Documentation: Finding the Minimum Element in a Rotated Sorted Array

Problem Description

You are given a sorted array that has been rotated between 1 and n times, where n is the length of the array. The goal is to find the minimum element in this rotated sorted array.

Input

- An array nums of length n where 1 <= n <= 5000.
- Each element in nums is a unique integer.
- The array nums is sorted in ascending order but has been rotated.

Output

- The minimum element in the rotated sorted array.

Constraints

- -5000 <= nums[i] <= 5000
- All elements in nums are unique.
- The array nums is a sorted array that has been rotated between 1 and n times.

Example 1:

- **Input:** nums = [3, 4, 5, 1, 2]
- **Output:** 1
- **Explanation:** The original sorted array [1, 2, 3, 4, 5] was rotated 3 times. The minimum element in this rotated array is 1.

Example 2:

- **Input:** nums = [4, 5, 6, 7, 0, 1, 2]
- **Output:** 0
- **Explanation:** The original sorted array [0, 1, 2, 4, 5, 6, 7] was rotated 4 times. The minimum element in this rotated array is 0.

Example 3:

- **Input:** nums = [11, 13, 15, 17]
- **Output:** 11
- **Explanation:** The original sorted array [11, 13, 15, 17] was rotated 4 times. The minimum element in this rotated array is 11.

Approach

To solve this problem efficiently with a time complexity of (O(log n)), we use a binary search approach. *The key steps in this approach are:*

1. **Initialization:** Set up two pointers, left and right, to represent the start and end of the array, respectively.

2. **Binary Search:**

- Compute the middle index mid.
- *Compare the element at the mid index with the element at the right index to determine which side of the array contains the minimum element:*
  - ➢ If the element at mid is greater than the element at right, the minimum element must be in the right half of the array.
  - ➢ If the element at mid is less than or equal to the element at right, the minimum element must be in the left half or could be the middle element itself.

3. **Update Pointers:** Adjust the left and right pointers based on the comparison to narrow down the search range.

4. **Termination:** The search terminates when left equals right, at which point this index will be the minimum element in the array.

This method leverages the properties of a rotated sorted array to efficiently find the minimum element, avoiding the need for a linear scan of the entire array.