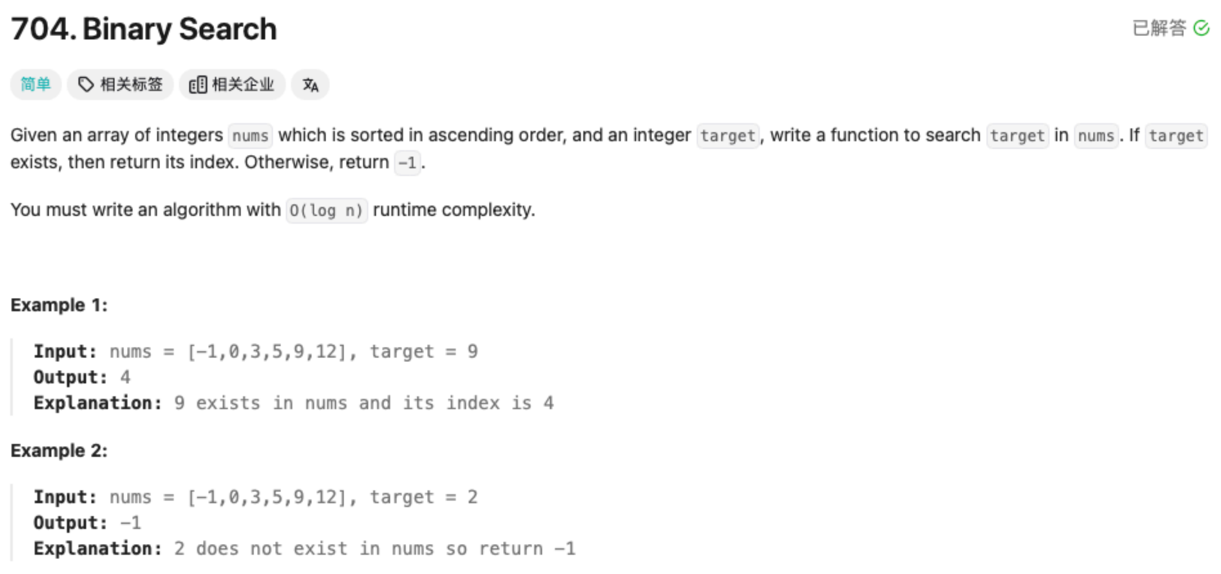
代码随想录 Chapter1 数组Array

建议大家先独立做题，然后看视频讲解，然后看文章讲解，然后在重新做一遍题，最后整理

Day1 (Jan 10)| 704. Binary Search, 35, 34, 27. Remove Elements

LC704 Binary Search (easy)



704. Binary Search

已解答

简单 相关标签 相关企业

Given an array of integers `nums` which is sorted in ascending order, and an integer `target`, write a function to search `target` in `nums`. If `target` exists, then return its index. Otherwise, return `-1`.

You must write an algorithm with `O(log n)` runtime complexity.

Example 1:

```
Input: nums = [-1,0,3,5,9,12], target = 9
Output: 4
Explanation: 9 exists in nums and its index is 4
```

Example 2:

```
Input: nums = [-1,0,3,5,9,12], target = 2
Output: -1
Explanation: 2 does not exist in nums so return -1
```

写在前面：二分法也是一种特殊的双指针，有left, right，还有mid指针。

写法1: 左闭右闭 [left, right]

第一种写法，我们定义 target 是在一个在左闭右闭的区间里，也就是[left, right] （这个很重要）。

区间的定义这就决定了二分法的代码应该如何写，因为定义target在[left, right]区间，所以有如下两点：

- while left <= right: 要使用 **<=** ，因为left == right是有意义的，所以使用 <=
- if nums[middle] > target: right 要赋值为 **middle - 1**，因为当前这个nums[middle]一定不是target，那么接下来要查找的左区间结束下标位置就是 middle - 1

In [1]:
```python
class Solution:
    def search(self, nums:list[int], target: int) -> int:
        # method1: [left, right]
        # 如果没找到, res为-1
        res = -1
        left, right = 0, len(nums) - 1

        while left <= right:
            mid = (left + right) // 2 # floor division: 向下取整
            # or mid = left + (right - left) // 2

            if nums[mid] > target:
                right = mid - 1  # target在左区间, 所以[left, middle - 1]
            elif nums[mid] < target:
                left = mid + 1   # target在右区间, 所以[middle + 1, right]
            else:
                # 找到目标, 更新res
                res = mid
                break

        return res

# test
nums = [1, 2, 3, 4, 5, 6, 7, 8]
target = 5
s = Solution()
s.search(nums, target)
```

