Strivers-A2Z-DSA-Sheet-main\02.Binary Search\1D Arrays\12.Find_single_element_in_sorted_array.cpp

```cpp
/*
QUESTION:
You are given a sorted array consisting of only integers where every element appears exactly twice, except for one element which appears exactly once.

Return the single element that appears only once.

APPROACH:
Since the array is sorted and every element appears exactly twice except for one element, we can use binary search to find the single element.
1. Initialize low = 0 and high = nums.size()-1, where nums is the input array.
2. While low < high, calculate mid = low + (high - low) / 2.
3. Check if mid is an even index (mid % 2 == 0).
    - If nums[mid] is equal to nums[mid + 1], it means the single element is on the right side, so update low = mid + 1.
    - Otherwise, the single element is on the left side, so update high = mid.
4. If mid is an odd index (mid % 2 == 1).
    - If nums[mid] is not equal to nums[mid + 1], it means the single element is on the right side, so update low = mid + 1.
    - Otherwise, the single element is on the left side, so update high = mid.
5. After the loop ends, low will be pointing to the single element.
6. Return nums[low] as the result.

CODE:
*/

int singleNonDuplicate(vector<int>& nums) {
    int low = 0, high = nums.size() - 1;
    while (low < high) {
        int mid = low + (high - low) / 2;
        if (mid % 2 == 0) {
            if (nums[mid] == nums[mid + 1])
                low = mid + 1;
            else
                high = mid;
        } else {
            if (nums[mid] != nums[mid + 1])
                low = mid + 1;
            else
                high = mid;
        }
    }
    return nums[low];
}

// TIME COMPLEXITY: O(log n)
// SPACE COMPLEXITY: O(1)

```