# **Leetcode 34 — Find First and Last Position of Element in Sorted Array**

## Problem Understanding

You're given a **sorted integer array nums** and a target value target.  
Return the **starting and ending position** of the given target in the array.  
If not found, return [-1, -1].

Example:

Input: nums = [5,7,7,8,8,10], target = 8

Output: [3,4]

## Optimized Java Solution (Binary Search)

We do **2 binary searches**:

1. One for the **leftmost index**
2. One for the **rightmost index**

class Solution {

public int[] searchRange(int[] nums, int target) {

int first = findBound(nums, target, true);

int last = findBound(nums, target, false);

return new int[]{first, last};

}

private int findBound(int[] nums, int target, boolean isFirst) {

int left = 0, right = nums.length - 1;

int bound = -1;

while (left <= right) {

int mid = left + (right - left) / 2;

if (nums[mid] == target) {

bound = mid;

if (isFirst)

right = mid - 1; // search left half

else

left = mid + 1; // search right half

} else if (nums[mid] < target) {

left = mid + 1;

} else {

right = mid - 1;

}

}

return bound;

}

}

## Dry Run Using Table

For:  
nums = [5, 7, 7, 8, 8, 10], target = 8

#### First Occurrence (isFirst = true)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| left | right | mid | nums[mid] | action | bound |
| 0 | 5 | 2 | 7 | left = mid+1 | -1 |
| 3 | 5 | 4 | 8 | bound = 4, right = 3 | 4 |
| 3 | 3 | 3 | 8 | bound = 3, right = 2 | 3 |

🟢 First index = 3

#### Last Occurrence (isFirst = false)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| left | right | mid | nums[mid] | action | bound |
| 0 | 5 | 2 | 7 | left = mid+1 | -1 |
| 3 | 5 | 4 | 8 | bound = 4, left = 5 | 4 |
| 5 | 5 | 5 | 10 | right = 4 | 4 |

🟢 Last index = 4

✅ Final output = [3, 4]

## Time / Space Complexity

* **Time:** O(log n)  
  → Two binary searches
* **Space:** O(1)

## Alternate Approaches

1. **Linear Scan (Brute Force):**
   * Scan left to right and store indices
   * Time: O(n)
2. **HashMap (if not sorted):**
   * But this problem depends on sorted array → so not applicable
3. **Binary Search for First and Last using separate functions (as done above)**
   * Most optimal