Binary Search: Duplicate Elements (Left-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 **leftmost position** where a target value can be inserted without violating sort order. It's particularly useful when the array may contain duplicates.

The search continues until left == right, and the loop invariant guarantees that left is the smallest index where arr[left] >= target.

Time Complexity: O(log n) **Space Complexity:** O(1).

Key Ideas

1. This is a **boundary-style** binary search (open interval on the right).
2. Use condition arr[mid] >= target to move right boundary leftward.
3. Loop ends when left == right; that index is the insertion point.
4. If target exists, this gives the index of its **first occurrence**.
5. If not found, it gives the correct place to insert to maintain order.

Example 1: Find First Occurrence of Target

Goal: Return the first index where target appears in a sorted array with duplicates.
Approach: Use left-bound binary search; check if the value at the result index equals target.

```
def first_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 < len(nums) and nums[left] == target:
        return left
    return -1

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

Example 2: Find Left-most Insertion Point

Goal: Return the position where target should be inserted to maintain sort order.
Approach: Same as the template; this is equivalent to bisect_left in Python.

```
def insertion_index(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([1,2,4,6], 5))  # Output: 3
```

Example 3: Count Occurrences of Target

Goal: Count how many times a target appears in a sorted array.
Approach: Find leftmost and rightmost positions using two binary searches.

```
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 First occurrenceLeft-boundarr[mid] >= targetIndex of first match Insertion pointBoundaryarr[mid] >= targetLeftmost insertion index Count occurrencesTwo searches≥ and > checksupper - lower