

# 0033. 搜索旋转排序数组

👤 ITCharge ⌚ 大约 3 分钟

- 标签：数组、二分查找
- 难度：中等

## 题目链接

- [0033. 搜索旋转排序数组 - 力扣](#)

## 题目大意

**描述：** 给定一个整数数组  $nums$ ，数组中值互不相同。给定的  $nums$  是经过升序排列后的又进行了「旋转」操作的。再给定一个整数  $target$ 。

**要求：** 从  $nums$  中找到  $target$  所在位置，如果找到，则返回对应下标，找不到则返回  $-1$ 。

**说明：**

- 旋转操作：升序排列的数组  $nums$  在预先未知的第  $k$  个位置进行了右移操作，变成了  $[nums[k], nums[k + 1], \dots, nums[n - 1], \dots, nums[0], nums[1], \dots, nums[k - 1]]$ 。

**示例：**

- 示例 1：

```
输入: nums = [4,5,6,7,0,1,2], target = 0  
输出: 4
```

py

- 示例 2：

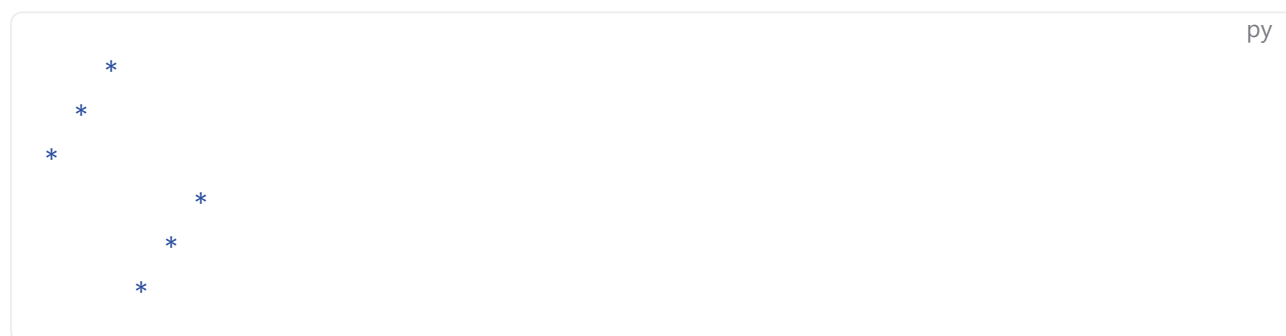
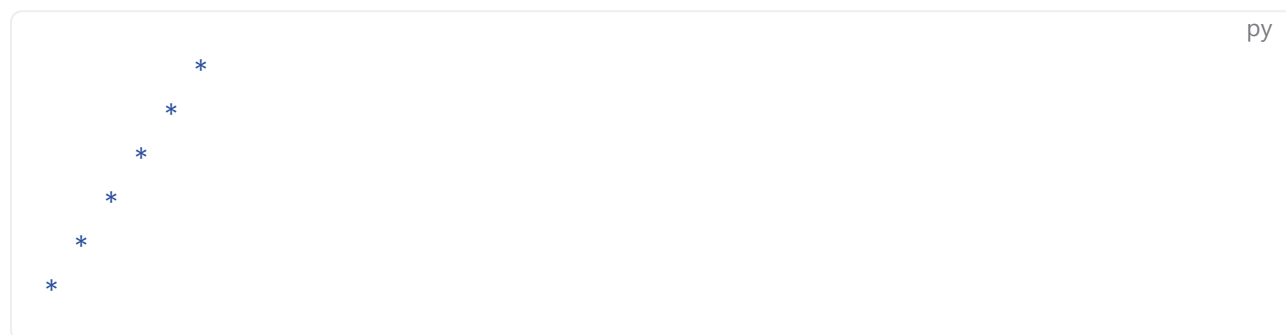
```
输入: nums = [4,5,6,7,0,1,2], target = 3  
输出: -1
```

py

# 解题思路

## 思路 1：二分查找

原本为升序排列的数组 *nums* 经过「旋转」之后，会有两种情况，第一种就是原先的升序序列，另一种是两段升序的序列。



最直接的办法就是遍历一遍，找到目标值 *target*。但是还可以有更好的方法。考虑用二分查找来降低算法的时间复杂度。

我们将旋转后的数组看成左右两个升序部分：左半部分和右半部分。

有人会说第一种情况不是只有一个部分吗？其实我们可以把第一种情况中的整个数组看做是左半部分，然后右半部分为空数组。

然后创建两个指针 *left*、*right*，分别指向数组首尾。让后计算出两个指针中间值 *mid*。将 *mid* 与两个指针做比较，并考虑与 *target* 的关系。

- 如果  $nums[mid] == target$ ，说明找到了 *target*，直接返回下标。
- 如果  $nums[mid] \geq nums[left]$ ，则 *mid* 在左半部分（因为右半部分值都比  $nums[left]$  小）。

- 如果  $nums[mid] \geq target$ , 并且  $target \geq nums[left]$ , 则  $target$  在左半部分, 并且在  $mid$  左侧, 此时应将  $right$  左移到  $mid - 1$  位置。
  - 否则如果  $nums[mid] \leq target$ , 则  $target$  在左半部分, 并且在  $mid$  右侧, 此时应将  $left$  右移到  $mid + 1$  位置。
  - 否则如果  $nums[left] > target$ , 则  $target$  在右半部分, 应将  $left$  移动到  $mid + 1$  位置。
- 如果  $nums[mid] < nums[left]$ , 则  $mid$  在右半部分 (因为右半部分值都比  $nums[left]$  小)。
    - 如果  $nums[mid] < target$ , 并且  $target \leq nums[right]$ , 则  $target$  在右半部分, 并且在  $mid$  右侧, 此时应将  $left$  右移到  $mid + 1$  位置。
    - 否则如果  $nums[mid] \geq target$ , 则  $target$  在右半部分, 并且在  $mid$  左侧, 此时应将  $right$  左移到  $mid - 1$  位置。
    - 否则如果  $nums[right] < target$ , 则  $target$  在左半部分, 应将  $right$  左移到  $mid - 1$  位置。

## 思路 1：代码

```
class Solution:
    def search(self, nums: List[int], target: int) -> int:
        left = 0
        right = len(nums) - 1
        while left <= right:
            mid = left + (right - left) // 2
            if nums[mid] == target:
                return mid

            if nums[mid] >= nums[left]:
                if nums[mid] > target and target >= nums[left]:
                    right = mid - 1
                else:
                    left = mid + 1
            else:
                if nums[mid] < target and target <= nums[right]:
                    left = mid + 1
                else:
                    right = mid - 1

        return -1
```

## 思路 1：复杂度分析

- **时间复杂度：** $O(\log n)$ 。二分查找算法的时间复杂度为  $O(\log n)$ 。
- **空间复杂度：** $O(1)$ 。只用到了常数空间存放若干变量。