Problem Statement:

You are given an integer array `nums` sorted in ascending order with distinct values. However, this array might be rotated at an unknown pivot index `k` (0-indexed), resulting in a new arrangement where the elements from index `k` to the end of the array are moved to the beginning, and the elements from the beginning to index `k-1` are moved to the end. For example, `[0,1,2,4,5,6,7]` might be rotated at pivot index `3` and become `[4,5,6,7,0,1,2]`.

Given the rotated sorted array `nums` and an integer `target`, you need to find and return the index of `target` in the array. If `target` is not in `nums`, return `-1`.

Your algorithm must have a runtime complexity of O(log n).

Examples:

Example 1:

Input: nums = [4,5,6,7,0,1,2], target = 0

Output: 4

Example 2:

Input: nums = [4,5,6,7,0,1,2], target = 3

Output: -1

Example 3:

Input: nums = [1], target = 0

Output: -1

Constraints:

- 1 <= nums.length <= 5000

- -10^4 <= nums[i] <= 10^4

- All values of `nums` are unique.

- `nums` is an ascending array that is possibly rotated.

- -10^4 <= target <= 10^4

Solution Approach:

To solve this problem efficiently, we can utilize a binary search algorithm. The idea is to divide the array into two halves and determine which half contains the target element. We perform this operation recursively until we find the target element or determine that it doesn't exist in the array.

Here's a high-level overview of the algorithm:

1. Initialize two pointers, `left` and `right`, to indicate the boundaries of the search range.
2. While `left` is less than or equal to `right`, perform the following steps:
    a. Calculate the middle index `mid`.
    b. Check if `nums[mid]` is equal to the `target`. If so, return `mid`.
    c. Determine which half of the array (`left` to `mid-1` or `mid+1` to `right`) contains the target element based on the sorted order of the elements.
3. If the target is not found after the loop, return `-1`.

This algorithm guarantees O(log n) runtime complexity since it eliminates half of the search space in each iteration.