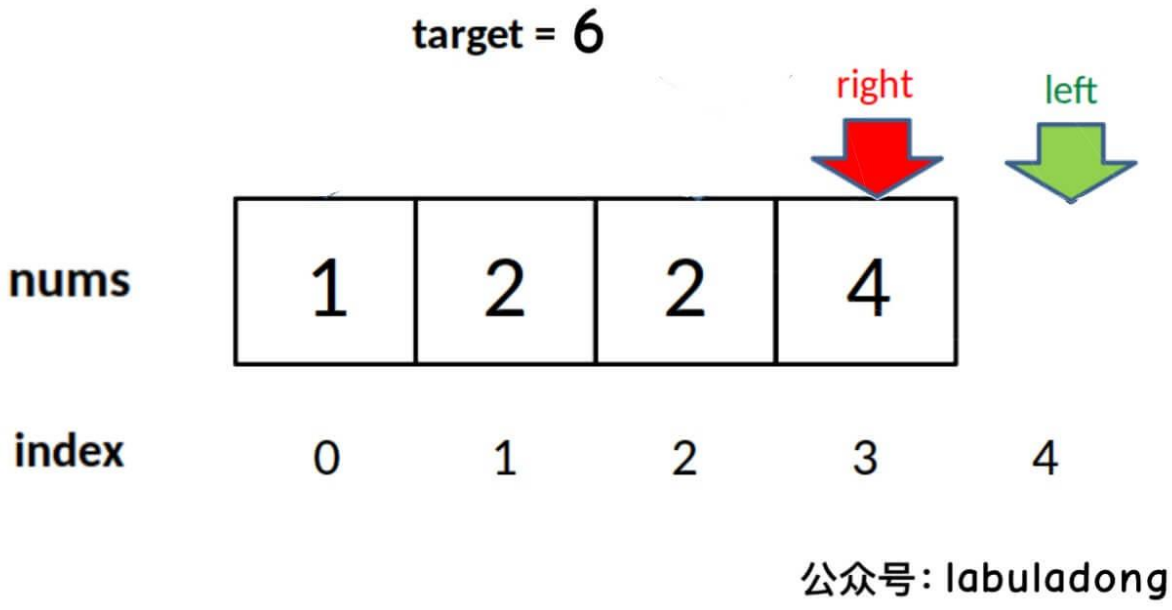
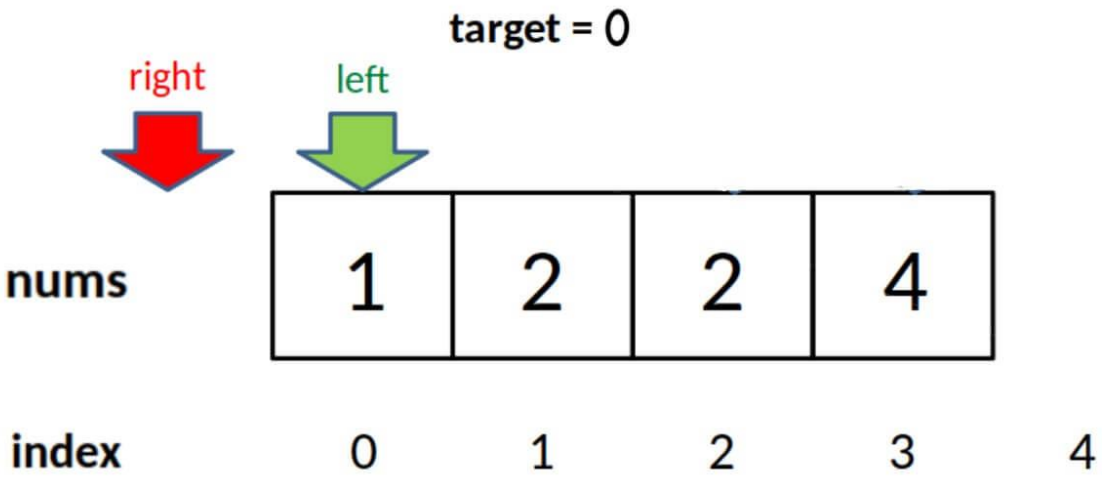
# 34. Find First and Last Position of Element in Sorted Array (M)

## 基本思路

PS:这道题在《算法小抄》的第 71 页。
二分搜索的难点就在于如何搜索左侧边界和右侧边界，代码的边界的控制非常考验你的微操，这也是很多人知道二分搜索原理但是很难写对代码的原因。
二分搜索框架详解 专门花了很大篇幅讨论如何写对二分搜索算法，总结来说：
写对二分搜索的关键在于搞清楚搜索边界，到底是开区间还是闭区间？到底应该往左侧收敛还是应该往右侧收敛？
深入的探讨请看详细题解。
详细题解 :我作了首诗，保你闭着眼睛也能写对二分查找
标签 :二分搜索

## 解法代码






公众号：labuladong

```
class Solution {
    public int[] searchRange(int[] nums, int target) {
        return new int[]{left_bound(nums, target), right_bound(nums,
target)};
    }

    int left_bound(int[] nums, int target) {
        int left = 0, right = nums.length - 1;
        // 搜索区间为 [left, right]
        while (left <= right) {
            int mid = left + (right - left) / 2;
            if (nums[mid] < target) {
                // 搜索区间变为 [mid+1, right]
                left = mid + 1;
            } else if (nums[mid] > target) {
                // 搜索区间变为 [left, mid-1]
                right = mid - 1;
            } else if (nums[mid] == target) {
                // 收缩右侧边界
                right = mid - 1;
            }
        }
        // 检查出界情况
        if (left >= nums.length || nums[left] != target) {
```

```
            return -1;
        }
        return left;
    }

    int right_bound(int[] nums, int target) {
        int left = 0, right = nums.length - 1;
        while (left <= right) {
            int mid = left + (right - left) / 2;
            if (nums[mid] < target) {
                left = mid + 1;
            } else if (nums[mid] > target) {
                right = mid - 1;
            } else if (nums[mid] == target) {
                // 这里改成收缩左侧边界即可
                left = mid + 1;
            }
        }
        // 这里改为检查 right 越界的情况，见下图
        if (right < 0 || nums[right] != target) {
            return -1;
        }
        return right;
    }
}
```

类似题目：

- 704. 二分查找（简单）