

## **Exp-2.6**

### **Title:**

Find a peak element's index in an array using  $O(\log n)$  time.

### **Aim:**

To design and implement a Python program to find the index of a peak element in an array, where a peak element is strictly greater than its neighbors, in logarithmic time complexity.

### **Procedure:**

1. Read the input array nums.
2. Use a binary search approach to locate a peak element:
  - Start with pointers left at 0 and right at  $\text{len}(\text{nums}) - 1$ .
  - While  $\text{left} < \text{right}$ :
    - Compute  $\text{mid} = (\text{left} + \text{right}) // 2$ .
    - If  $\text{nums}[\text{mid}] < \text{nums}[\text{mid} + 1]$ , then peak must be in the right half, set  $\text{left} = \text{mid} + 1$ .
    - Else, peak is in the left half (including mid), set  $\text{right} = \text{mid}$ .
3. After loop ends, left points to a peak element.
4. Return left as the peak element index.
5. Print the index.

### **Algorithm:**

1. Start
2. Input array nums.
3. Initialize  $\text{left} = 0$ ,  $\text{right} = \text{len}(\text{nums}) - 1$ .
4. While  $\text{left} < \text{right}$ :
  - $\text{mid} = (\text{left} + \text{right}) // 2$
  - If  $\text{nums}[\text{mid}] < \text{nums}[\text{mid} + 1]$ , set  $\text{left} = \text{mid} + 1$
  - Else, set  $\text{right} = \text{mid}$

5. Return left as the peak index.
6. Stop.

**Input:**

4

1 2 3 1

7

1 2 1 3 5 6 4

**Output:**

2

1 or 0

**Program:**

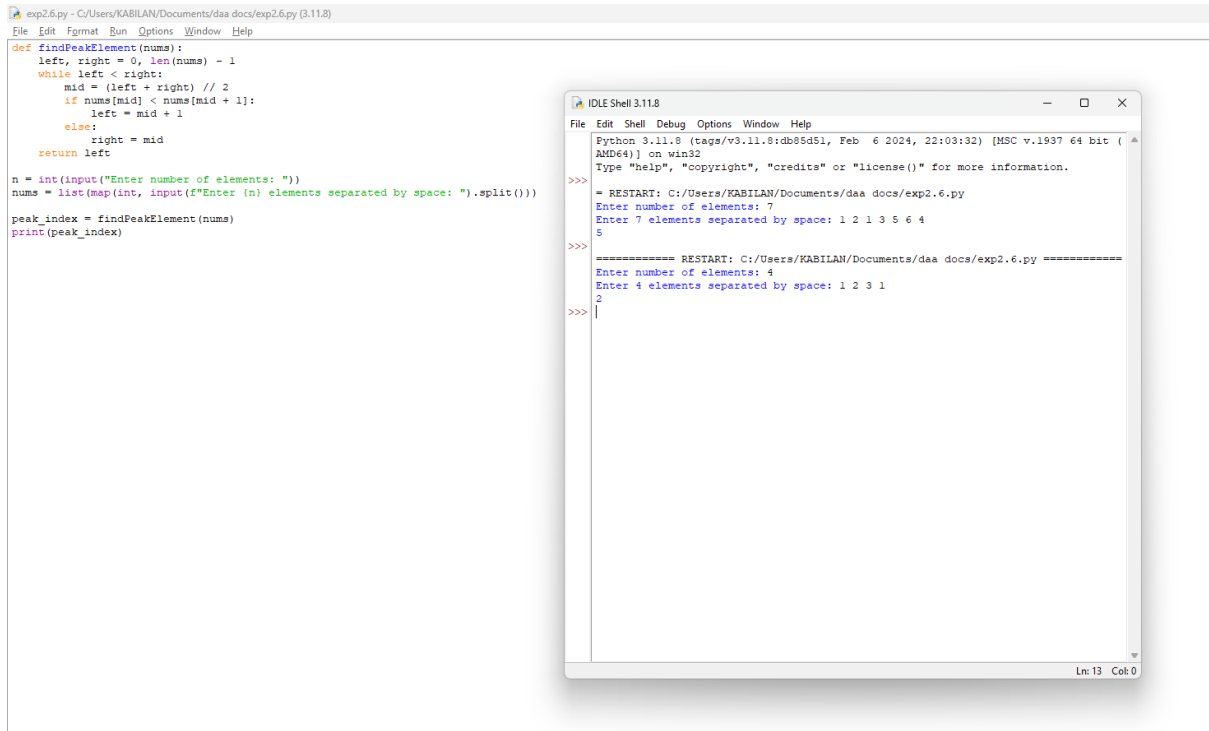
```
def findPeakElement(nums):  
    left, right = 0, len(nums) - 1  
    while left < right:  
        mid = (left + right) // 2  
        if nums[mid] < nums[mid + 1]:  
            left = mid + 1  
        else:  
            right = mid  
    return left  
  
n = int(input("Enter number of elements: "))  
nums = list(map(int, input(f"Enter {n} elements separated by space: ").split()))  
peak_index = findPeakElement(nums)  
print(peak_index)
```

## Performance Analysis:

**Time Complexity:**  $O(\log n)$

**Space Complexity:**  $O(1)$

## Program Output:



The image shows a Python IDE window titled 'exp2.6.py - C:/Users/KABILAN/Documents/daa docs/exp2.6.py (3.11.8)'. The code defines a function `findPeakElement(nums)` that uses a binary search algorithm to find the index of a peak element in a list. The main program prompts the user for the number of elements and the elements themselves, then calls the function and prints the result.

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

n = int(input("Enter number of elements: "))
nums = list(map(int, input(f"Enter {n} elements separated by space: ").split()))
peak_index = findPeakElement(nums)
print(peak_index)
```

The output window shows the execution of the program. It prompts for the number of elements (7), then the elements (1 2 1 3 5 6 4), and prints the peak index (5). It then prompts for the number of elements (4), then the elements (1 2 3 1), and prints the peak index (2).

```
>>>
= RESTART: C:/Users/KABILAN/Documents/daa docs/exp2.6.py
Enter number of elements: 7
Enter 7 elements separated by space: 1 2 1 3 5 6 4
5
>>>
===== RESTART: C:/Users/KABILAN/Documents/daa docs/exp2.6.py =====
Enter number of elements: 4
Enter 4 elements separated by space: 1 2 3 1
2
>>>
```

## Result:

Thus the given program Peak Element Finder is executed and got output successfully.