# 题目

整数数组nums按升序排列，数组中的值互不相同 。

在传递给函数之前，nums在预先未知的某个下标k（0<= k < nums.length）上进行了旋转，使数组变为[nums[k], nums[k+1], ..., nums[n-1], nums[0], nums[1], ..., nums[k-1]]（下标从0开始计数）。例如，[0,1,2,4,5,6,7]在下标3处经旋转后可能变为[4,5,6,7,0,1,2]。

给你旋转后的数组nums和一个整数target，如果nums中存在这个目标值target，则返回它的下标，否则返回-1。

你必须设计一个时间复杂度为O(log n)的算法解决此问题。

示例 1：

输入：nums = [4,5,6,7,0,1,2], target = 0

输出：4

示例 2：

输入：nums = [4,5,6,7,0,1,2], target = 3

输出：-1

示例 3：

输入：nums = [1], target = 0

输出：-1

提示：

1 <= nums.length <= 5000

-104 <= nums[i] <= 104

nums中的每个值都 独一无二

题目数据保证 nums 在预先未知的某个下标上进行了旋转

-104 <= target <= 104

# 分析

## 方法一：二分查找

思路：

代码：

class Solution {

public:

int search(vector<int>& nums, int target) {

int n = nums.size();

if (n == 0) return -1;

if (n == 1) {

return nums[0] == target ? 0 : -1;

}

int left = 0, right = n - 1;

while (left <= right) {

int mid = (left + right) / 2;

if (nums[mid] == target) return mid;

// 选择第一个作为哨兵，判断这个是否旋转后的数组

if (nums[0] <= nums[mid]) {

if (nums[0] <= target && target < nums[mid]) {

right = mid - 1;

} else {

left = mid + 1;

}

} else {

if (nums[mid] < target && target <= nums[n-1]) {

left = mid + 1;

} else {

right = mid - 1;

}

}

}

return -1;

}

};

复杂度分析：

时间复杂度：O(logn)，其中n为nums数组的大小。整个算法时间复杂度即为二分查找的时间复杂度O(logn)。

空间复杂度：O(1)。我们只需要常数级别的空间存放变量。