Binary Search — Coding Interview Notes (Light Theme)

General Pattern Template

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

    # left is the insertion point
    return left
```

Concept:
The **Binary Search** algorithm efficiently locates a target value in a sorted array by repeatedly dividing the search interval in half. It’s a fundamental divide-and-conquer technique used for search, optimization, and boundary detection problems.

Time Complexity: O(log n) **Space Complexity:** O(1) (iterative version).
When to use: Data is sorted or the search space can be ordered/monotonic.

Key Ideas

1. Maintain two pointers: left (low) and right (high).
2. Calculate mid and compare arr[mid] with target.
3. Shrink search space accordingly until left > right.
4. If not found, 'left' gives the correct insertion index.
5. Binary search can find conditions or transitions (first/last occurrence, boundary).

Example 1: Basic Binary Search

Goal: Return the index of the target if found, else -1.
Approach: Standard binary search on a sorted array.

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

# Example
print(binary_search([1,2,4,5,6,8], 5))  # Output: 3
```

Example 2: Find First Occurrence of Target

Goal: Find the first index where target appears (useful for duplicates).
Approach: Continue searching left after finding target.

```
def first_occurrence(nums, target):
    left, right = 0, len(nums) - 1
    res = -1
    while left <= right:
        mid = (left + right) // 2
        if nums[mid] == target:
            res = mid
            right = mid - 1  # move left for first occurrence
        elif nums[mid] < target:
            left = mid + 1
        else:
            right = mid - 1
    return res

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

Example 3: Search Insert Position

Goal: Find the index where the target should be inserted in sorted order.
Approach: When the loop ends, 'left' will be the insertion point.

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

```
# Example
print(search_insert_position([1,3,5,6], 2))  # Output: 1
```

Summary Table

ProblemVariantLogicComplexity Search elementStandardCompare mid with targetO(log n) First occurrenceLeft-biasedMove right = mid-1 when matchO(log n) Insert positionBoundary searchReturn left at endO(log n)