**Peak Index in a Mountain Array**

An array arr a **mountain** if the following properties hold:

* arr.length >= 3
* There exists some i with 0 < i < arr.length - 1 such that:
  + arr[0] < arr[1] < ... < arr[i - 1] < arr[i]
  + arr[i] > arr[i + 1] > ... > arr[arr.length - 1]

Given a mountain array arr, return the index i such that arr[0] < arr[1] < ... < arr[i - 1] < arr[i] > arr[i + 1] > ... > arr[arr.length - 1].

You must solve it in O(log(arr.length)) time complexity.

**Example 1:**

**Input:** arr = [0,1,0]

**Output:** 1

**Example 2:**

**Input:** arr = [0,2,1,0]

**Output:** 1

**Example 3:**

**Input:** arr = [0,10,5,2]

**Output:** 1

**Constraints:**

* 3 <= arr.length <= 105
* 0 <= arr[i] <= 106
* arr is **guaranteed** to be a mountain array.

Approach 1: Linear Scan

**Intuition and Algorithm**

The mountain increases until it doesn't. The point at which it stops increasing is the peak.

class Solution {

public int peakIndexInMountainArray(int[] A) {

int i = 0;

while (A[i] < A[i+1]) i++;

return i;

}

}

**Complexity Analysis**

* Time Complexity: O(N)*O*(*N*), where N*N* is the length of A.
* Space Complexity: O(1)*O*(1).

#### Approach 2: Binary Search

**Intuition and Algorithm**

The comparison A[i] < A[i+1] in a mountain array looks like [True, True, True, ..., True, False, False, ..., False]: 1 or more boolean Trues, followed by 1 or more boolean False. For example, in the mountain array [1, 2, 3, 4, 1], the comparisons A[i] < A[i+1] would be True, True, True, False.

We can binary search over this array of comparisons, to find the largest index i such that A[i] < A[i+1]. For more on binary search, see the [LeetCode explore topic here.](https://leetcode.com/explore/learn/card/binary-search/)

class Solution {

public int peakIndexInMountainArray(int[] A) {

int lo = 0, hi = A.length - 1;

while (lo < hi) {

int mi = lo + (hi - lo) / 2;

if (A[mi] < A[mi + 1])

lo = mi + 1;

else

hi = mi;

}

return lo;

}

}

**Complexity Analysis**

* Time Complexity: O(\log N)*O*(log*N*), where N*N* is the length of A.
* Space Complexity: O(1)*O*(1).