9.25.2024

704 二分法：

二分法听起来简单，在做起来时候会经常遇到边界错误，原因则在于对于边界条件判定的不一致

左闭右开时，left index 在正常判断时必然小于 right index，

举个例子：

[0, 1, 2, 3, 4]， 我们若搜索 4 的时候， 第一步骤将 array 分成[0, 2），[3, 5)

但实际上我们永远不会再取到 2 了。

所以当其不满足判断条件时候，即为失败。

```python
class Solution:
    def search(self, nums: List[int], target: int) -> int:
        # 左闭右开
        left = 0
        right = len(nums)
        while left < right:
            mid = (left + right) // 2
            # 判定
            # 在右边
            if nums[mid] < target:
                left = mid + 1
            # 在左边
            elif nums[mid] > target:
                right = mid
            else:
                return mid
        return -1
```

同理，左闭右闭

```python
class Solution:
    def search(self, nums: List[int], target: int) -> int:
        # 左闭右闭
        left = 0
        right = len(nums) - 1
        while left <= right:
            mid = (left + right) // 2
            # 判定
            # 在右边
            if nums[mid] < target:
                left = mid + 1
            # 在左边
            elif nums[mid] > target:
                right = mid - 1
            else:
                return mid
        return -1
```

27 移除元素

暴力法 $O(n^2)$:

外层找值，内层移动数据

```
def removeElement(self, nums: List[int], val: int) -> int:
    # brute force
    left, right = 0, len(nums)
    while left < right:
        if nums[left] == val:
            for i in range(left+1, right):
                nums[i - 1] = nums[i]
            right -= 1
        else:
            left += 1
    return left
```

如果 left 所在值等于 val， 移除之后在下个循环仍要继续判断 left，因为需要

确定新的 left 是否满足条件

双指针法只需要 O（n）的时间复杂度

```
# 双指针法
slow, fast = 0, 0
while fast < len(nums):
    if nums[fast] != val:
        nums[slow] = nums[fast]
        slow += 1
        fast += 1
    else:
        fast += 1
return slow
```

确保 slow 所在的位置即是更新后数组最后位置即可

977.有序数组的平方

最简单的方法便是算出平方后的值并且重新进行排序，时间复杂度为 O(nlogn)

更有效率的方法是使用双指针，因为大头永远在两端(绝对值最大的元素)

```python
def sortedSquares(self, nums: List[int]) -> List[int]:
    res = [0] * len(nums)
    left, right, res_index = 0, len(nums) - 1, len(nums) - 1
    while left <= right:
        if pow(nums[left], 2) < pow(nums[right], 2):
            res[res_index] = pow(nums[right], 2)
            right -= 1
        else:
            res[res_index] = pow(nums[left], 2)
            left += 1
        res_index -= 1
    return res
```