说明：之前的博客电脑损坏数据丢失还没时间修复，新博客还没来得及搭建，暂时把博客内容放在这里啦。

Day 2

977. Squares of a Sorted Array

暴力解法

```
class Solution:
        def sortedSquares(self, nums: List[int]) -> List[int]:
                newlist = []
                for i in range(len(nums)):
                        newlist.append(nums[i] * nums[i])
                return sorted(newlist)
```

- list.append()的使用
- sorted(list)的使用

双指针法

```
class Solution:
        def sortedSquares(self, nums: List[int]) -> List[int]:
                i, j, k = 0, len(nums)-1, len(nums)-1
                newlist = [-1]*len(nums)
                while i <= j:
                        l = pow(nums[i],2)
                        r = pow(nums[j],2)
                        if l < r:
                                newlist[k] = r
                                j -= 1
                        else:
                                newlist[k] = l
                                i += 1
                        k -= 1
                return newlist
```

注意点：

- while与if结合使用
- 从小到大排列所以新数组从右往左输入新值
- 问题：双指针法的时间复杂度为O(n)，理论上相对于暴力排序的解法O(n + nlog n)还是提升不少的。然而我多次跑出来的结果双指针法用了104ms而暴力解法用了50ms，我知道leetcode上执行的时间不准，但有多次这么不准吗?

相关视频与文章

209. Minimum Size Subarray Sum

滑动窗口

```
class Solution:
        def minSubArrayLen(self, target: int, nums: List[int]) -> int:
                result = float("inf")
                total = index = 0
                for i in range(len(nums)):
                        total += nums[i]
                        while total >= target:
                                result = min(result, i - index + 1)
                                total -= nums[index]
                                index += 1
                return 0 if result== float("inf") else result
```

注意点：

- 比较窗口内数值之和与目标数值
- 设定结果为最大值float("inf")

相关视频与文章

59. Spiral Matrix II

```
class Solution:
        def generateMatrix(self, n: int) -> List[List[int]]:
                matrix = [[0]*n for i in range(n)]
                loop = mid = n//2
                startx = starty = 0
                count = 1
                for offset in range(1, loop+1):
                        for i in range(starty, n-offset):
```

```
                matrix[startx][i] = count
                count += 1
            for i in range(startx, n-offset):
                matrix[i][n-offset] = count
                count += 1
            for i in range(n-offset, starty, -1):
                matrix[n-offset][i] = count
                count += 1
            for i in range(n-offset, startx, -1):
                matrix[i][starty] = count
                count += 1
            startx += 1
            starty += 1
        if n%2 ==1:
            matrix[mid][mid] = count
        return matrix
```

注意点：

- loop与n的关系是1：2
- 最终offset = loop
- 奇数情况的终点

相关视频与文章

Day1

704. Binary Search

```
class Solution:
    def search(self, nums: List[int], target: int) -> int:
        left, right = 0, len(nums)-1
        while left <= right:
            middle = (left + right)//2
            if target < nums[middle]:
                right = middle-1
            elif target > nums[middle]:
                left = middle + 1
            elif target == nums[middle]:
                return middle
        return -1
```

注意点：

- while部分考虑两端闭合情况，所以是 <=
- 一开始脑短路写了`return nums.index(target)`，其实就是`return middle`...

相关视频与文章

27. Remove Element

双指针法

```
class Solution:
        def removeElement(self, nums: List[int], val: int) -> int:
                if len(nums) == 0: return 0
                l,r = 0, len(nums)-1
                while l < r:
                        while(l<r and val != nums[l]):
                                l += 1
                        while(l<r and val == nums[r]):
                                r -= 1
# remove left element covered by right element
                        nums[l], nums[r] = nums[r], nums[l]
                print(nums)
                if nums[l] == val:
                        return l
                else:
                        return l+1
```

注意点：

- 要考虑到空集的情况
- 双指针 左右交互
  相关视频与文章