LeetCode 1944 — Number of Visible People in a Queue

🔗 https://leetcode.com/problems/number-of-visible-people-in-a-queue/

1. Problem Title & Link

Title: LeetCode 1944: Number of Visible People in a Queue
Link: https://leetcode.com/problems/number-of-visible-people-in-a-queue/

2. Problem Statement — Simple Explanation

You are given heights[], where each element is a person in a line.
A person **i** can see person **j (j > i)** if:

1. **j is to the right of i**
2. There is no one between them taller than person j
3. Visibility stops when you meet someone **taller or equal to current person**

Return an array where ans[i] = how many people person i can see.

📌 People block visibility when they are **taller or equal height**.

3. Examples (Input → Output)

Example 1

Input: heights = [10,6,8,5,11,9]
Output: [3,1,2,1,1,0]
Explanation:

- 10 can see: 6, 8, **11 (stopped)** → 3
- 6 → 8, **11** → 1
- 8 → 5, 11 → 2
- 5 → 11 → 1
- 11 → 9 → 1
- 9 sees nobody → 0

Example 2

Input: heights = [5,1,2,3,10]
Output: [4,1,1,1,0]

4. Constraints

- 1 ≤ n ≤ 10^5
- 1 ≤ heights[i] ≤ 10^5
- Must solve faster than $O(n^2)$
- Required: **O(n)** (using **Monotonic Stack**)

5. Core Concept (Pattern / Topic)

⭐ **Monotonic Stack (Decreasing)**

Because we need to see how many people are visible towards the right efficiently.

6. Thought Process — Step-by-Step

❌ **Brute Force (slow)**

For each i, scan right until blocked → **O(n²)** → TLE for n = 100k.

✔ Optimized — Monotonic Stack from Right → Left

We traverse from rightmost person:

1. Maintain a **monotonic decreasing stack** (heights only)
2. For each person:
   - Pop shorter people (because they are visible AND won't block others)
   - Number of popped people += visible count
   - If stack still has someone → they are visible blocker → count += 1
3. Push current height into stack

This gives **O(n)** because each person is pushed & popped once.

7. Visual Diagram

For heights = [10,6,8,5,11,9]

Process **from right → left**

```
i=5: 9  → sees none   stack=[9]
i=4: 11 → sees 9      stack=[11]
i=3: 5  → sees 11     stack=[11,5]
i=2: 8  → pops 5, sees 11  stack=[11,8]
i=1: 6  → sees 8, sees 11  stack=[11,8,6]
i=0:10 → sees 6,8,11 stack=[11,10]
Output = [3,1,2,1,1,0]
```

8. Pseudocode

```
ans[n]
stack = empty

for i from n-1 to 0:
    count = 0
    while stack not empty AND stack.top < heights[i]:
```

```
        stack.pop()
        count++

    if stack not empty:
        count++   // sees first taller/equal person

    ans[i] = count
    stack.push(heights[i])

return ans
```

9. Code Implementation

✔ Python — O(n)

```python
class Solution:
    def canSeePersonsCount(self, heights):
        stack = []
        res = [0] * len(heights)

        for i in range(len(heights)-1, -1, -1):
            count = 0
            while stack and stack[-1] < heights[i]:
                stack.pop()
                count += 1
            if stack:
                count += 1
            res[i] = count
            stack.append(heights[i])

        return res
```

✔ Java — O(n)

```java
class Solution {
    public int[] canSeePersonsCount(int[] heights) {
        int n = heights.length;
        int[] ans = new int[n];
        Stack<Integer> stack = new Stack<>();

        for (int i = n-1; i >= 0; i--) {
            int count = 0;

            while(!stack.isEmpty() && stack.peek() < heights[i]) {
```

```
                stack.pop();
                count++;
            }
            if(!stack.isEmpty()) count++;

            ans[i] = count;
            stack.push(heights[i]);
        }
        return ans;
    }
}
```

10. Time & Space Complexity

Operation	Complexity
Overall Time	O(n)
Space	**O(n)** stack

Each height pushed & popped ≤1 time.

11. Common Mistakes / Edge Cases

❌ Scanning range for each element → O(n^2)

❌ Forgetting visibility for *first taller/equal* person

❌ Using monotonic increasing stack — **wrong direction**

Edge cases:

✔ strictly decreasing → each sees 1
✔ strictly increasing → each sees all to right
✔ equal values — visibility stops on equal

12. Dry Run Table

For: heights = [5,1,2,3,10]

i	h	stack before	popped	sees blocker	ans
4	10	[]	0	no	0
3	3	[10]	0	yes	1
2	2	[10,3]	0	yes	1
1	1	[10,3,2]	0	yes	1
0	5	[10,3,2,1]	3	yes	4

Output → [4,1,1,1,0]

13. Use Cases

- Visibility in queue/people/buildings
- Stock span variants
- Monotonic decreasing stack pattern

14. Common Traps

⚠ Using < incorrectly instead of <=

⚠ Counting pops but forgetting final visible person

⚠ Processing from left instead of right

15. Builds To Related Problems

- LC 739 — Daily Temperatures
- LC 84 — Histogram Largest Rectangle
- LC 901 — Stock Span
- LC 2866 — Beautiful Towers

16. Alternate Approaches

Approach	Time	Notes
Monotonic Stack ✔	O(n)	Best & standard
Bruteforce	$O(n^2)$	TLE
Segment tree / RMQ	O(n log n)	Overkill

17. Why This Works (1-liner)

We eliminate shorter people first & the first taller/equal person blocks the view — monotonic stack simulates visibility.

18. Follow-Up Questions

- Modify to count **distance** instead of people
- Count visibility in **both directions**
- Return **indices** of visible, not count
- Do this in streaming input (online queue)