finding the first smaller bar on the left and right.”

---

**Python Solution (WITH INPUT)**

```
# File: largest_rectangle_histogram.py


def largestRectangleArea(heights):
    stack = []
    max_area = 0
    heights.append(0)  # sentinel to flush stack

    for i in range(len(heights)):
        while stack and heights[i] < heights[stack[-1]]:
            h = heights[stack.pop()]
            width = i if not stack else i - stack[-1] - 1
            max_area = max(max_area, h * width)
        stack.append(i)

    return max_area



# INPUT
heights = [2, 1, 5, 6, 2, 3]
print(largestRectangleArea(heights))
# Output: 10
```

---

**JavaScript Solution (WITH INPUT)**

```
// File: largestRectangleHistogram.js


function largestRectangleArea(heights) {
    const stack = [];
```

```
    heights.push(0); // sentinel
    let maxArea = 0;

    for (let i = 0; i < heights.length; i++) {
        while (stack.length && heights[i] < heights[stack[stack.length - 1]]) {
            const h = heights[stack.pop()];
            const width = stack.length === 0 ? i : i - stack[stack.length - 1] - 1;
            maxArea = Math.max(maxArea, h * width);
        }
        stack.push(i);
    }
    return maxArea;
}

// INPUT
console.log(largestRectangleArea([2, 1, 5, 6, 2, 3]));
// Output: 10
```

---

**Complexity (Say This Confidently)**

- **Time Complexity:** O(n)
- **Space Complexity:** O(n)

“Each index is pushed and popped once.”

---

**Sony Follow-Ups (Very Common)**

1. Why add sentinel 0?
2. How is width calculated?
3. Can you explain with an example?
4. Largest rectangle in binary matrix?
5. Real-world application?

---

**Sony Follow-Ups (Very Common — Answer Like This)**

---

### 1 Why add sentinel 0?

**Spoken Answer:**

"The sentinel ensures that all remaining bars in the stack are processed.
Without it, increasing sequences would never trigger rectangle calculation."

---

### 2 How is width calculated?

**Spoken Answer:**

"When a bar is popped, it becomes the smallest bar.
The current index i is the first smaller bar on the right, and the new top of the stack is the first smaller bar on the left."

Formula:

- If stack empty → width = i
- Else → width = i - stack.top - 1

---

### 3 Explain with an example

Example: [2, 1, 5, 6, 2, 3]

**Spoken Explanation:**

"When height 2 appears at index 4, bars 6 and 5 are popped.
For height 5, left boundary is index 2 and right boundary is index 4, giving width 2 and area 10."

---

### 4 Largest rectangle in binary matrix?

**Spoken Answer:**

"Convert each row into a histogram of consecutive 1s and apply this algorithm row by row."

(This is a **classic Sony follow-up**.)

---

### 5 Real-world application?

**Spoken Answer:**

"Used in skyline analysis, image processing, database query optimization, and load-balancing visualizations."

---

**One-Line Sony Summary (Memorize This)**

“Largest rectangle in histogram uses a monotonic increasing stack to compute left and right boundaries in linear time.”

---

**Interview Confidence Booster**

If the interviewer asks **“Why not brute force?”**, say:

“Brute force checks all subarrays, which is $O(n^2)$.
Monotonic stack avoids recomputation by maintaining valid boundaries.”

---

**4. Sony-Level Variations (Expect One)**

1. Max rectangle in binary matrix
2. Daily temperatures
3. Next greater in circular array
4. Remove K digits
5. Sum of subarray minimums

---

**1. Max Rectangle in Binary Matrix**

**Spoken Explanation (Say This)**

“Each row is treated as a histogram of consecutive 1s.
For every row, I update heights and reuse the largest rectangle in histogram logic.”

