Leaders in Array

---

1. 💡 Understand the Problem

- **Read & Identify:** Find all **leaders** in an array.
- **Goal:** A leader is an element strictly greater than all elements to its right. The rightmost element is always a leader.
- **Paraphrase:** For each element, check if no element to its right is greater or equal.

---

2. 📋 Input, Output, & Constraints

- **Input:** Integer array `nums` of size `n`
- **Output:** List of leader elements in the order they appear

Constraints:

- 1 ≤ n ≤ 10^5
- -10^9 ≤ nums[i] ≤ 10^9
- Target time complexity: O(n)

---

3. 🧪 Examples & Edge Cases

Example 1 (Normal Case):
Input: `[1, 2, 3, 2]` → Output: `[3, 2]`

Example 2 (Tricky Case):
Input: `[16, 17, 4, 3, 5, 2]` → Output: `[17, 5, 2]`

Example 3 (Edge Case):
Input: `[5]` → Output: `[5]`

Edge Case Checklist:

- Single element → leader
- All increasing → last element leader
- All decreasing → all elements leaders
- Large n → performance check

---

4. 🧠 Approach 1: Brute Force

Idea: Check all elements to the right for each element

Pseudocode:

```
function findLeadersBrute(nums):
leaders = []
for i = 0 to n-1:
isLeader = true
for j = i+1 to n-1:
if nums[j] >= nums[i]:
isLeader = false
if isLeader:
leaders.append(nums[i])
return leaders
```

Java Code:

```java
import java.util.*;

class LeadersBruteForce {
    public static List<Integer> findLeadersBrute(int[] nums) {
        List<Integer> leaders = new ArrayList<>();
        for (int i = 0; i < nums.length; i++) {
            boolean isLeader = true;
            for (int j = i + 1; j < nums.length; j++) {
                if (nums[j] >= nums[i]) isLeader = false;
            }
            if (isLeader) leaders.add(nums[i]);
        }
        return leaders;
    }
}
```

Complexity: Time: O(n^2) Space: O(1)

---

5. ✨ Approach 2: Optimized Solution

- Idea / Optimization: Scan right-to-left, track the maximum element so far (Greedy approach)

Pseudocode:

```
function findLeadersOptimized(nums):
    leaders = []
    maxRight = -∞
    for i = n-1 downto 0:
        if nums[i] > maxRight:
            leaders.addFront(nums[i])
            maxRight = nums[i]
    return leaders
```

Java Code:

```
import java.util.*;

class LeadersOptimized {
    public static List<Integer> findLeadersOptimized(int[] nums) {
        List<Integer> leaders = new ArrayList<>();
        int maxRight = Integer.MIN_VALUE;
        for (int i = nums.length - 1; i >= 0; i--) {
            if (nums[i] > maxRight) {
                leaders.add(nums[i]);
                maxRight = nums[i];
            }
        }
        Collections.reverse(leaders);
        return leaders;
    }
}
```

Complexity: Time O(n) Space O(1)

---

6. ☑ Justification / Proof of Optimality

- The right-to-left scan guarantees all elements larger than maxRight are leaders.
- Greedy choice is always correct because once maxRight is updated, no smaller element can be a leader. *Meets O(n) time and O(1) extra space requirement.

7. 🏷 Patterns & Tags

- Data Structures: Array
- Algorithms / Techniques: Greedy, Reverse Traversal

---

8. ⏭ Variants / Follow-Ups

- Count leaders instead of listing them
- Leaders in a circular array
- Leaders in a 2D matrix (row-wise / column-wise)