# **Leetcode 162 – Find Peak Element**

## Problem Understanding

You are given an integer array nums.  
A **peak element** is an element that is **strictly greater than its neighbors**.

Return the **index of any one** of the peak elements.

🔹 You may assume that nums[-1] = nums[n] = -∞.  
🔹 Must solve it in **logarithmic time** → O(log n)

Example:

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

Output: 5 // (6 is a peak element)

## Optimized Java Solution (Binary Search)

class Solution {

public int findPeakElement(int[] nums) {

int left = 0, right = nums.length - 1;

while (left < right) {

int mid = left + (right - left) / 2;

if (nums[mid] > nums[mid + 1]) {

// Peak is in left half including mid

right = mid;

} else {

// Peak is in right half excluding mid

left = mid + 1;

}

}

return left; // or right, since left == right

}

}

## Dry Run Using Table

For input:  
nums = [1, 2, 1, 3, 5, 6, 4]

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| left | right | mid | nums[mid] | nums[mid+1] | action |
| 0 | 6 | 3 | 3 | 5 | left = mid+1 |
| 4 | 6 | 5 | 6 | 4 | right = mid |
| 4 | 5 | 4 | 5 | 6 | left = mid+1 |
| 5 | 5 | - | - | - | done |

🔹 left = 5 → nums[5] = 6 is a peak

## Time / Space Complexity

* **Time:** O(log n) – classic binary search behavior
* **Space:** O(1)

## Alternate Approaches

1. **Linear Scan (Brute Force)**
   * Check every element against neighbors
   * Time: O(n)
   * Space: O(1)
2. **Recursive Binary Search**
   * Can write recursive version of the above logic