Binary Search: Duplicate Elements (Right-most Insertion Point) — Coding Interview Notes (Light Theme)

General Pattern Template

```
def fn(arr, target):
    left = 0
    right = len(arr)
    while left < right:
        mid = (left + right) // 2
        if arr[mid] > target:
            right = mid
        else:
            left = mid + 1

    return left
```

Concept:
This variant of **Binary Search** finds the **rightmost insertion point** for a target in a sorted array (possibly with duplicates). It behaves like Python's bisect_right: it returns the index immediately after the last occurrence of the target.

Key Property: After the loop, left equals the smallest index where arr[left] > target or len(arr) if none.
Time Complexity: O(log n) **Space Complexity:** O(1)

Key Ideas

1. This is the **right-bound** version of binary search.
2. Use condition arr[mid] > target to move the right boundary left.
3. Stops when left == right; this is the first index where value > target.
4. If target exists, left - 1 gives the index of its last occurrence.
5. Pairs naturally with the left-bound search to count duplicates or define intervals.

Example 1: Find Last Occurrence of Target

Goal: Return the last index where target appears in a sorted array with duplicates.
Approach: Use right-bound binary search and return left - 1 if found.

```
def last_occurrence(nums, target):
    left, right = 0, len(nums)
    while left < right:
        mid = (left + right) // 2
```

```
        if nums[mid] > target:
            right = mid
        else:
            left = mid + 1
    if left > 0 and nums[left - 1] == target:
        return left - 1
    return -1

# Example
print(last_occurrence([1,2,2,2,3,4], 2))  # Output: 3
```

Example 2: Find Right-most Insertion Point

Goal: Return the position where target should be inserted to maintain sort order, after all existing duplicates.
Approach: Same as the template; this is equivalent to bisect_right.

```
def insertion_index_right(nums, target):
    left, right = 0, len(nums)
    while left < right:
        mid = (left + right) // 2
        if nums[mid] > target:
            right = mid
        else:
            left = mid + 1
    return left

# Example
print(insertion_index_right([1,2,2,4], 2))  # Output: 3
```

Example 3: Count Occurrences Using Both Boundaries

Goal: Compute the number of times target appears in a sorted array.
Approach: Use the left-bound and right-bound binary searches together.

```
def count_occurrences(nums, target):
    def lower_bound(x):
        left, right = 0, len(nums)
        while left < right:
            mid = (left + right) // 2
            if nums[mid] >= x:
                right = mid
            else:
                left = mid + 1
        return left

    def upper_bound(x):
        left, right = 0, len(nums)
        while left < right:
            mid = (left + right) // 2
            if nums[mid] > x:
```

```
                right = mid
            else:
                left = mid + 1
        return left

    return upper_bound(target) - lower_bound(target)

# Example
print(count_occurrences([1,2,2,2,3,4], 2))  # Output: 3
```

Summary Table

ProblemVariantConditionResult Last occurrenceRight-boundarr[mid] > targetIndex of last match Insertion pointBoundaryarr[mid] > targetRightmost insertion index Count occurrencesCombined≥ and > searchesupper - lower