Out[1]:
```
4
```

写法2: 左闭右开

如果说定义 target 是在一个在左闭右开的区间里，也就是[left, right) ，那么二分法的边界处理方式则截然不同。

有如下两点：

- while (left < right)，这里使用 < ,因为left == right在区间[left, right)是没有意义的
- if (nums[middle] > target) right 更新为 middle，因为当前nums[middle]不等于target，去左区间继续寻找，而寻找区间是左闭右开区间，所以right更新为middle，即：下一个查询区间不会去比较nums[middle]

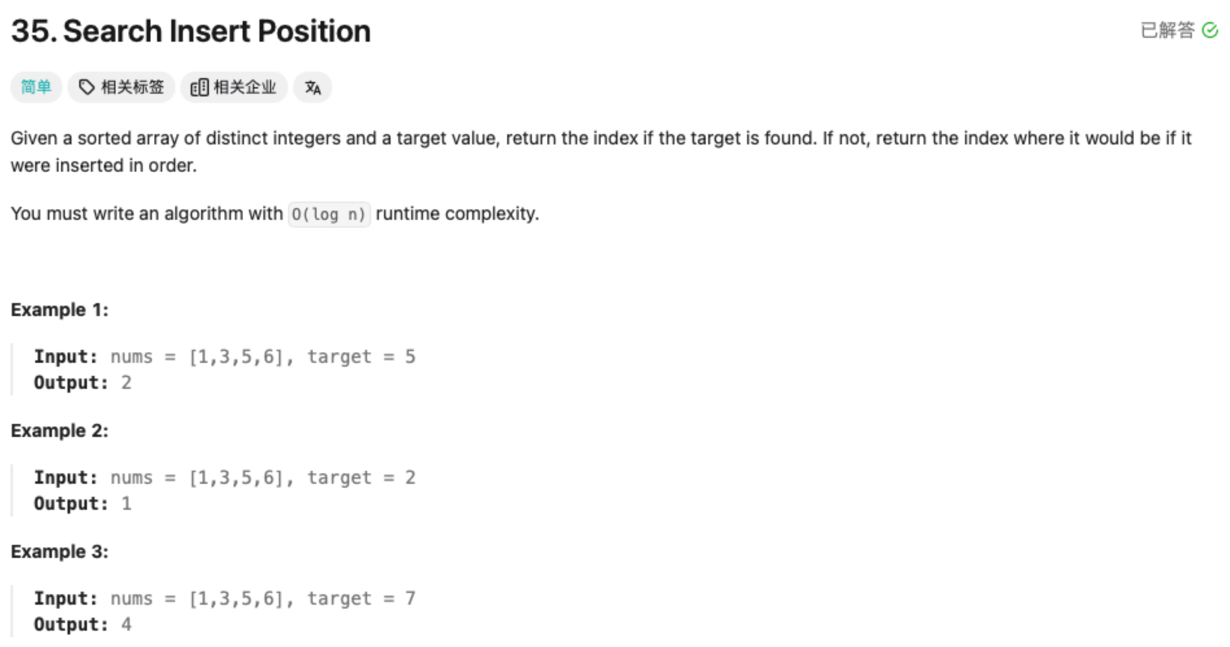
In [8]:
```python
class Solution:
    def search(self, nums: list[int], target: int) -> int:
        # method 2: [left, right)
        # 如果没找到，res为-1
        res = -1
        left, right = 0, len(nums)

        # 因为left == right的时候，在[left, right)是无效的空间
        while left < right:
            mid = (left + right) // 2 # floor division: 向下取整
            # or mid = left + (right - left) // 2

            if nums[mid] > target:
                right = mid   # target在左区间，所以[left, middle)
                              # 注意此时mid 一定不是答案
            elif nums[mid] < target:
                left = mid + 1   # target在右区间，所以[middle + 1, right)
            else:
                # 找到目标，更新res
                res = mid
                break

        return res

# test
nums = [1, 2, 3, 4, 5, 6, 7, 8]
target = 5
s = Solution()
s.search(nums, target)
```

Out[8]:
```
4
```

时间复杂度：O(log N)

空间复杂度：O(1)

LC35 Search Insert Position (Easy)



35. Search Insert Position

Given a sorted array of distinct integers and a target value, return the index if the target is found. If not, return the index where it would be if it were inserted in order.

You must write an algorithm with `O(log n)` runtime complexity.

