Documentation for Solution to "Search in Rotated Sorted Array II"

Problem Description

Given an integer array nums sorted in nondecreasing order (with possible duplicate values) and rotated at an unknown pivot index k, you need to determine if a given target value exists in the array.

The rotated array is formed by taking the initial sorted array and rotating it at some pivot index k ($0 \leq k <$ nums.length). For example, an array [0,1,2,4,4,4,5,6,6,7] rotated at pivot index 5 becomes [4,5,6,6,7,0,1,2,4,4].

Function Signature

def search(self, nums: List[int], target: int) > bool

Parameters

- **nums:** A list of integers representing the rotated sorted array.
- **target:** An integer representing the target value to search for in the array.

Returns

- True if the target is found in the array.
- False if the target is not found in the array.

Constraints

- 1 <= nums.length <= 5000
- 10^4 <= nums[i] <= 10^4
- nums is guaranteed to be rotated at some pivot.
- 10^4 <= target <= 10^4

Solution Explanation

The algorithm uses a modified binary search to handle the rotated array with duplicates. The main idea is to leverage the properties of the rotated sorted array to narrow down the search space efficiently.

Here's a stepbystep explanation of the approach:

1. **Initialization:** Start with two pointers, left and right, at the beginning and end of the array respectively.

2. **Binary Search Loop:** Continue the loop while left is less than or equal to right.

3. **Middle Element Check:** Calculate the midpoint and check if the mid element is equal to the target. If so, return True.

4. **Handling Duplicates:** If the values at left, mid, and right are the same, increment left and decrement right to bypass the duplicates.

5. **Determine Sorted Part:**

- If the left part is sorted (nums[left] <= nums[mid]):
- If the target is in the range of the left part (nums[left] <= target < nums[mid]), move the right pointer to mid 1.
- Otherwise, move the left pointer to mid + 1.
- If the right part is sorted:
- If the target is in the range of the right part (nums[mid] < target <= nums[right]), move the left pointer to mid + 1.
- Otherwise, move the right pointer to mid 1.

6. Return Result: If the loop exits without finding the target, return False.

Followup Discussion

This problem is similar to the "Search in Rotated Sorted Array" problem, but the presence of duplicates introduces additional complexity. The runtime complexity is generally O(log n) due to the binary search approach. However, in the worst case where duplicates are present, it can degrade to O(n) because the algorithm may need to perform a linear search when it cannot determine which part of the array is sorted. This happens when the values at left, mid, and right are the same, necessitating the adjustment of the pointers (left += 1 and right = 1).