LeetCode 496 — Next Greater Element I

1. Problem Title & Link

- **Title:** LeetCode 496 — Next Greater Element I
- **Link:** https://leetcode.com/problems/next-greater-element-i/

2. Problem Statement (Short Summary)

You are given two arrays:

- nums1 — a subset of nums2
- nums2 — distinct integers

For each element in nums1, find its **next greater element** in nums2.

Next Greater Element = first element to the **right** in nums2 that is **strictly greater**.

If none exists → return -1.

3. Examples (Input → Output)

Example 1

Input: nums1 = [4,1,2], nums2 = [1,3,4,2]

Output: [-1,3,-1]

Example 2

Input: nums1 = [2,4], nums2 = [1,2,3,4]

Output: [3,-1]

4. Constraints

- 1 <= nums1.length <= nums2.length <= 1000
- All values are unique
- nums1 ⊆ nums2
- Must return answers in order of nums1

5. Core Concept (Pattern / Topic)

Monotonic Stack (Decrease Stack) + Hashmap

Scan nums2 from left to right and maintain a **monotonic decreasing stack**:

- When a number is **greater** than stack top → it is the “next greater” for that top.
- Pop stack, assign its next greater in a hashmap.
- Push current number.

Finally, use hashmap for fast lookup for nums1.

6. Thought Process (Step-by-Step Explanation)

1. Traverse nums2
2. Maintain a stack where elements are **waiting** for next greater element
3. For each num:
    - While stack NOT empty and num > stack.top:
        - stack.top has found its next greater → store: nextGreater[x] = num
        - pop stack
    - Push current num
4. After finishing → remaining stack elements have no next greater → map them to -1
5. Answer queries for nums1 using hashmap

This gives **O(n)** time complexity.

7. Visual / Intuition Diagram

Example:

nums2 = [1, 3, 4, 2]

stack = []

Process:

- 1 → stack = [1]
- 3 → 3 > 1 → nextGreater[1] = 3, stack = [3]
- 4 → 4 > 3 → nextGreater[3] = 4, stack = [4]
- 2 → 2 < 4 → stack = [4,2]

Remaining:

- nextGreater[4] = -1
- nextGreater[2] = -1

8. Pseudocode

```
map = {}
stack = empty

for num in nums2:
    while stack not empty and num > stack.top:
        map[stack.pop()] = num
    push num into stack

while stack not empty:
    map[stack.pop()] = -1

result = []
for num in nums1:
    result.add(map[num])
```

```
return result
```

9. Code Implementation

✅ Python

```
class Solution:
    def nextGreaterElement(self, nums1: List[int], nums2: List[int]) -> List[int]:
        stack = []
        nge = {}  # map: num -> next greater

        for num in nums2:
            while stack and num > stack[-1]:
                nge[stack.pop()] = num
            stack.append(num)

        # remaining have no next greater
        while stack:
            nge[stack.pop()] = -1

        return [nge[n] for n in nums1]
```

✅ Java

```
class Solution {
    public int[] nextGreaterElement(int[] nums1, int[] nums2) {
        Map<Integer, Integer> map = new HashMap<>();
        Stack<Integer> stack = new Stack<>();

        for (int num : nums2) {
            while (!stack.isEmpty() && num > stack.peek()) {
                map.put(stack.pop(), num);
            }
            stack.push(num);
        }

        while (!stack.isEmpty()) {
            map.put(stack.pop(), -1);
        }

        int[] res = new int[nums1.length];
        for (int i = 0; i < nums1.length; i++) {
            res[i] = map.get(nums1[i]);
        }

        return res;
    }
}
```

10. Time & Space Complexity

Metric	Complexity
Time	O(n + m) — every element pushed + popped once
Space	O(n) — stack + hashmap

11. Common Mistakes / Edge Cases

❌ Mistakes:

- Using nested loops → O(n^2)
- Confusing “next greater” with “greater anywhere”
- Forgetting to set remaining elements to -1
- Returning results in wrong order for nums1

Edge cases:

- nums1 = nums2
- strictly decreasing array
- nums2 of size 1

12. Detailed Dry Run (Step-by-Step)

Input:

nums1 = [4,1,2]

nums2 = [1,3,4,2]

Process nums2:

1. num = 1
   stack = [1]
2. num = 3
   3 > 1 → nextGreater[1] = 3
   stack = [3]

3. num = 4
   4 > 3 → nextGreater[3] = 4
   stack = [4]
4. num = 2
   2 < 4 → push
   stack = [4,2]

Remaining:

- nextGreater[2] = -1
- nextGreater[4] = -1

Final map:

1 → 3

3 → 4

4 → -1

2 → -1

Answer for nums1:

[4→-1, 1→3, 2→-1] = [-1, 3, -1]

13. Common Use Cases

- Stock next greater price
- Next warmer day
- Bracket matching variants
- Stack-based monotonic patterns
- Scheduling tasks

14. Common Traps

- Forgetting that nums1 order must remain unchanged
- Misusing stack (push > pop order incorrect)
- Looking left instead of right

15. Builds To (Related Problems)

- **LC 503** — Next Greater Element II (circular array)
- **LC 739** — Daily Temperatures
- **LC 901** — Online Stock Span
- **LC 84** — Largest Rectangle in Histogram

16. Alternate Approaches + Comparison

Approach	Time	Space	Notes
Monotonic Stack	O(n)	O(n)	✔ Best
Brute Force	$O(n^2)$	O(1)	Too slow
Reverse scanning + stack	O(n)	O(n)	Similar but more steps

17. Why This Solution Works (Short Intuition)

By maintaining a **monotonic decreasing stack**,
every time a bigger number appears,
it becomes the next greater element for all smaller numbers on the stack.

Only one pass needed.

18. Variations / Follow-Up Questions

- What if nums2 has duplicates?
- What if we want next **smaller** element?
- What if we want previous greater element?
- Circular version (LC 503)?