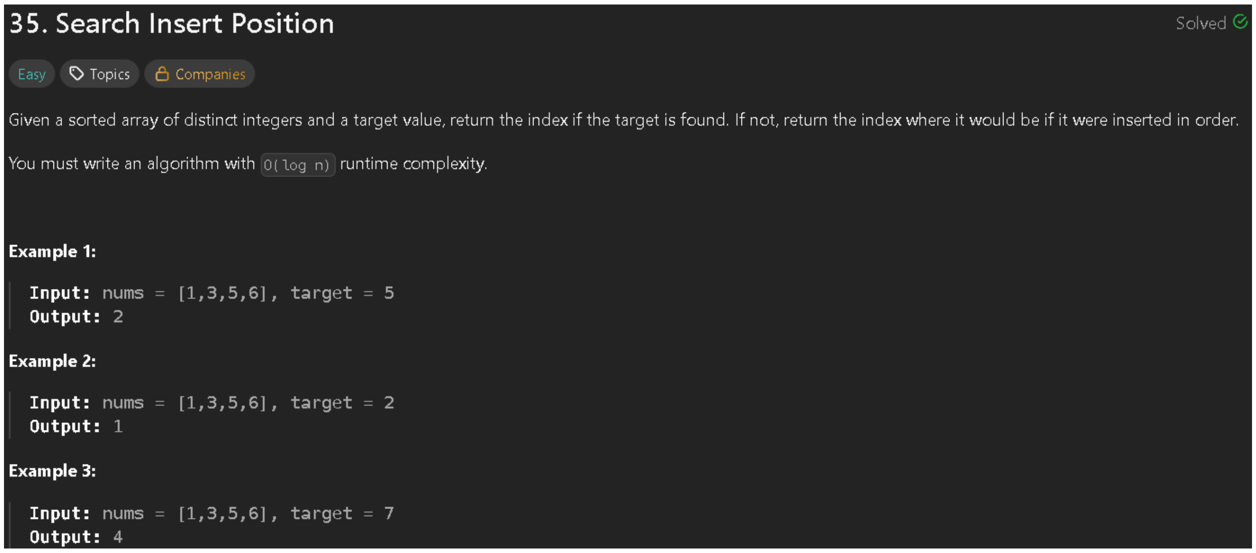


35. Search Insert Position

Solved

Easy · Topics · Companies

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(log n)` runtime complexity.

Example 1:

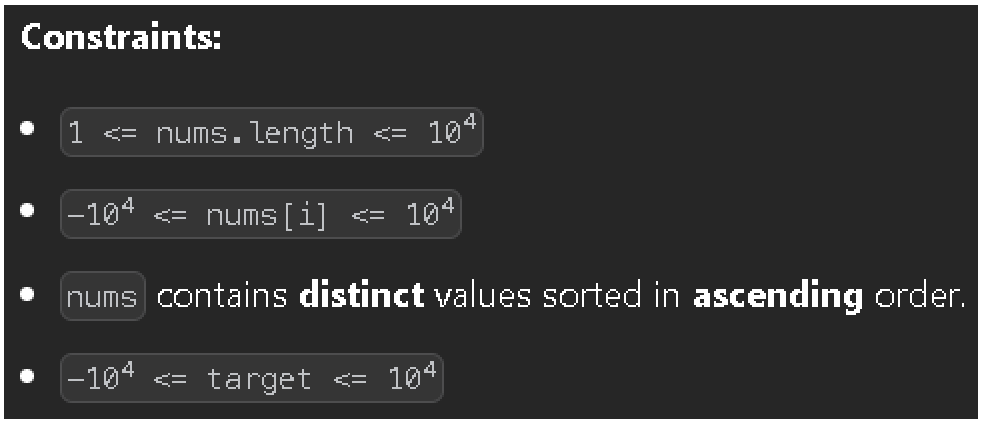
```
Input: nums = [1,3,5,6], target = 5
Output: 2
```

Example 2:

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

Example 3:

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



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

Python:

```
class Solution:
    def searchInsert(self, nums: List[int], target: int) -> int:
        left, right = 0, len(nums) - 1

        while left <= right:
            mid = (left + right) // 2

            if nums[mid] == target:
                return mid  # Target found at mid
            elif nums[mid] < target:
                left = mid + 1  # Search right half
            else:
                right = mid - 1  # Search left half

        # If not found, left will be the correct insert position
        return left
```

JavaScript:

```
/**
 * @param {number[]} nums
 * @param {number} target
 * @return {number}
 */
var searchInsert = function(nums, target) {
    let left = 0;
    let right = nums.length - 1;

    while (left <= right) {
        let mid = Math.floor((left + right) / 2);

        if (nums[mid] === target) {
            return mid; // target found
        } else if (nums[mid] < target) {
            left = mid + 1; // search in right half
        } else {
            right = mid - 1; // search in left half
        }
    }

    // If target not found, left will be the insert position
    return left;
};
```

Java:

```
class Solution {
    public int searchInsert(int[] nums, int target) {
        int left = 0, right = nums.length - 1;

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

            if (nums[mid] == target) {
                return mid; // Target found
            } else if (nums[mid] < target) {
                left = mid + 1; // Move right
            } else {
                right = mid - 1; // Move left
            }
        }

        // If not found, left will be the insertion index
        return left;
    }
}
```