

34. Find First and Last Position of Element in Sorted Array

Medium

4754

183

Add to List

Share

Given an array of integers `nums` sorted in ascending order, find the starting and ending position of a given `target` value.

If `target` is not found in the array, return `[-1, -1]`.

Follow up: Could you write an algorithm with $O(\log n)$ runtime complexity?

Example 1:

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

Output: `[3,4]`

Example 2:

Input: `nums = [5,7,7,8,8,10]`, `target = 6`

Output: `[-1,-1]`

Example 3:

Input: `nums = []`, `target = 0`

Output: `[-1,-1]`

Constraints:

- $0 \leq \text{nums.length} \leq 10^5$
- $-10^9 \leq \text{nums}[i] \leq 10^9$
- `nums` is a non-decreasing array.
- $-10^9 \leq \text{target} \leq 10^9$

```
int left_bound(int[] nums, int target) {
    if (nums.length == 0) return -1;
    int left = 0;
    int right = nums.length;
    while (left < right) {
        int mid = (left + right) / 2;
        if (nums[mid] == target) {
            right = mid;
        } else if (nums[mid] < target) {
            left = mid + 1;
        } else if (nums[mid] > target) {
            right = mid;
        }
    }
    if (left == nums.length) return -1;
    return nums[left] == target ? left : -1;
}
```

Handwritten notes:
- `right` is `nums.length` (not `nums.length - 1`)
- `left` and `right` are pointers to the search range
- `mid` is calculated as $(\text{left} + \text{right}) / 2$
- `left = mid + 1` when `nums[mid] < target`
- `right = mid` when `nums[mid] > target` or `nums[mid] == target`
- `left` is in the range $[0, \text{nums.length}]$
- `left == nums.length` means the target is not found
- `nums[left] == target` checks if the element at the found position is the target

```
int right_bound(int[] nums, int target) {
    if (nums.length == 0) return -1;
    int left = 0;
    int right = nums.length;
    while (left < right) {
        int mid = (left + right) / 2;
        if (nums[mid] == target) {
            left = mid + 1;
        } else if (nums[mid] < target) {
            left = mid + 1;
        } else if (nums[mid] > target) {
            right = mid;
        }
    }
    if (left == 0) return -1;
    return nums[left - 1] == target ? (left - 1) : -1;
}
```

Handwritten notes:
- `left` is in the range $[0, \text{nums.length}]$
- `left == 0` means the target is not found
- `nums[left - 1] == target` checks if the element at the found position is the target

```

1  class Solution {
2  public:
3      vector<int> searchRange(vector<int>& nums, int target) {
4
5          vector<int> res;
6
7          if (nums.size() == 0) {
8              res.push_back(-1);
9              res.push_back(-1);
10             return res;
11         }
12
13         res.push_back(low_bound(nums, target));
14         res.push_back(high_bound(nums, target));
15
16         return res;
17     }
18
19 private:
20     int low_bound(vector<int> nums, int target) {
21         int left = 0;
22         int right = nums.size();
23
24         while (left < right) {
25             int mid = left + (right - left) / 2;
26
27             if (nums[mid] < target) {
28                 left = mid + 1;
29             } else if (nums[mid] > target) {
30                 right = mid;
31             } else if (nums[mid] == target) {
32                 right = mid;
33             }
34         }
35
36         if (left == nums.size() || nums[left] != target) {
37             return -1;
38         }
39
40         return left;
41     }
42
43     int high_bound(vector<int> nums, int target) {
44         int left = 0;
45         int right = nums.size();
46
47         while (left < right) {
48             int mid = left + (right - left) / 2;
49
50             if (nums[mid] < target) {
51                 left = mid + 1;
52             } else if (nums[mid] > target) {
53                 right = mid;
54             } else if (nums[mid] == target) {
55                 left = mid + 1;
56             }
57         }
58
59         left--;
60
61         if (left < 0 || nums[left] != target) {
62             return -1;
63         }
64
65         return left;
66     }
67 }
68 };

```