Leetcode(python) 二分查找 中等题

29两数相除

题目：

给定两个整数，被除数 dividend 和除数 divisor。将两数相除，要求不使用乘法、除法和 mod 运算符。

返回被除数 dividend 除以除数 divisor 得到的商。

示例 1:

输入: dividend = 10, divisor = 3

输出: 3

示例 2:

输入: dividend = 7, divisor = -3

输出: -2

说明:

被除数和除数均为 32 位有符号整数。

除数不为 0。

假设我们的环境只能存储 32 位有符号整数，其数值范围是 [−231, 231 − 1]。本题中，如果除法结果溢出，则返回 231 − 1。

思路：

可以利用位操作(Bit Operator)，即二进制相关的形式来实现。操作符 "<<" 和 ">>" 分别为将数整体左移和右移，例如a<<1表示将a向左移动一位，即变为原来的二倍。

思路:思考了一下，当被除数大于等于除数时(否则的话就为0了)，我们设置两个变量t和p，并分别初始化为除数和1(最小的情况)，当被除数大于等于t的二倍时，将t和p同时扩大二倍(左移)，并将返回值加上p，除数减去t。拿十进制举例:29除以8，8扩大二倍，16小于29，再扩大二倍，超过29，于是29减去之前的16，返回值加上2。第二次循环时因为此时的13小于8的二倍，故加上1，整个循环结束，最终结果为2+1=3，符合要求。此外还要注意判断结果正负号时亦或的作用。

代码：

1. **class** Solution(object):
2. **def** divide(self, dividend, divisor):
3. """
4. :type dividend: int
5. :type divisor: int
6. :rtype: int
7. """
8. result=0
9. temp = 0
10. **if** divisor >= 0 **and** dividend >= 0 **or** divisor < 0 **and** dividend < 0:
11. sign = 1
12. **else**:
13. sign = -1
14. **if** dividend == 0 **or** divisor == 0:
15. **return** 0
16. dividend = abs(dividend)
17. divisor = abs(divisor)
19. **for** i **in** range(32, -1, -1):
20. **if** temp + (divisor << i) <= dividend:
21. temp += divisor << i
22. result |= 1 << i
24. **if** sign < 0:
25. result = -result
26. **elif** sign > 0:
27. result = result
29. **if** result < -2\*\*31 **or** result > (2\*\*31) -1:
30. **return** 2\*\*31 -1
31. **else**:
32. **return** result

33搜索旋转排序数组

题目：

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。

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

思路：

代码：

1. **class** Solution(object):
2. **def** search(self, nums, target):
3. """
4. :type nums: List[int]
5. :type target: int
6. :rtype: int
7. """
8. #先找到两个第二个升序数组的第一项的index
9. l = 0
10. r = len(nums) -1
11. **while** l < r:
12. mid = (l + r)//2
13. **if** nums[mid] > nums[r]:
14. l = mid + 1
15. **else**:
16. r = mid
17. pol = l
18. ans = self.binary\_search(target, nums[:pol])
19. **if** ans == -1:
20. ans = self.binary\_search(target, nums[pol:])
21. **if** ans != -1:
22. ans += len(nums[:pol])
24. **return** ans
26. **def** binary\_search(self, target, nums):
27. index = -1
28. l = 0
29. r = len(nums) - 1
30. **while** l <= r:
31. mid = (l+r)//2
32. **if** nums[mid] < target:
33. l = mid + 1
34. **elif** nums[mid] > target:
35. r = mid - 1
36. **else**:
37. index = mid
38. **break**
39. **return** index