二分算法问题合集（因为二分问题变种太多）

1. 普通二分：对于给定的数组，从中找出target的位置，如果没有返回-1

代码：

l, r = 0, len(arr)-1

            while l <= r:

                mid = l + (r-l+1) // 2

                if arr[mid] < target:

                    l = mid + 1

                elif arr[mid] > target:

                    r = mid - 1

                else:

                    return mid

            return -1

注意的点是：1.while循环的=问题：如上所示mid计算会四舍五入而不是向下取整，所以在循环中如果得到l==r-1的情况，计算出mid=r，此时任意条件都会使得l==r从而跳出循环，但l这个值本身并没有计算到，所以会漏掉情况。2.mid计算：mid计算通常使用向下取整，但是会导致l==r-1时一直mid=l，所以通过+1的方式使得mid可以正确更新，同时保证取到所有的值。

1. 变种1：在排序数组中找到target第一次出现的位置和最后一次出现的位置。

对于第一次出现的位置，和普通问题不同的是当找到mid=target时的处理，因为要找左边第一次出现的位置，如果当前==target，说明当前点mid至少为候选答案，但是左边可能依然后满足条件的解，所以r=mid；

代码：

l, r = 0, len(nums)-1

        while l < r:

            mid = l + (r-l)//2

            if nums[mid] >= target:

                r = mid

            else:

                l = mid + 1

对于最后一次出现的位置，同理，如果mid=target，则右侧依然可能有满足条件的解，所以l=mid.

代码：

l, r = 0, len(nums)-1

        while l < r:

            mid = l + (r - l + 1) // 2

            if nums[mid] <= target:

                l = mid

            else:

                r = mid - 1

此时要注意，如果l==r-1，计算得到mid==l，所以需要+1保证mid的正确更新。

3.变种2：使用二分算法找到数组中的旋转点。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] )。

满足条件的旋转点应该有nums[i]>nums[i+1]，可想，如果当前mid>l，表明左侧序列满足升序条件，旋转点>=mid，所以l=mid。否则，左侧序列必不满足，则旋转点最大位置在mid-1，所以r=mid-1.

l, r = 0, len(nums)-1

        while l < r:

            mid = l + (r - l + 1) // 2

            if nums[l] < nums[mid]:

                l = mid

            else:

                r = mid - 1