搜索旋转排序数组

题目链接

题目要求

升序排列的整数数组 nums 在预先未知的某个点上进行了旋转，例如：[0,1,2,4,5,6,7] 经旋转后可能变为 [4,5,6,7,0,1,2] 。

请你在数组中搜索 target ，如果数组中存在这个目标值，则返回它的索引，否则返回 -1 。

示例1:

```
Input: nums = [4,5,6,7,0,1,2], target = 0
Output: 4
```

示例2:

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

示例3:

```
Input: nums = [1], target = 0
Output: -1
```

题目思路及解法

方法一（暴力解法）

不考虑算法的时间复杂度以及数组本身的两部分的升序序列，直接使用线性扫描的方法进行排序搜索。

- 示例代码

```python
class Solution:
    def search(self, nums: List[int], target: int) -> int:
        # 边界条件
        if not nums:
            return -1
        # 遍历完整数组，有与target相等的元素，直接返回其索引
        for i in range(len(nums)):
             if nums[i] == target:
                 return i
        #如果没有则返回 -1
        return -1
```

- 复杂度计算

  时间复杂度为 $O(N)$，空间复杂度是 $O(1)$。其中 N 为数组的长度，因为单次循环对于完整数组进行遍历，所以时间复杂度是 N。由于仅仅定义了几个常数个的临时变量，所以空间复杂度为 1。

- 运行结果

  执行用时：**32 ms**, 在所有 Python3 提交中击败了 **94.58%** 的用户
  内存消耗：**14.9 MB**, 在所有 Python3 提交中击败了 **19.04%** 的用户

方法二

考虑数组本身的两部分序列，一定有一部分为有序（升序）的，所以我们可以假设一个节点（可以初始化为中间节点），该节点的左右肯定有一边是有序的。基于此我们可以判断target的位置信息。

- 具体过程，共有三种情况
  - 第一种，我们的节点正好为target的位置，则我们直接返回该位置即可。
  - 第二种，如果我们节点的左侧部分是有序（升序）的，那么target就会落在节点的左侧或者右侧；如果是左侧，那么就会满足 $nums[left] <= target <= nums[mid]$ ，则可以继续向左侧搜索；如果是右侧，那么就不会满足上面的条件，那么就需要向右搜索。
  - 第三种，如果我们节点的左侧部分不是有序（升序）的，那么右侧部分一定为有序（升序）的；如果满足 $nums[mid] <= target <= nums[right]$，那么继续向右搜索，否则向左搜索。
- 示例代码

```python
class Solution:
    def search(self, nums: List[int], target: int) -> int:
        # 边界条件
        if not nums:
            return -1
        low, high = 0, len(nums) - 1
        while low <= high:
            mid = (low + high) // 2
            # 找到目标值了直接返回
            if nums[mid] == target:
                return mid
            # 判断左半部分是否有序
            if nums[low] <= nums[mid]:
                # 同时target在 left~mid 中，那么就在这段有序区间查找
                if nums[low] <= target <= nums[mid]:
                    high = mid - 1
                # 否则去反方向查找
                else:
                    low = mid + 1
            # 左部分无序，则右半部分一定有序
            else:
                # 同时target在 mid~rigth 中，那么就在这段有序区间查找
                if nums[mid] <= target <= nums[high]:
                    low = mid + 1
                # 否则去反方向查找
```

```
                else:
                    high = mid - 1
        return -1
```

- 复杂度计算

  时间复杂度为 $O(logN)$ ，空间复杂度是 $O(1)$。其中 N 为数组长度，因为每一次循环就会排除一半的数据，所以时间复杂度是对数级别的。由于仅仅定义了几个常数个的临时变量，所以空间复杂度为 1。

- 运行结果

  执行用时：**36 ms**, 在所有 Python3 提交中击败了 **83.65%** 的用户
  内存消耗：**14.8 MB**, 在所有 Python3 提交中击败了 **20.40%** 的用户