Previous Next

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

Nov. 16, 2017 | 217.5K views

★★★★★
Average Rating: 3.42 (94 votes)

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

Your algorithm's runtime complexity must be in the order of *O*(log *n*).

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

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]
```

Constraints:

- `0 <= nums.length <= 10^5`
- `-10^9 <= nums[i] <= 10^9`
- `nums` is a non decreasing array.
- `-10^9 <= target <= 10^9`

Approach 1: Linear Scan

Intuition

Checking every index for `target` exhausts the search space, so it must work.

Algorithm

First, we do a linear scan of `nums` from the left, `break`ing when we find an instance of `target`. If we never `break`, then `target` is not present, so we can return the "error code" of `[-1, -1]` early. Given that we did find a valid left index, we can do a second linear scan, but this time from the right. In this case, the first instance of `target` encountered will be the rightmost one (and because a leftmost one exists, there is guaranteed to also be a rightmost one). We then simply return a list containing the two located indices.

Java Python3 Copy

```python
class Solution:
    def searchRange(self, nums, target):
        # find the index of the leftmost appearance of `target`. if it does not
        # appear, return [-1, -1] early.
        for i in range(len(nums)):
            if nums[i] == target:
                left_idx = i
                break
        else:
            return [-1, -1]

        # find the index of the rightmost appearance of `target` (by reverse
        # iteration). it is guaranteed to appear.
        for j in range(len(nums)-1, -1, -1):
            if nums[j] == target:
                right_idx = j
                break

