题目来源于LeetCode上第540号问题：有序数组中的单一元素。题目难度为中等，目前通过率60.2%。 ##题目描述 给定一个只包含整数的有序数组，每个元素都会出现两次，唯有一个数只会出现一次，找出这个数。

示例 1:

输入: [1,1,2,3,3,4,4,8,8]
输出: 2
示例 2:

输入: [3,3,7,7,10,11,11]
输出: 10
注意: 您的方案应该在 O(log n)时间复杂度和 O(1)空间复杂度中运行。

##题目解析 我们先读题，找出题中关键词"包含整数的有序数组"、''元素出现两次"、"只有一个数出现一次"，这里我们可以知道：只出现一次的那个元素所在的有序数组元素个数必为奇数。这个是解题的关键。因为题目要求我们的时间复杂度为O(log n)，因此我们可以用二分搜索法。

解法一：二分搜索

首先将lo和hi分别指向数组首尾元素，mid指向中间元素，这时我们会发现中间元素与其左右两边元素分别有以下三种情况，如：（1）3、3、4，（2）3、4、3，（3）4、3、3。对于第二种情况，我们立马就找到了只出现一次的那个元素。那么对于第一种情况mid=mid-1，那么以（3、3）为界将数组一分为二，判断两边的元素个数，因为我们知道只出现一次的那个元素所在的有序数组元素个数必为奇数，如果（3、3）左边的元素个数为奇数，那么只出现一次的那个数载左边，则将hi移到mid-2位置，即（3、3）的左边，如果（3、3）右边的元素个数为奇数，那么只出现一次的那个数在右边，则将lo移到mid+1位置，即（3、3）的右边。第三种情况与第二种情况类似分析，如果（3、3）左边为奇数，则hi移到mid-1位置，如果（3、3）右边为奇数，则lo移到mid+2位置。以此类推，直到lo=hi，搜索结束。

##动画理解

##代码实现

```
class Solution {
    public int singleNonDuplicate(int[] nums) {
        int lo = 0;
        int hi = nums.length - 1;
        while (lo < hi) {
            int mid = lo + (hi - lo) / 2;
            boolean halvesAreEven = (hi - mid) % 2 == 0;
            if (nums[mid + 1] == nums[mid]) {
                if (halvesAreEven) {
                    lo = mid + 2;
                } else {
                    hi = mid - 1;
                }
            } else if (nums[mid - 1] == nums[mid]) {
                if (halvesAreEven) {
                    hi = mid - 2;
                } else {
                    lo = mid + 1;
                }
```

```
            } else {
                return nums[mid];
            }
        }
        return nums[lo];
    }
}
```

##复杂度分析

- 时间复杂度：O(log n)，在每次循环迭代中，我们将搜索空间减少了一半。
- 空间复杂度：O(1)，仅用了常数空间

##解法二：仅对偶数索引进行二分搜索

我们发现当mid索引为偶数时，mid两边的数组元素个数是偶数，如果mid索引为奇数时，mid两边的数组元素个数为奇数。当mid索引为偶数时，如果mid=mid+1，即解法一的第三种情况，因为mid右边个数为偶数，所以mid+2到hi个数为奇数，那么只出现一次的那个元素肯定在mid的右边，将lo移动到mid+2位置。如果mid！= mid+1，那么只出现一次的那个元素肯定在mid的左边或者就是mid。如果mid 索引为奇数，那么我们为了保证mid索引为偶数，将mid移到mid-1位置，这样mid索引就变成偶数了，重复上述操作，直到hi=lo，搜索结束。

##动画理解

##代码实现

```
class Solution {
    public int singleNonDuplicate(int[] nums) {
        int lo = 0;
        int hi = nums.length - 1;
        while (lo < hi) {
            int mid = lo + (hi - lo) / 2;
            if (mid % 2 == 1) mid--;
            if (nums[mid] == nums[mid + 1]) {
                lo = mid + 2;
            } else {
                hi = mid;
            }
        }
        return nums[lo];
    }
}
```

##复杂度分析

- 时间复杂度：O(log2 / n)=O(log n)，我们仅对元素的一半进行二分搜索。
- 空间复杂度：O(1)，仅用了常数空间