**1095. Find in Mountain Array: -**

Hard Accepted: 101.6K Submissions: 262.1K Acceptance Rate: 38.8%

*(This problem is an****interactive problem****.)*

You may recall that an array arr is a **mountain array** if and only if:

* 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 mountainArr, return the **minimum** index such that mountainArr.get(index) == target. If such an index does not exist, return -1.

**You cannot access the mountain array directly.** You may only access the array using a MountainArray interface:

* MountainArray.get(k) returns the element of the array at index k (0-indexed).
* MountainArray.length() returns the length of the array.

Submissions making more than 100 calls to MountainArray.get will be judged *Wrong Answer*. Also, any solutions that attempt to circumvent the judge will result in disqualification.

**Example 1:**

**Input:** array = [1,2,3,4,5,3,1], target = 3

**Output:** 2

**Explanation:** 3 exists in the array, at index=2 and index=5. Return the minimum index, which is 2.

**Example 2:**

**Input:** array = [0,1,2,4,2,1], target = 3

**Output:** -1

**Explanation:** 3 does not exist in the array, so we return -1.

**Constraints:**

* 3 <= mountain\_arr.length() <= 104
* 0 <= target <= 109
* 0 <= mountain\_arr.get(index) <= 109

**Code: -**

/\*\*

 \* // This is the MountainArray's API interface.

 \* // You should not implement it, or speculate about its implementation

 \* class MountainArray {

 \*   public:

 \*     int get(int index);

 \*     int length();

 \* };

 \*/

class Solution {

public:

    int leftbs(MountainArray &arr, int start, int end, int &target){

        int low = 0, high = end, mid;

        while(low <= high){

            mid = low + (high - low) / 2;

            if(arr.get(mid) == target)

                return mid;

            else if(arr.get(mid) < target)

                low = mid + 1;

            else

                high = mid - 1;

        }

        return -1;

    }

    int rightbs(MountainArray &arr, int start, int end, int &target){

        int low = 0, high = end, mid;

        while(low <= high){

            mid = low + (high - low) / 2;

            if(arr.get(mid) == target)

                return mid;

            else if(arr.get(mid) < target)

                high = mid - 1;

            else

                low = mid + 1;

        }

        return -1;

    }

    int findInMountainArray(int target, MountainArray &arr) {

        int n = arr.length(), low = 1, high = n - 2, highest;

        while(low <= high){

            int mid = low + (high - low) / 2;

            int left = arr.get(mid - 1);

            int center = arr.get(mid);

            int right = arr.get(mid + 1);

            // peak found immediately

            if(left < center and center > right){

                highest = mid;

                break;

            }

            // found in increasing peak

            else if(left < center and center < right)

                low = mid + 1;

            else

                high = mid - 1;

        }

        int leftsearch = leftbs(arr, 0, highest, target);

        int rightsearch = rightbs(arr, highest+1, n-1, target);

        if(leftsearch != -1 and rightsearch != -1)

            return min(leftsearch, rightsearch);

        return max(leftsearch, rightsearch);

    }

};

**T.C: - O(log N)**

**S.C: - O(1)**