        return [left_idx, right_idx]
```

Complexity Analysis

- Time complexity : $O(n)$

  This brute-force approach examines each of the `n` elements of `nums` exactly twice, so the overall runtime is linear.

- Space complexity : $O(1)$

  The linear scan method allocates a fixed-size array and a few integers, so it has a constant-size memory footprint.

Approach 2: Binary Search

Intuition

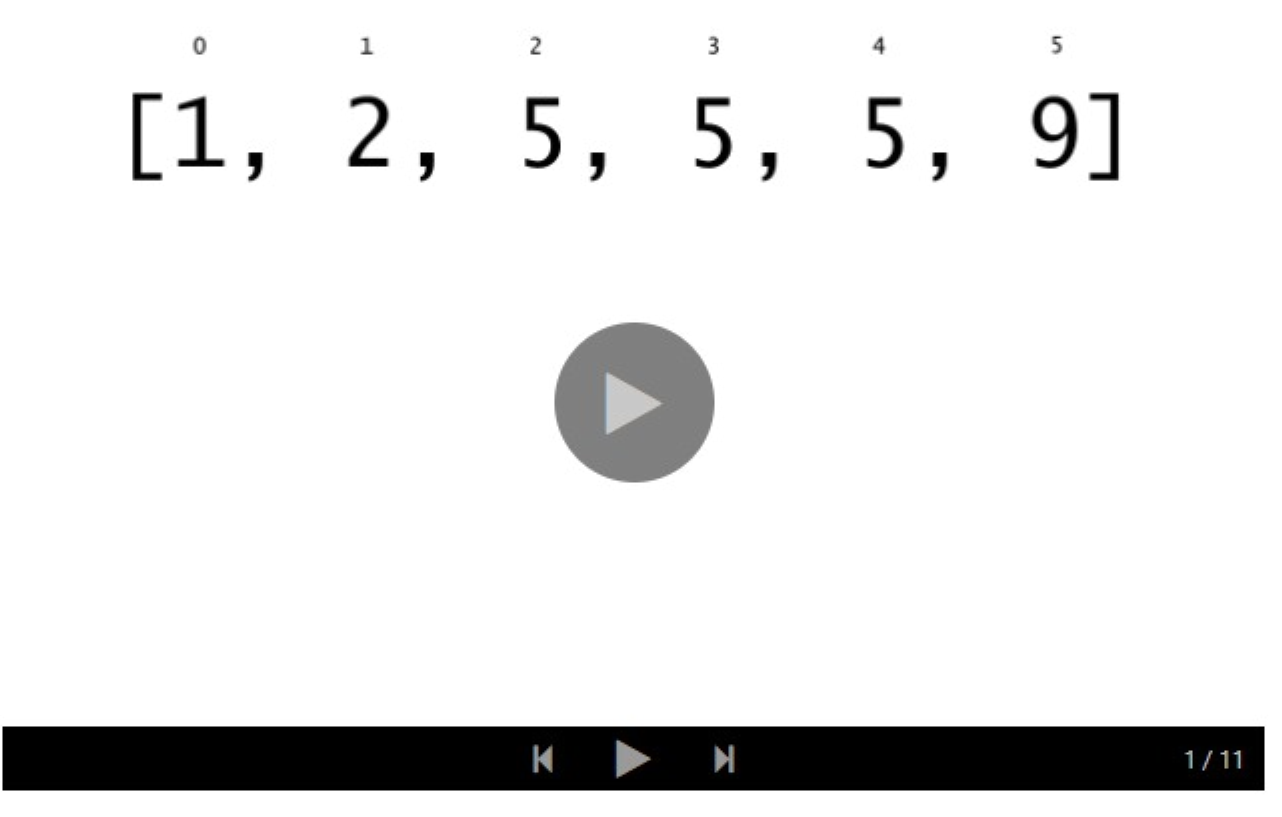
Because the array is sorted, we can use binary search to locate the left and rightmost indices.

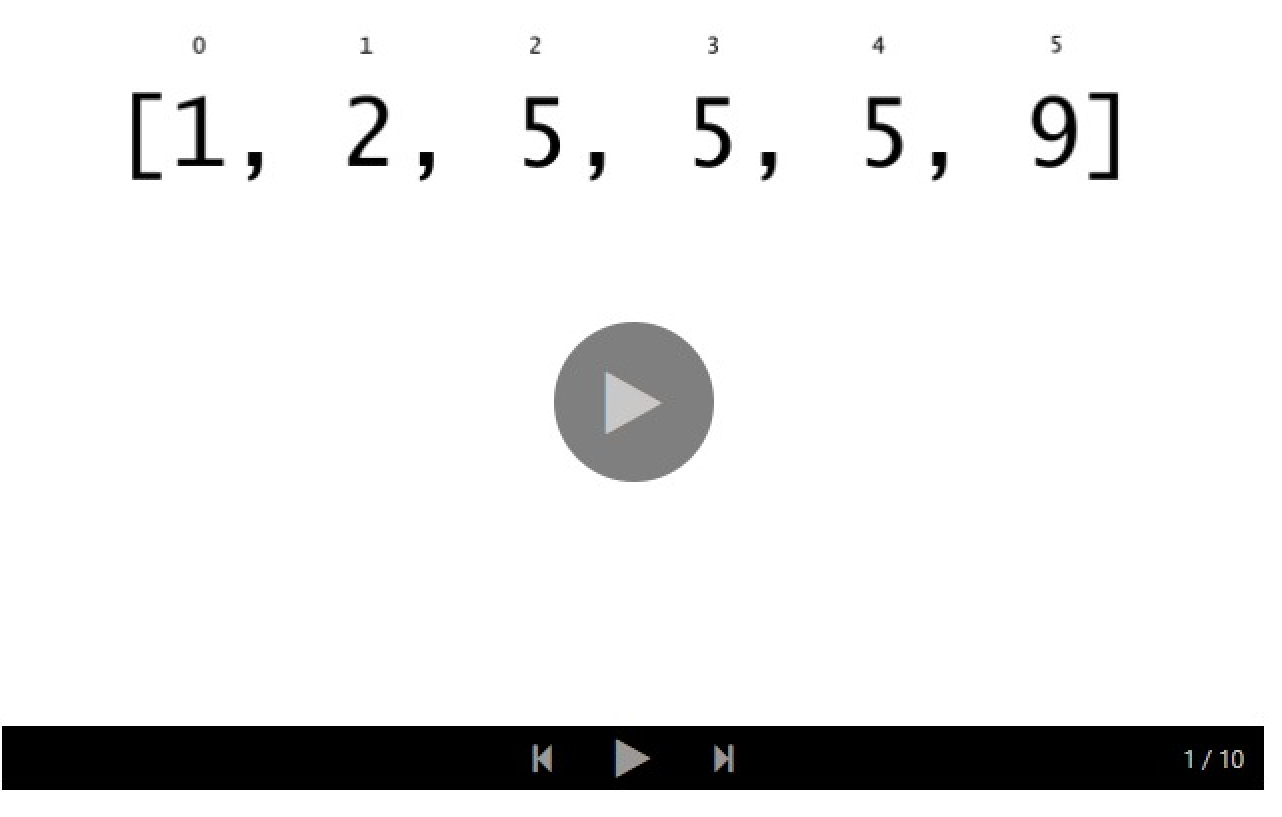
Algorithm

The overall algorithm works fairly similarly to the linear scan approach, except for the subroutine used to find the left and rightmost indices themselves. Here, we use a modified binary search to search a sorted array, with a few minor adjustments. First, because we are locating the leftmost (or rightmost) index containing `target` (rather than returning `true` iff we find `target`), the algorithm does not terminate as soon as we find a match. Instead, we continue to search until `lo == hi` and they contain some index at which `target` can be found.

The other change is the introduction of the `left` parameter, which is a boolean indicating what to do in the event that `target == nums[mid]`; if `left` is `true`, then we "recurse" on the left subarray on ties. Otherwise, we go right. To see why this is correct, consider the situation where we find `target` at index `i`. The leftmost `target` cannot occur at any index greater than `i`, so we never need to consider the right subarray. The same argument applies to the rightmost index.

The first animation below shows the process for finding the leftmost index, and the second shows the process for finding the index right of the rightmost index.





Java Python3 Copy

