2.6 BINARY SEARCH FOR PEAK ELEMENT.

Question:

A peak element is an element that is strictly greater than its neighbors. Given a 0-indexed integer array nums, find a peak element, and return its index. If the array contains multiple peaks, return the index to any of the peaks. You may imagine that nums[-1] = nums[n] = -∞. In other words, an element is always considered to be strictly greater than a neighbor that is outside the array. You must write an algorithm that runs in O(log n) time.

AIM

To design an algorithm that finds the index of a peak element in an array using binary search in O(log n) time.

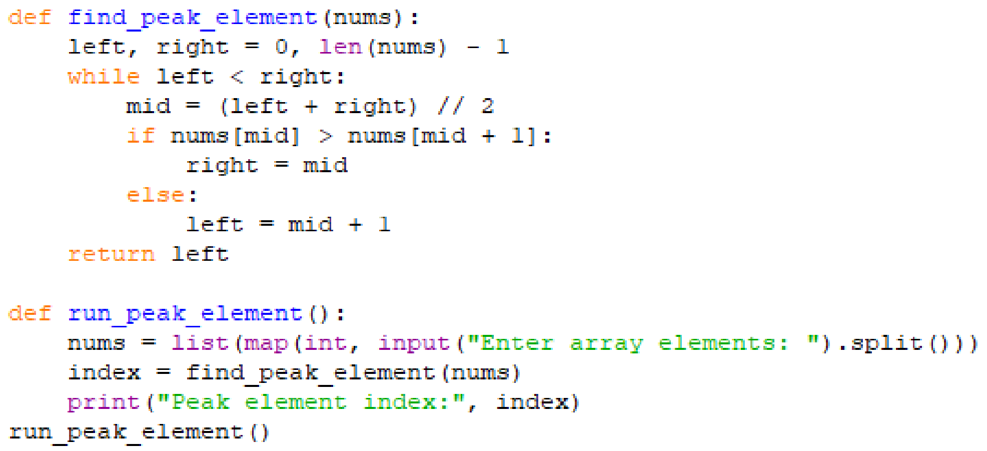
ALGORITHM

1. Let low = 0, high = n-1.
2. While low < high:
    - Compute mid = (low + high) // 2.
    - If nums[mid] > nums[mid + 1]:
    - Then the peak lies in the left half (including mid), so set high = mid.

    Else:

    The peak lies in the right half, so set low = mid + 1.
3. When the loop ends, low == high and that index is a peak.

PROGRAM



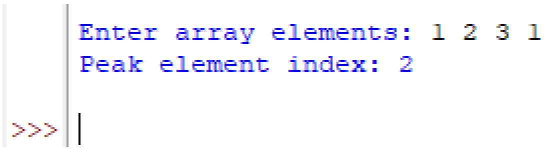
```
def find_peak_element(nums):
    left, right = 0, len(nums) - 1
    while left < right:
        mid = (left + right) // 2
        if nums[mid] > nums[mid + 1]:
            right = mid
        else:
            left = mid + 1
    return left

def run_peak_element():
    nums = list(map(int, input("Enter array elements: ").split()))
    index = find_peak_element(nums)
    print("Peak element index:", index)
run_peak_element()
```

Input:

[1, 2, 3, 1]

Output:



```
Enter array elements: 1 2 3 1
Peak element index: 2

>>> |
```

RESULT:

Thus the program is successfully executed and the output is verified.

PERFORMANCE ANALYSIS:

- Time Complexity:
    - Each step reduces the search space by half → O(log n).
- Space Complexity:
    - O(1) (binary search done in-place, no extra memory).
    - Best Case: Found in first comparison → O(1).
    - Worst Case: O(log n) comparisons until narrowing down to one element.