Next Greater Element I Documentation

1. **Problem Statement**

   Given two distinct 0-indexed integer arrays nums1 and nums2 where nums1 is a subset of nums2, the task is to find the next greater element of each element in nums1 based on nums2.

   For each nums1[i], find the index j such that nums1[i] == nums2[j], then determine the next element greater than nums2[j] in nums2 (to its right). If no such element exists, return -1.

2. **Intuition**

   - Since all elements in nums2 are unique, we can efficiently track the next greater element for each using a stack.
   - We first preprocess nums2 to store the next greater element of each value.
   - Then, we simply look up each element in nums1 using this precomputed mapping.

3. **Key Observations**

   - Each element in nums2 appears exactly once.
   - We only need to look to the right side of each number in nums2 to find its next greater element.
   - Stack can be used to efficiently track and resolve these relationships in linear time.

4. **Approach**

   - Initialize:
     - An empty stack to keep elements for which we haven't found the next greater.
     - A dictionary next_greater to store the result for each element in nums2.

- Traverse nums2:
    - For each number:
        - While the stack is not empty and the current number is greater than the top of the stack:
            - Pop the element from the stack.
            - Map it in the next_greater dictionary as having the current number as its next greater.
        - Push the current number to the stack.
- Finalize:
    - For any remaining elements in the stack, assign -1 as they have no greater element.
- Build the result:
    - For each element in nums1, use the dictionary to look up its next greater value.

5. Edge Cases

- All elements in nums1 are at the end of nums2 → Output will be all -1.
- Elements in nums2 are in strictly decreasing order → All values in nums1 will have -1.
- nums1 equals nums2 → Full next greater mapping is needed.

6. Complexity Analysis

Time Complexity:

- O(n + m), where n = len(nums1) and m = len(nums2):
    - Each element is pushed and popped once from the stack.
    - Lookup in dictionary is O(1).

Space Complexity:

- O(m) for:
    - The stack (at most m elements).
    - The next_greater dictionary storing results for up to m elements.

7. Alternative Approaches

Brute-force (Nested Loops)

- For each nums1[i], search its index in nums2, then scan to the right to find the next greater.
- Time Complexity: O(n × m)
- Not scalable for large inputs.

8. Test Cases

✅ Test Case 1:

```
nums1 = [4, 1, 2]
nums2 = [1, 3, 4, 2]
# Output: [-1, 3, -1]
```

✅ Test Case 2:

```
nums1 = [2, 4]
nums2 = [1, 2, 3, 4]
# Output: [3, -1]
```

✅ Test Case 3 (All decreasing):

```
nums1 = [3, 2, 1]
nums2 = [3, 2, 1]
# Output: [-1, -1, -1]
```

✅ Test Case 4 (All increasing):

```
nums1 = [1, 2]
nums2 = [1, 2, 3, 4]
# Output: [2, 3]
```

9. Final Thoughts

- This problem is a classic use case for monotonic stacks, frequently seen in "next greater/smaller element" problems.
- Understanding this pattern helps in solving a variety of stack-based problems efficiently.
- Always prefer precomputing and hashing (dictionary) when lookups are needed repeatedly.