```python
class Solution:
    # returns leftmost (or rightmost) index at which `target` should be inserted in sorted
    # array `nums` via binary search.
    def extreme_insertion_index(self, nums, target, left):
        lo = 0
        hi = len(nums)

        while lo < hi:
            mid = (lo + hi) // 2
            if nums[mid] > target or (left and target == nums[mid]):
                hi = mid
            else:
                lo = mid+1

        return lo


    def searchRange(self, nums, target):
        left_idx = self.extreme_insertion_index(nums, target, True)

        # assert that `left_idx` is within the array bounds and that `target`
        # is actually in `nums`.
        if left_idx == len(nums) or nums[left_idx] != target:
            return [-1, -1]

        return [left_idx, self.extreme_insertion_index(nums, target, False)-1]
```

Complexity Analysis

- Time complexity : $O(\log_{10}(n))$

  Because binary search cuts the search space roughly in half on each iteration, there can be at most $\lceil \log_{10}(n) \rceil$ iterations. Binary search is invoked twice, so the overall complexity is logarithmic.

- Space complexity : $O(1)$

  All work is done in place, so the overall memory usage is constant.

Rate this article: ★★★★★

Previous Next

Comments: 93

Sort By

Type comment here... (Markdown is supported)

Preview Post

bephrem ★ 3372 January 14, 2019 12:52 AM

Doing `int mid = (lo + hi) / 2;` is prone to overflow. Instead `int mid = lo + (hi - lo) / 2`.

285 Share Reply

SHOW 11 REPLIES

niketanrane ★ 213 November 29, 2018 4:33 PM

Time complexity should be Log2n instead of log10n. Since we are dividing each time into half, this should be log with base 2.

211 Share Reply

SHOW 7 REPLIES

abhi1287 ★ 48 June 16, 2018 8:18 PM

Shouldn't it be log (base 2) (n) ?

48 Share Reply

SHOW 2 REPLIES

ygongdev ★ 57 August 13, 2018 10:54 PM

I think a more straightforward alternative could be just to perform 3 binary searches. First search to find any possible position of the target and a left and right binary search to find the leftmost and rightmost position of the target. Here's a video explanation: https://www.youtube.com/watch?v=kE6DBnYTrlU&t=9s

Read More

41 Share Reply

SHOW 9 REPLIES

jeffvwei ★ 69 January 6, 2019 6:33 AM

"your algorthim must have a runtime of log(n)"

First solution is linear.

k

69 Share Reply

SHOW 4 REPLIES

prince2 ★ 53 July 9, 2018 11:36 PM

Other alternative is to do regular binary search to find target. Once you find target walk to the left until its different than target. And do the same on right side. Time complexity is O(log(n) + k) where k is the number of occurance of target

30 Share Reply

SHOW 7 REPLIES

AmitDharmadhikari ★ 12 January 18, 2019 1:36 AM

Why log to the base 10 and not log to the base 2?

12 Share Reply

SHOW 3 REPLIES

BarryF ★ 12 March 11, 2018 4:29 AM

Why is Linear Scan accepted if it has a runtime complexity of O(n)?

12 Share Reply

SHOW 1 REPLY

Dr_Sean ★ 541 January 2, 2019 11:35 AM

My Python code:
class Solution:

```
def searchRange(self, nums, target):
    if len(nums) == 0: return [-1, -1]
```

Read More

11 Share Reply

SHOW 4 REPLIES

edwardsun007 ★ 15 April 19, 2018 12:24 PM

The 1st approach should not be there at all ..... question clearly asks for O(log n) time, your interview fails at the moment you try to do this in linear !

2nd Approach is not straightforward enough for me.

13 Share Reply

SHOW 2 REPLIES

‹ 1 2 3 4 5 6 ... 9 10 ›