Cheat Sheet: Binary Search Series (Java - DSA)

1. Lower Bound

Definition:

The **lower bound** of a target value in a sorted array is the index of the **first element that is not less than the target** (i.e., greater than or equal to target).

Use Case:

- Finding the starting position of a value or the first valid position to insert it.

The binary search continues narrowing the search space while maintaining a possible candidate for the answer. We lean left whenever we find a potential match.

Java Code:

```
public int lowerBound(int[] nums, int target) {
    int low = 0, high = nums.length;
    while (low < high) {
        int mid = low + (high - low) / 2;
        if (nums[mid] < target) {
            low = mid + 1;
        } else {
            high = mid;
        }
    }
    return low;
}
```

2. Upper Bound

Definition:

The **upper bound** is the index of the **first element that is greater than the target**.

Use Case:

- Useful in frequency counting or range-based queries.

Same as lower bound but condition checks for strictly greater values.

Java Code:

```
public int upperBound(int[] nums, int target) {
```

```
    int low = 0, high = nums.length;
    while (low < high) {
        int mid = low + (high - low) / 2;
        if (nums[mid] <= target) {
            low = mid + 1;
        } else {
            high = mid;
        }
    }
    return low;
}
```

3. First Occurrence

Definition:

Find the **first index** where a given element occurs in a sorted array (with duplicates).

Use Case:

- Used in frequency counting and range checks.

Perform standard binary search but move left if element is found, until the first instance is identified.

Java Code:

```
public int firstOccurrence(int[] nums, int target) {
    int low = 0, high = nums.length - 1, result = -1;
    while (low <= high) {
        int mid = low + (high - low) / 2;
        if (nums[mid] == target) {
            result = mid;
            high = mid - 1; // move left
        } else if (nums[mid] < target) {
            low = mid + 1;
        } else {
            high = mid - 1;
        }
    }
    return result;
}
```

4. Last Occurrence

Definition:

Find the **last index** where a given element occurs in a sorted array (with duplicates).

Use Case:

- Often paired with first occurrence for range calculations.

Binary search pattern but move right when a match is found.

Java Code:

```
public int lastOccurrence(int[] nums, int target) {
   int low = 0, high = nums.length - 1, result = -1;
   while (low <= high) {
      int mid = low + (high - low) / 2;
      if (nums[mid] == target) {
         result = mid;
         low = mid + 1; // move right
      } else if (nums[mid] < target) {
         low = mid + 1;
      } else {
         high = mid - 1;
      }
   }
   return result;
}
```

5. Search in Rotated Sorted Array
Definition:

Given a rotated sorted array, search for a specific element and return its index. The array was originally sorted but then rotated.

Use Case:

- Common interview question; demonstrates binary search flexibility.

Split the array into two halves. One half will always be sorted. Based on the target's position relative to the sorted half, adjust search bounds.

Java Code:

```
public int searchRotated(int[] nums, int target) {
   int low = 0, high = nums.length - 1;

   while (low <= high) {
```

```
        int mid = low + (high - low) / 2;
        if (nums[mid] == target) return mid;

        // Left half is sorted
        if (nums[low] <= nums[mid]) {
            if (nums[low] <= target && target < nums[mid]) {
                high = mid - 1;
            } else {
                low = mid + 1;
            }
        }
        // Right half is sorted
        else {
            if (nums[mid] < target && target <= nums[high]) {
                low = mid + 1;
            } else {
                high = mid - 1;
            }
        }
    }
    return -1;
}
```

Summary Table

Topic	Purpose	Time Complexity
Lower Bound	First index ≥ target	O(log n)
Upper Bound	First index > target	O(log n)
First Occurrence	First index of target	O(log n)
Last Occurrence	Last index of target	O(log n)
Search Rotated Sorted	Index of target in rotated sorted array	O(log n)

Practice: LeetCode Problem - Search Insert Position

Practice: GeeksforGeeks - Upper Bound

Practice: LeetCode - First and Last Position of Element

Practice: GeeksforGeeks - Last Occurrence

Practice: LeetCode - Search in Rotated Sorted Array