162. Find Peak Element

Medium 9.5K 4.3K

Companies

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.

Example 1:

```
Input: nums = [1,2,3,1]
Output: 2
Explanation: 3 is a peak element and your function should return the
index number 2.
```

Example 2:

```
Input: nums = [1,2,1,3,5,6,4]
Output: 5
Explanation: Your function can return either index number 1 where the
peak element is 2, or index number 5 where the peak element is 6.
```

Constraints:

- `1 <= nums.length <= 1000`
- -2^{31} `<= nums[i] <=` $2^{31} - 1$
- `nums[i] != nums[i + 1]` for all valid `i`.

Accepted 1M | Submissions 2.3M | Acceptance Rate 45.9%

Approach 1: Using linear scan $O(n)$
at every index 1 to n-2 just check
if (a[i] > a[i-1] && a[i] > a[i+1])

Approach 2: Using binary search

Observations:

Observations:

It is given that nums[-1] and nums[n] are $-\infty$ means nums[0] is always greater than its left neighbour and nums[n-1] is always greater than its right neighbour. And it is also given that we can return any peak element index if there are multiple peaks. So even before starting our algorithm on the array we can check if nums[0] and nums[n-1] are peak elements, if they are then we can simpy return their index.

```cpp
class Solution {
public:
    int findPeakElement(vector<int>& nums) {
        int n=nums.size();
        if(n==1) return 0;
        if(nums[0]>nums[1]) return 0;
        if(nums[n-1]>nums[n-2]) return n-1;
        int l=0,r=n-1;
        while(l<r){
            int mid=(l+r)/2;
            if(nums[mid]<nums[mid+1]) l=mid+1;
            else r=mid;
        }

        return l;
    }
};
```

Here the main intuition behind this algorithm is if we found that

nums[mid] < nums[mid+1]

then we go to the right half hoping that mid+1 indexed element is the peak element, even if its not there will be other element in the right half that is peak element.

0	1	2	3	4	5	6	7	8
1	2	3	4	1	7	6	5	4

$l = 0 \quad r = 8$

$mid = 4$

$l = 5 \quad r = 8$

$mid = 6$

$l = 5 \quad r = 6$

$mid = 5$

$l = 5 \quad r = 5$

So return 5

0	1	2	3	4	5	6	7	8
1	2	3	4	1	7	9	10	1

$l = 0 \quad r = 8$

$mid = 4$

$l = 5 \quad r = 8$

$mid = 6$

$l = 7 \quad r = 8$

$mid = 7$

$l = 7 \quad r = 7$

So return 7.