1. 题目
2. 题干

假设按照升序排序的数组在预先未知的某个点上进行了旋转。

( 例如，数组 [0,1,2,4,5,6,7] 可能变为 [4,5,6,7,0,1,2] )。

搜索一个给定的目标值，如果数组中存在这个目标值，则返回它的索引，否则返回 -1 。

你可以假设数组中不存在重复的元素。

你的算法时间复杂度必须是 O(log n) 级别。

1. 示例

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

1. 题解
2. 思路

仔细读题，原有的数组为升序，在某个点进行旋转以后，变为了两个分开的升序数组，如【0, 1, 2, 4, 5, 6, 7】在数字4的位置进行了旋转，4,5,6,7的位置移到了前面，0,1,2的位置移到了后面，变为了两段升序数组【4,5,6,7,0,1,2】

那么，如果用时间复杂度为O(logn)进行查找呢指定的数呢？

从logn的复杂度，我们联想到了二分法，但是二分法是依靠于有序数列进行的，这里是两段有序序列，是没办法直接进行二分法的。但是，如果我们选取了中间值7后发现，左边的数列为4， 5， 6.右边的数列为0， 1， 2，这里恰巧两边都是有序的；但是不论怎么取中间值，就算是假设中间值为6，左边的数列为4，5，右边的数列为7,0,1,2。左右两段数列总会有一段为有序的，因此可以在这一段中使用二分法。

在有序的一端数列中进行二分法后，判断target在不在这一段数列中，如果在，则继续搜索，如果不在，则到另一端数列中查找。

1. 代码实现

C++：

class Solution {

public:

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

        int n = nums.size();

        if (!n)

            return -1;

        if (n == 1) {

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

        }

        //二分法

        int l = 0, r = n - 1;

        while (l <= r) {

            int mid = (l + r) / 2;

            if (nums[mid] == target)

                return mid;

            //如果左边为有序的

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

                //target在有序数列中

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

                    r = mid - 1;

                }

                //target不在有序数列中，则在右边的数列中查找

                else {

                    l = mid + 1;

                }

            }

            //右端数列为有序的

            else {

                //如果target在右端数列中

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

                    l = mid + 1;

                }

                //target不在右端数列中，则在左端数列中查找

                else {

                    r = mid - 1;

                }

            }

        }

        return -1;

    }

};