”排序数组中的搜索问题，首先想到 二分法 解决,其次，双指针也是高频选项。”

```
# 二分法模板：
# left = 0
# right = len(nums)-1
# while left <= right:
#     m = (left+right) // 2
#     if nums[m] <= target:
        # ！！！ 注意：这个等号是核心与灵魂所在，逼近最右边的target的值,
        # 然后返回>target的第一个值
#         left = m+1
#     # elif nums[m] > target:
#     else:
#         right = m-1
# return left     # 返回的值left为第一次出现的target的值
# resRight = left
```

*题目：LC-JZ52

统计一个数字在排序数组中出现的次数。

输入：nums = [5,7,7,8,8,10], target = 8

输出：2

输入：nums = [5,7,7,8,8,10], target = 6

输出：0

```
class Solution:
    def search(self, nums: List[int], target: int) -> int:
        # # 法1：二分法
        # # 二分法的拓展应用：此题的核心就在于寻找左右边界
        # # 查找右边区间值：
        # left = 0
        # right = len(nums)-1
        # while left <= right:
        #     m = (left+right) // 2
        #     if nums[m] <= target:  # ！！！ 注意：这个等号是核心与灵魂所在，逼近最右边的target的值,然后返回>target的第一个值
        #         left = m+1
        #     elif nums[m] > target:
        #     # else:
        #         right = m-1
        # resRight = left
        # # 查找左边区间值：
        # left = 0
        # right = len(nums)-1
        # # right = resRight     # 若是nums = [],right = 0而不是-1，就会导致数组越界。
        # while left <= right:
```

```
        #     m = (left+right) // 2
        #     if nums[m] < target:
        #         left = m+1
        #     elif nums[m] >= target:    # ！ ！ ！ 注意：这个等号是核心与灵魂所在，逼近最左边的值
        #         right = m-1
        # resLeft = right
        # # 返回结果：
        # res = resRight - resLeft-1
        # return res
        # #---------------------------------------
        # # 法1-2：优化的解法
        # # 自己的代码：
        # left = 0
        # right = len(nums)-1
        # while left <= right:
        #     m = (left+right) // 2
        #     if nums[m] <= target:  # ！ ！ ！ 注意：这个等号是核心与灵魂所在，逼近最右边的值
        #         left = m+1
        #     elif nums[m] > target:
        #     # else:
        #         right = m-1
        # resRight = left
        # # 查找左边区间值：
        # left = 0
        # # 确定下一个查找区间的右边界，可缩小范围
        # # 首先判断是否存在
        # # if right >0 and nums[right] == target:
        # #     pass
        # # else:
        # #     return 0
        # if right >0 and nums[right] != target:
        #     return 0
        # while left <= right:
        #     m = (left+right) // 2
        #     if nums[m] < target:
        #         left = m+1
        #     elif nums[m] >= target:    # ！ ！ ！ 注意：这个等号是核心与灵魂所在，逼近最左边的值
        #         right = m-1
        # resLeft = right
        # # 返回结果：
        # res = resRight - resLeft-1
        # return res

        # ----------------------------------------
```

```
# # 法2：闭包函数解法
# # 查找数字 target0 在数组 nums 中的 插入点(右边)：
# def helper(target0):
#     left = 0
#     right = len(nums)-1
#     while left <= right:
#         m = (left+right) //2
#         if nums[m] <= target0:
#             left += 1
#         else:
#             right -=1
#     resRight = left
#     return resRight
# # 寻找 小于targer的target0 的最有边界
# target2 =  target - 1
# res = helper(target) - helper(target2)
# return res
# ------------------------------------
# # 法2-2
# # 查找数字 target0 在数组 nums 中的 插入点(左边)：
# def helper(target0):
#     left = 0
#     right = len(nums)-1
#     while left <= right:
#         m = (left+right) //2
#         if nums[m] < target0:
#             left += 1
#         else:
#             right -=1
#     resLeft = right
#     return resLeft
# # 寻找大于targer的target0 的最左界
# target0 =  target + 1
# res = helper(target0) - helper(target)
# return res

# ---------------------------------------------
# 法3:迭代法
# ---写了但是超时，没有留下记录

# ---------------------------------------------
# 法4:双指针
# 速度很慢，时间复杂度为O（n）
left = 0
right = len(nums)-1
# while left <= right:        # 条件错误
if target not in nums:
```

```
    return 0
while nums[left]!=target or nums[right] !=target:
# while nums[left]!=target or nums[right] !=target and left <= right:
    # if nums[left] < target:
    if nums[left] != target:
        left += 1
    if nums[right] != target:
    # if nums[right] > target:
        right -= 1
return right-left+1
# if left <= right:
#     return right-left+1
# else:
#     return 0
```