

■ First and Last Position of Element

Find both boundaries of a repeated element in a sorted array

● Binary Search

■ MEDIUM

■ What You Will Learn

- How to find the FIRST occurrence of a number
- How to find the LAST occurrence of a number
- The 'biased binary search' technique — lean left or lean right
- Why you need TWO separate binary searches

■ Intuition First

Imagine searching for the word 'cat' in a dictionary. It appears on pages 120, 121, 122. You want page 120 (first) and page 122 (last). Normal binary search lands somewhere in the middle. The trick: run binary search TWICE — once biased toward left (first), once biased toward right (last).

■ Key Insight

When `arr[mid] == target`, instead of returning immediately, keep searching! For first occurrence: go left (`right=mid-1`). For last occurrence: go right (`left=mid+1`). Record the answer each time you find the target.

■ Visual Explanation

Array: [5, 7, 7, 8, 8, 8, 10] Target = 8

Finding FIRST occurrence (lean left):

`L=0, R=6 → mid=3 → arr[3]=8 == 8 → ans=3, R=2`

`L=0, R=2 → mid=1 → arr[1]=7 < 8 → L=2`

`L=2, R=2 → mid=2 → arr[2]=7 < 8 → L=3`

`L=3 > R=2 → stop → first = 3 ✓`

Finding LAST occurrence (lean right):

`L=0, R=6 → mid=3 → arr[3]=8 == 8 → ans=3, L=4`

`L=4, R=6 → mid=5 → arr[5]=8 == 8 → ans=5, L=6`

`L=6, R=6 → mid=6 → arr[6]=10 > 8 → R=5`

`L=6 > R=5 → stop → last = 5 ✓`

Answer: [3, 5]

■ Java Code

```

1  class Solution {
2  public int[] searchRange(int[] nums, int target) {
3      return new int[]{findFirst(nums, target), findLast(nums, target)};
4  }
5
6  private int findFirst(int[] nums, int target) {
7      int left = 0, right = nums.length - 1, ans = -1;
8      while (left <= right) {
9          int mid = left + (right - left) / 2;
10         if (nums[mid] == target) {
11             ans = mid; // Record, but keep going LEFT
12             right = mid - 1;
13         } else if (nums[mid] < target) {
14             left = mid + 1;
15         } else {
16             right = mid - 1;
17         }
18     }
19     return ans;
20 }
21
22 private int findLast(int[] nums, int target) {
23     int left = 0, right = nums.length - 1, ans = -1;
24     while (left <= right) {
25         int mid = left + (right - left) / 2;
26         if (nums[mid] == target) {
27             ans = mid; // Record, but keep going RIGHT
28             left = mid + 1;
29         } else if (nums[mid] < target) {
30             left = mid + 1;
31         } else {
32             right = mid - 1;
33         }
34     }
35     return ans;
36 }
37 }

```

■ ■ Complexity

Complexity	Value	What it means
Time	$O(\log n)$	Two binary searches, each $O(\log n)$. Total still $O(\log n)$.
Space	$O(1)$	Only a few variables used

Common Mistakes

Mistake 1

Returning immediately when `arr[mid]==target`. You MUST continue searching to find the true first/last boundary.

Mistake 2

Forgetting to initialize `ans=-1`. If target doesn't exist, you need to return `[-1,-1]`.

Quick Revision

✓	Run binary search twice — once for first, once for last
✓	Finding first: when match found, save ans and go left (<code>right=mid-1</code>)
✓	Finding last: when match found, save ans and go right (<code>left=mid+1</code>)
✓	Initialize <code>ans=-1</code> to handle 'not found' case
✓	Time $O(\log n)$, Space $O(1)$

Visual Thinking

Think of the matching elements as a painted stripe across the array. The first search finds the left edge of the stripe, the second finds the right edge. Both searches 'push' past the stripe in opposite directions.