**LeetCode 540: Single Element in a Sorted Array**

**🧠 Problem Statement**

You're given a **sorted array** of integers where every element appears **exactly twice**, except for **one element** that appears only once.  
Return the single element that appears only once.

**Constraints:**

* Must run in **O(log n)** time and **O(1)** space.
* Array is sorted and contains pairs except for one unique element.

**✅ Approach: Binary Search on Index Pattern**

Since the array is sorted and all elements appear twice except one, we can use binary search to find the unique element by observing the **index pairing pattern**:

* Before the single element:
  + First occurrence at even index
  + Second occurrence at odd index
* After the single element:
  + First occurrence at odd index
  + Second occurrence at even index

**🧮 C++ Code**

#include <vector>

using namespace std;

class Solution {

public:

int singleNonDuplicate(vector<int>& nums) {

int low = 0, high = nums.size() - 1;

while (low < high) {

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

// Ensure mid is even for comparison with next element

if (mid % 2 == 1) mid--;

// If pair is valid, move right

if (nums[mid] == nums[mid + 1]) {

low = mid + 2;

} else {

// Unique element is on the left

high = mid;

}

}

return nums[low];

}

};

**🧪 Example**

Input: [1,1,2,3,3,4,4,8,8]

Output: 2

Input: [3,3,7,7,10,11,11]

Output: 10

**📈 Time & Space Complexity**

* **Time**: O(log n) — binary search
* **Space**: O(1) — constant space

**🔍 Intuition Recap**

* Use binary search to narrow down the location of the unique element.
* Adjust mid to always point to the start of a pair.
* If the pair is valid, move right; otherwise, move left.
* When low == high, you've found the unique element.