Example 1:

```
Input: nums = [1,3,5,6], target = 5
Output: 2
```

Example 2:

```
Input: nums = [1,3,5,6], target = 2
Output: 1
```

Example 3:

```
Input: nums = [1,3,5,6], target = 7
Output: 4
```

Comment:

- 本体也是二分法的变体，就比上一题多一步。
- 边界处理的时候自己拿草稿纸多演算几步就好。

写法1: 左闭右闭

In [10]:
```
# 自己写的
class Solution:
    def searchInsert(self, nums: list[int], target: int) -> int:
        # 二分法
        # 法1: 左闭右闭
        left, right = 0, len(nums) - 1
        found = 0

        # left == right, 区间有意义
        while left <= right:
            mid = (left + right) // 2

            if nums[mid] > target:
                right = mid - 1  # target在左区间, 所以[left, middle - 1]
            elif nums[mid] < target:
                left = mid + 1  # target在右区间, 所以[middle + 1, right]
            else:
                # 找到了
                found = 1
                res = mid
                break

        # 如果没找到, 那么最后肯定是left == right
        # 然后进到while loop里面, 然后left = mid + 1  (不可能是right = mid - 1,
        if found == 0:
            res = left # left == right + 1

        return res

# test
nums = [1,3,5,6]
target = 2
s = Solution()
s.searchInsert(nums, target)
```

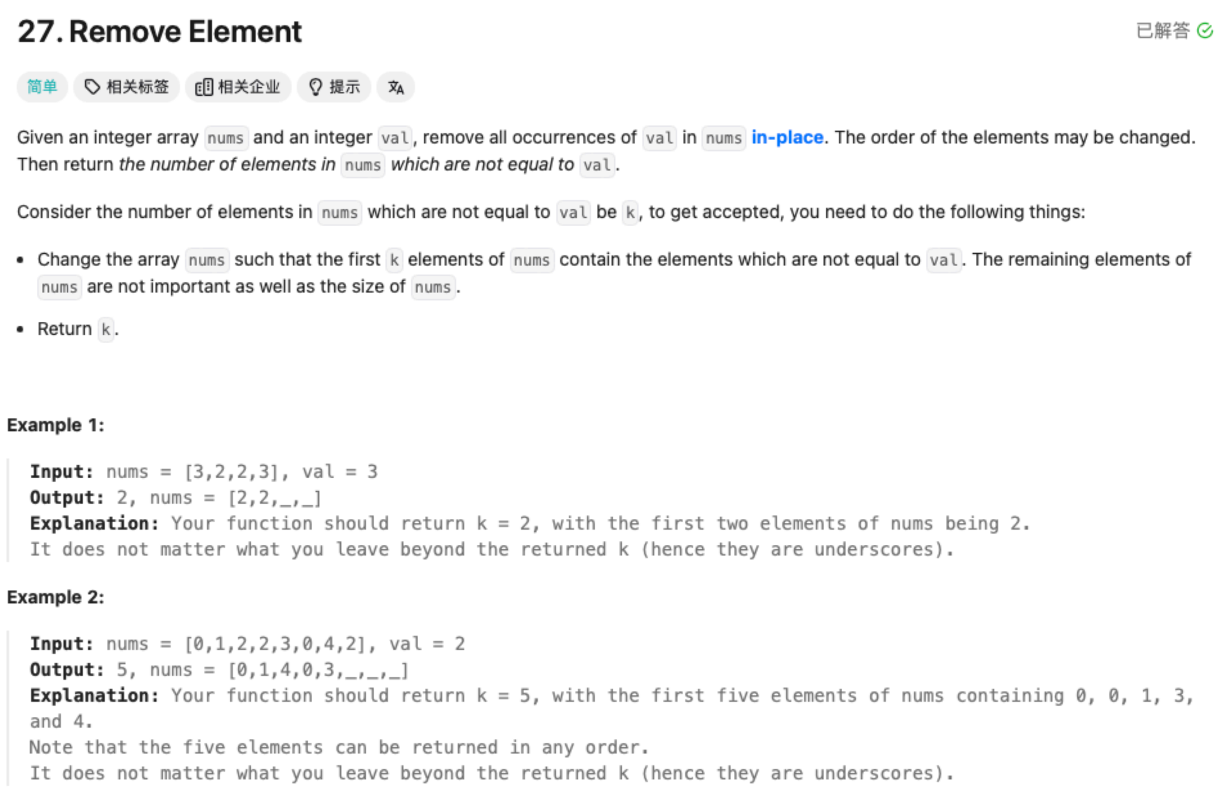
Out[10]:
```
1
```

In [12]:
```
# 代码随想录优化

class Solution:
    def searchInsert(self, nums: list[int], target: int) -> int:
        # 二分法
        # 法1: 左闭右闭
        left, right = 0, len(nums) - 1

        while left <= right:
            mid = (left + right) // 2
            if nums[mid] > target:
                right = mid - 1     # target在左区间, 所以[left, middle - 1]
            elif nums[mid] < target:
                left = mid + 1      # target在右区间, 所以[middle + 1, right]
            else:
                return mid          # return 直接 end function

        return left                 # left == right + 1 跟随想录一样
```