---

**Python — max_rectangle_binary_matrix.py**

```
def maximalRectangle(matrix):
    if not matrix:
        return 0

    heights = [0] * len(matrix[0])
    max_area = 0

    for row in matrix:
        for i in range(len(row)):
            heights[i] = heights[i] + 1 if row[i] == "1" else 0
        max_area = max(max_area, largestRectangleArea(heights))
```

```
    return max_area


def largestRectangleArea(heights):
    stack = []
    heights.append(0)
    area = 0

    for i in range(len(heights)):
        while stack and heights[i] < heights[stack[-1]]:
            h = heights[stack.pop()]
            w = i if not stack else i - stack[-1] - 1
            area = max(area, h * w)
        stack.append(i)

    heights.pop()
    return area


# INPUT
matrix = [
    ["1","0","1","0","0"],
    ["1","0","1","1","1"],
    ["1","1","1","1","1"],
    ["1","0","0","1","0"]
]
print(maximalRectangle(matrix))
```

---

**JavaScript — maxRectangleBinaryMatrix.js**

```
function maximalRectangle(matrix) {
```

```
    if (!matrix.length) return 0;
    const heights = new Array(matrix[0].length).fill(0);
    let maxArea = 0;

    for (const row of matrix) {
        for (let i = 0; i < row.length; i++) {
            heights[i] = row[i] === "1" ? heights[i] + 1 : 0;
        }
        maxArea = Math.max(maxArea, largestRectangleArea(heights));
    }
    return maxArea;
}
```

---

**Sony Follow-Ups**

- **Why reuse histogram?** → Converts 2D → multiple 1D problems
- **Time complexity?** → O(rows × cols)
- **Real use?** → Image processing, grid analytics

---

**2. Daily Temperatures**

**Spoken Explanation**

"I use a monotonic decreasing stack of indices.
When a warmer day appears, I resolve all colder days before it."

---

**Python — daily_temperatures.py**

```
def dailyTemperatures(temps):
    stack = []
    res = [0] * len(temps)

    for i in range(len(temps)):
        while stack and temps[i] > temps[stack[-1]]:
            idx = stack.pop()
```

```
            res[idx] = i - idx
        stack.append(i)

    return res


# INPUT
print(dailyTemperatures([73,74,75,71,69,72,76,73]))
```

---

**JavaScript — dailyTemperatures.js**

```
function dailyTemperatures(temps) {
    const stack = [];
    const res = new Array(temps.length).fill(0);

    for (let i = 0; i < temps.length; i++) {
        while (stack.length && temps[i] > temps[stack[stack.length - 1]]) {
            const idx = stack.pop();
            res[idx] = i - idx;
        }
        stack.push(i);
    }
    return res;
}
```

---

**Sony Follow-Ups**

- **Why indices?** → To calculate distance
- **Streaming?** → Yes, online stack update
- **Real use?** → Weather forecasting systems

---

**3. Next Greater Element in Circular Array**

**Spoken Explanation**

“I simulate circularity by traversing the array twice and using modulo indexing.”

---

**Python — next_greater_circular.py**

```
def nextGreaterElements(nums):
    n = len(nums)
    res = [-1] * n
    stack = []

    for i in range(2 * n):
        while stack and nums[i % n] > nums[stack[-1]]:
            res[stack.pop()] = nums[i % n]
        if i < n:
            stack.append(i)

    return res



# INPUT
print(nextGreaterElements([1,2,1]))
```

---

**JavaScript — nextGreaterCircular.js**

```
function nextGreaterElements(nums) {
    const n = nums.length;
    const res = new Array(n).fill(-1);
    const stack = [];

    for (let i = 0; i < 2 * n; i++) {
        while (stack.length && nums[i % n] > nums[stack[stack.length - 1]]) {
            res[stack.pop()] = nums[i % n];
        }
        if (i < n) stack.push(i);
```

```
    }
    return res;
}
```

---

**Sony Follow-Ups**

- **Why double traversal?** → Simulates circular array
- **Time complexity?** → Still O(n)
- **Use case?** → Circular buffers, schedulers

---

**4. Remove K Digits**

**Spoken Explanation**

"I greedily remove larger digits before smaller ones using a monotonic increasing stack."

