Documentation: Finding a Peak Element in an Array

Problem Overview:

- **Objective:** Given a 0-indexed integer array nums, find a peak element and return its index. A peak element is defined as an element that is strictly greater than its immediate neighbors.

- **Definition:** *For an array nums, a peak element is an element nums[i] such that:*
  - nums[i] > nums[i-1] (if i > 0)
  - nums[i] > nums[i+1] (if i < n-1), where n is the length of the array.

- **Boundary Conditions:**
  - It is assumed that nums[-1] and nums[n] are both -∞ (negative infinity), which means elements at the boundaries of the array are considered greater than non-existent neighbors outside the array.
  - Multiple Peaks: The array may contain multiple peak elements. The algorithm is required to return the index of any one of these peaks.

Constraints:

- The length of the array nums is between 1 and 1000 inclusive.
- Each element in nums is a 32-bit signed integer, ranging from -2^31 to 2^31 1.
- No two adjacent elements in nums are equal, i.e., nums[i] != nums[i + 1] for all valid i.

Approach:

- **Algorithm Type:** The problem is optimally solved using a binary search technique, which takes advantage of the logarithmic nature of the search space.

- **Time Complexity Requirement:** The problem explicitly requires an algorithm that operates in (O(log n)) time complexity, indicating that a divide-and-conquer approach like binary search is suitable.

Solution Strategy:

1. **Initialization:**

- Define two pointers: left initialized to the start of the array and right initialized to the end of the array.

2. **Binary Search Process:**

- Calculate the middle index mid using the formula: mid = (left + right) // 2.
- *Compare the element at mid with its right neighbor (nums[mid + 1]):*
  - If nums[mid] is greater than nums[mid + 1], it implies a peak might be on the left side (including mid itself), so update the right pointer to mid.
  - If nums[mid] is less than nums[mid + 1], the peak must lie on the right side, so update the left pointer to mid + 1.
- Repeat this process until the left pointer equals the right pointer.

3. **Termination Condition:**

- The binary search loop continues until the left and right pointers converge to the same index. At this point, the converged index is the peak element's index.

4. **Result:**

- Return the index at which the left pointer and right pointer converge. This index corresponds to a peak element in the array.

Edge Cases:

- **Single Element Array:** If the array contains only one element, that element is trivially a peak, and the index 0 should be returned.

- **Peak at the Start or End:** If the peak is at the start (nums[0]) or the end (nums[n-1]) of the array, the algorithm will correctly identify it because of the virtual -∞ assumption for elements outside the array bounds.

Summary:

The solution leverages the binary search algorithm to efficiently locate a peak element by systematically narrowing down the potential peak region. The algorithm's ability to discard half of the array in each iteration ensures that the solution meets the required (O(log n)) time complexity. The flexibility of returning any peak element makes the problem simpler, as multiple correct answers are possible.