search_insert_position

May 10, 2025

1 Search Insert Position

1.1 Problem Definition

Given a sorted array of distinct integers and a target value, return the index if the target is found. If not, return the index where it would be if it were inserted in order.

You must write an algorithm with $O(logn)$ runtime complexity.

1.1.1 Example Cases

Example 1:

Input: nums = [1,3,5,6], target = 5
Output: 2

Example 2:

Input: nums = [1,3,5,6], target = 2
Output: 1

Example 3:

Input: nums = [1,3,5,6], target = 7
Output: 4

1.1.2 Constraints

Constraints:

- 1 <= nums.length <= 10^4
- -10^4 <= nums[i] <= 10^4
- nums contains distinct values sorted in ascending order.
- -10^4 <= target <= 10^4

1.2 Example test cases

[70]:
```
def test_cases():
    assert searchInsert([1,3,5,6], 5) == 2
    assert searchInsert([1,3,5,6], 2) == 1
    assert searchInsert([1,3,5,6], 7) == 4
    print("All test cases passed!")
```

1.3 Solutions

$O(n)$

We can just iterate over all elements and check if the number if higher or equal than the target, we should return the index of that element.
If none of the elements is higher than the target, then we should return the index + 1 (insert from the end).

[71]:
```
def searchInsert(nums, target):
    for index, num in enumerate(nums):
        if num >= target:
            return index
    return index + 1
```

[72]:
```
test_cases()
```

```
All test cases passed!
```

1.3.1 Improvements

$O(logn)$

That's a binary search problem that should be optimized a bit for the following use case.

The function stops the same way as the binary search (when left_index is higher than the right_end), but when it exits without finding the target, it returns left as the correct insert spot.

That's because when the while loop stops: - All numbers to the left of left_end are smaller than the target. - All numbers to the right of left_end are bigger.

So, left_end points to the exact position where the target should go, whether it's between numbers or at the start or end of the list.

[75]:
```
def searchInsert(nums, target):
    left_end = 0
    right_end = len(nums) - 1

    while left_end <= right_end:
        center = (left_end + right_end) // 2

        if nums[center] == target:
            return center

        elif nums[center] > target:
            right_end = center - 1
        else:
            left_end = center + 1

    return left_end
```

[76]:
```
test_cases()
```

```
All test cases passed!
```