---

**Python — remove_k_digits.py**

```
def removeKdigits(num, k):
    stack = []

    for d in num:
        while stack and k > 0 and stack[-1] > d:
            stack.pop()
            k -= 1
        stack.append(d)

    result = ''.join(stack[:len(stack)-k]).lstrip('0')
    return result if result else "0"



# INPUT
print(removeKdigits("1432219", 3))
```

---

**JavaScript — removeKDigits.js**

```
function removeKdigits(num, k) {
  const stack = [];

  for (const d of num) {
    while (stack.length && k > 0 && stack[stack.length - 1] > d) {
      stack.pop();
      k--;
    }
    stack.push(d);
  }

  let res = stack.slice(0, stack.length - k).join("").replace(/^0+/, "");
  return res || "0";
}
```

---

**Sony Follow-Ups**

- **Why greedy works?** → Smaller digits earlier minimize number
- **Edge cases?** → Leading zeros, k ≥ n
- **Real use?** → Financial number optimization

---

**5. Sum of Subarray Minimums**

**Spoken Explanation**

"Each element contributes as the minimum in several subarrays.
I calculate its left and right span using monotonic stacks."

---

**Python — sum_subarray_minimums.py**

```
def sumSubarrayMins(arr):
  mod = 10**9 + 7
  stack = []
  res = 0
  arr.append(0)
```

```
    for i in range(len(arr)):
        while stack and arr[i] < arr[stack[-1]]:
            mid = stack.pop()
            left = mid - (stack[-1] if stack else -1)
            right = i - mid
            res += arr[mid] * left * right
        stack.append(i)

    return res % mod



# INPUT
print(sumSubarrayMins([3,1,2,4]))
```

---

**JavaScript — sumSubarrayMinimums.js**

```
function sumSubarrayMins(arr) {
    const mod = 1e9 + 7;
    const stack = [];
    let res = 0;
    arr.push(0);

    for (let i = 0; i < arr.length; i++) {
        while (stack.length && arr[i] < arr[stack[stack.length - 1]]) {
            const mid = stack.pop();
            const left = mid - (stack.length ? stack[stack.length - 1] : -1);
            const right = i - mid;
            res = (res + arr[mid] * left * right) % mod;
        }
        stack.push(i);
    }
```

```
    return res;
}
```

---

**Sony Follow-Ups**

- **Why multiplication?** → Counts subarrays where element is minimum
- **Why mod?** → Prevent overflow
- **Used where?** → Performance analytics, pricing models

---

**Sony Final Tip (Memorize)**

"If a problem asks for next greater/smaller, spans, ranges, or contributions — think monotonic stack."

---

**5. Project: Add Failed Login Counter (Security Feature)**

**Interview Framing**

"Design a mechanism to track failed login attempts and prevent brute-force attacks."

---

**Core Requirements**

- Count consecutive failures
- Reset on success
- Time-window aware
- Enable account lockout

---

**Simple Database Schema**

```
class FailedLoginCounter(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String(100), unique=True)
    failed_count = db.Column(db.Integer, default=0)
    last_failed_at = db.Column(db.DateTime)
```

---

**Logic Flow (Say This)**

"On each failed login, increment the counter. On success, reset it. If failures exceed a threshold within a time window, temporarily lock the account."

---

**Helper Functions**

```
def record_failure(username):
    record = FailedLoginCounter.query.filter_by(username=username).first()
    if not record:
        record = FailedLoginCounter(username=username, failed_count=1)
        db.session.add(record)
    else:
        record.failed_count += 1
        record.last_failed_at = datetime.utcnow()
    db.session.commit()
def reset_failures(username):
    record = FailedLoginCounter.query.filter_by(username=username).first()
    if record:
        record.failed_count = 0
        db.session.commit()
```

---

**Sony Follow-Ups on Failed Login Counter**

1. Fixed threshold vs time-based?
2. Distributed system handling?
3. Redis vs DB?
4. How to avoid user enumeration?
5. What about CAPTCHA integration?

---

**What Sony Wants You to Say**

“I’d combine a counter with a time window and store it in Redis for fast access, falling back to DB for persistence.”