Binary Search Patterns – Complete Guide (with Python Templates)

This guide covers the 7 core Binary Search sub-patterns with usage cues, keywords, common LeetCode references, and clean Python templates. Use it as a quick-reference during interviews and practice.

1) Classic Index Search (Order-Agnostic)

When to Use: Direct search in a sorted array when order may be ascending or descending.

Keywords: sorted array, search element, ascending/descending, return index

LeetCode: 704, 33, 81

Template (Python):

```
def binary_search(arr, target):
    # Search for target in ascending or descending sorted array.
    # Returns index or -1 if not found.
    if not arr:
        return -1
    low, high = 0, len(arr) - 1
    is_asc = arr[0] <= arr[-1]
    while low <= high:
        mid = (low + high) // 2
        if arr[mid] == target:
            return mid
        if is_asc:
            if arr[mid] < target:
                low = mid + 1
            else:
                high = mid - 1
        else:
            if arr[mid] > target:
                low = mid + 1
            else:
                high = mid - 1
    return -1
```

2) Boundary Finding in Duplicates

When to Use: Find first/last index, counts, or insert position in a sorted array with duplicates.

Keywords: first index, last index, lower_bound, upper_bound, count, insert position

LeetCode: 34, 35

First Occurrence:

```
def first_occurrence(arr, target):
    # Returns index of first occurrence of target, or -1 if not found.
    low, high, ans = 0, len(arr) - 1, -1
    while low <= high:
        mid = (low + high) // 2
        if arr[mid] >= target:
            if arr[mid] == target:
                ans = mid
            high = mid - 1
        else:
            low = mid + 1
    return ans
```

Last Occurrence:
```
def last_occurrence(arr, target):
    # Returns index of last occurrence of target, or -1 if not found.
    low, high, ans = 0, len(arr) - 1, -1
    while low <= high:
        mid = (low + high) // 2
        if arr[mid] <= target:
            if arr[mid] == target:
                ans = mid
            low = mid + 1
        else:
            high = mid - 1
    return ans
```

Lower/Upper Bound + Count:
```
def lower_bound(arr, target):
    # First index i such that arr[i] >= target. Returns len(arr) if none.
    low, high = 0, len(arr)
    while low < high:
        mid = (low + high) // 2
        if arr[mid] < target:
            low = mid + 1
        else:
            high = mid
    return low

def upper_bound(arr, target):
    # First index i such that arr[i] > target. Returns len(arr) if none.
    low, high = 0, len(arr)
    while low < high:
        mid = (low + high) // 2
        if arr[mid] <= target:
            low = mid + 1
        else:
            high = mid
    return low

def count_occurrences(arr, target):
    i = lower_bound(arr, target)
    j = upper_bound(arr, target)
    return max(0, j - i)
```

3) Boolean Boundary (First True / Last False)

When to Use: Predicate flips once (False→True). Find threshold index.

Keywords: earliest day, minimum x, threshold, first time condition holds

LeetCode: 278 (+ feasibility flavors: 410, 1011, 875)

Template (Python):
```
def first_true(n, check):
    # Given a monotonic predicate check(x) over integers 0..n, find the smallest x where check(x)
    # Returns -1 if no True exists.
    low, high, ans = 0, n, -1
    while low <= high:
        mid = (low + high) // 2
        if check(mid):
            ans = mid
            high = mid - 1
        else:
            low = mid + 1
```

```
    return ans

def last_false(n, check):
    # Given monotonic check(x), find the largest x where check(x) is False.
    # Returns -1 if all are True.
    low, high, ans = 0, n, -1
    while low <= high:
        mid = (low + high) // 2
        if not check(mid):
            ans = mid
            low = mid + 1
        else:
            high = mid - 1
    return ans
```

4) Binary Search on Answer (Parametric Search)

When to Use: Search the answer space with a monotonic feasibility function.

