Problem 2774: Array Upper Bound

Problem Information

Difficulty: Easy
Acceptance Rate: 0.00%
Paid Only: No

Problem Description

Write code that enhances all arrays such that you can call the

upperBound()

method on any array and it will return the last index of a given

target

number.

nums

is a sorted ascending array of numbers that may contain duplicates. If the

target

number is not found in the array, return

-1

.

Example 1:

Input:

nums = [3,4,5], target = 5

Output:

2

Explanation:

Last index of target value is 2

Example 2:

Input:

nums = [1,4,5], target = 2

Output:

-1

Explanation:

Because there is no digit 2 in the array, return -1.

Example 3:

Input:

nums = [3,4,6,6,6,6,7], target = 6

Output:

5

Explanation:

Last index of target value is 5

Constraints:

1 <= nums.length <= 10

4

-10

4

<= nums[i], target <= 10

4

nums

is sorted in ascending order.

Follow up:

Can you write an algorithm with O(log n) runtime complexity?

Code Snippets

JavaScript:

```
/**
* @param {number} target
* @return {number}
*/
Array.prototype.upperBound = function(target) {

};



// [3,4,5].upperBound(5); // 2
// [1,4,5].upperBound(2); // -1
// [3,4,6,6,6,6,7].upperBound(6) // 5
```

TypeScript:

```
interface Array<T> {
upperBound(target: number): number;
}

Array.prototype.upperBound = function(target): number {

};

// [3,4,5].upperBound(5); // 2
// [1,4,5].upperBound(2); // -1
// [3,4,6,6,6,6,7].upperBound(6) // 5
```

Solutions

JavaScript Solution:

```
/**
* Problem: Array Upper Bound
* Difficulty: Easy
* Tags: array, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

/**
* @param {number} target
* @return {number}
*/
Array.prototype.upperBound = function(target) {

};



// [3,4,5].upperBound(5); // 2
// [1,4,5].upperBound(2); // -1
// [3,4,6,6,6,6,7].upperBound(6) // 5
```

TypeScript Solution:

```
/**
* Problem: Array Upper Bound
* Difficulty: Easy
* Tags: array, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

interface Array<T> {
upperBound(target: number): number;
}

Array.prototype.upperBound = function(target): number {

};

// [3,4,5].upperBound(5); // 2
// [1,4,5].upperBound(2); // -1
// [3,4,6,6,6,6,7].upperBound(6) // 5
```