Keywords: minimize maximum, maximize minimum, smallest possible X, capacity/time/threshold

LeetCode: 410, 1011, 1482, 1552, 2226, 875

Template (Python):

```
def min_capacity_to_ship(weights, days):
    # Minimize maximum load per day so all packages ship within 'days'.
    # Feasibility check is monotonic over capacity.
    low, high = max(weights), sum(weights)

    def can_ship(cap):
        curr, cnt = 0, 1
        for w in weights:
            if w > cap:
                return False
            if curr + w > cap:
                cnt += 1
                curr = 0
            curr += w
        return cnt <= days

    while low <= high:
        mid = (low + high) // 2
        if can_ship(mid):
            high = mid - 1
        else:
            low = mid + 1
    return low  # smallest capacity that works
```

5) Rotated / Bitonic Arrays

When to Use: Find pivot/peak and then binary search in the correct side.

Keywords: rotated, pivot, mountain array, peak element, bitonic

LeetCode: 33, 153, 162, 852

Find Pivot (min index):

```
def find_pivot_min_index(arr):
    # Index of the smallest element in a rotated sorted array (with distinct values).
    low, high = 0, len(arr) - 1
    while low < high:
        mid = (low + high) // 2
```

```
        if arr[mid] > arr[high]:
            low = mid + 1
        else:
            high = mid
    return low
```

Search in Rotated Array:

```
def search_rotated(arr, target):
    # Search target in a rotated sorted array (distinct values).
    if not arr:
        return -1
    pivot = find_pivot_min_index(arr)
    n = len(arr)

    def bs(l, r):
        while l <= r:
            m = (l + r) // 2
            if arr[m] == target:
                return m
            if arr[m] < target:
                l = m + 1
            else:
                r = m - 1
        return -1

    if arr[pivot] <= target <= arr[-1]:
        return bs(pivot, n - 1)
    else:
        return bs(0, pivot - 1)
```

Peak in Mountain Array:

```
def find_peak_mountain(arr):
    # Peak index in a mountain (bitonic) array: strictly increasing then decreasing.
    low, high = 0, len(arr) - 1
    while low < high:
        mid = (low + high) // 2
        if arr[mid] < arr[mid + 1]:
            low = mid + 1
        else:
            high = mid
    return low  # peak index
```

6) 2D Matrix Search (Virtual Indexing)

When to Use: Matrix is globally sorted when flattened row-wise; binary search on index.

Keywords: sorted matrix, row-wise sorted, virtual indexing

LeetCode: 74 (240 is a related elimination approach)

Template (Python):

```
def search_matrix(matrix, target):
    # Search in a row-wise sorted matrix where the first element of each row is greater than
    # the last element of the previous row. Treat as flattened sorted array.
    if not matrix or not matrix[0]:
        return False
    rows, cols = len(matrix), len(matrix[0])
    low, high = 0, rows * cols - 1
    while low <= high:
        mid = (low + high) // 2
        val = matrix[mid // cols][mid % cols]
        if val == target:
```

```
            return True
        elif val < target:
            low = mid + 1
        else:
            high = mid - 1
    return False
```

7) Binary Search on Real Numbers (Floating Point)

When to Use: Continuous domains requiring a tolerance; search with eps precision.

Keywords: precision, epsilon, continuous, square root, cut cables/wood to max length

LeetCode: 69, 644, (many geometry/optimization problems)

Square Root via BS:

```
def sqrt_binary_search(x, eps=1e-6):
    # Compute square root of x using binary search on reals with precision eps.
    if x < 0:
        raise ValueError("x must be non-negative")
    low, high = (0.0, max(1.0, float(x)))
    while high - low > eps:
        mid = (low + high) / 2.0
        if mid * mid < x:
            low = mid
        else:
            high = mid
    return (low + high) / 2.0
```

Generic Real-Valued BS:

```
def binary_search_real(lo, hi, check, eps=1e-6):
    # Find minimal value x in [lo, hi] such that monotonic check(x) is True, with precision eps.
    # Assumes check is False for small x and True after some threshold.
    while hi - lo > eps:
        mid = (lo + hi) / 2.0
        if check(mid):
            hi = mid
        else:
            lo = mid
    return hi
```

Master Keyword Cheat Sheet

Keyword / Phrase	Pattern
sorted array	Classic Search
first occurrence, last occurrence	Boundary in Duplicates
earliest day, minimum x	Boolean Boundary
minimize maximum, maximize minimum	Binary Search on Answer
rotated, mountain, peak	Rotated / Bitonic Array
sorted matrix	2D Matrix Search
precision, epsilon, continuous	Real Number BS