写法2: 左闭右开

In [13]:
```python
# 自己写，随想录没有
class Solution:
    def searchInsert(self, nums: list[int], target: int) -> int:
        # 二分法
        # 法2: 左闭右开
        left, right = 0, len(nums)

        while left < right:
            mid = (left + right) // 2
            if nums[mid] > target:
                right = mid          # target在左区间，所以[left, middle)
            elif nums[mid] < target:
                left = mid + 1       # target在右区间，所以[middle + 1, right)
            else:
                return mid           # return 直接 end function

        # 如果没找到，最后是left == right 出的while loop
        return left
```

LC27 Remove Element



27. Remove Element

已解答

简单 相关标签 相关企业 提示

Given an integer array `nums` and an integer `val`, remove all occurrences of `val` in `nums` **in-place**. The order of the elements may be changed. Then return *the number of elements in* `nums` *which are not equal to* `val`.

Consider the number of elements in `nums` which are not equal to `val` be `k`, to get accepted, you need to do the following things:

- Change the array `nums` such that the first `k` elements of `nums` contain the elements which are not equal to `val`. The remaining elements of `nums` are not important as well as the size of `nums`.
- Return `k`.

Example 1:

```
Input: nums = [3,2,2,3], val = 3
Output: 2, nums = [2,2,_,_]
Explanation: Your function should return k = 2, with the first two elements of nums being 2.
It does not matter what you leave beyond the returned k (hence they are underscores).
```

Example 2:

```
Input: nums = [0,1,2,2,3,0,4,2], val = 2
Output: 5, nums = [0,1,4,0,3,_,_,_]
Explanation: Your function should return k = 5, with the first five elements of nums containing 0, 0, 1, 3,
and 4.
Note that the five elements can be returned in any order.
It does not matter what you leave beyond the returned k (hence they are underscores).
```

法1: 暴力解法

In [16]:
```
class Solution:
    def removeElement(self, nums: list[int], val: int) -> int:
        k = len(nums)
        for i in range(len(nums)):
            while(nums[i] == val):
                nums[i] = 200
                k -= 1
        nums.sort()
        return k
```

时间复杂度: O(N^2)
空间复杂度：O(1)

注：

- 其实也可以用del nums[i] 的方法写暴力，但是比较麻烦。原因是delete 某个元素过后，nums的长度会发生改变，很影响你for loop 的遍历（比如下面这个错误的代码）。所以我们只是巧妙地运用了0 <= nums[i] <= 50 这个限制条件，把和val相等的数字都改成200，这样数组长度不变。
- 另外这里目的是让我们学会del库函数的实现，carl说可以用库函数做的就别用库函数。

In [15]:
```
nums = [1,3,5,6]
del nums[0]
nums
```

Out[15]:
```
[3, 5, 6]
```

In [19]:
```
class Solution:
    def removeElement(self, nums: list[int], val: int) -> int:
        # n 为最初的len(nums)
        n = len(nums)
        k = len(nums)
        for i in range(n):
            while nums[i] == val:
                del nums[i]
                k -= 1

        # 补齐nums的长度
        underscores = ['_' for _ in range(n - k)]

        nums.extend(underscores)
        return k

# test
nums = [1,3,5,6]
val = 3
s = Solution()
s.removeElement(nums, val)
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
/Users/xingqitian/Desktop/Python/Leetcode/Leetcode_array.ipynb Cell 28 line
2
     <a href='vscode-notebook-cell:/Users/xingqitian/Desktop/Python/Leetcode
/Leetcode_array.ipynb#Y135sZmlsZQ%3D%3D?line=18'>19</a> val = 3
     <a href='vscode-notebook-cell:/Users/xingqitian/Desktop/Python/Leetcode
/Leetcode_array.ipynb#Y135sZmlsZQ%3D%3D?line=19'>20</a> s = Solution()
---> <a href='vscode-notebook-cell:/Users/xingqitian/Desktop/Python/Leetcode
/Leetcode_array.ipynb#Y135sZmlsZQ%3D%3D?line=20'>21</a>
s.removeElement(nums, val)

/Users/xingqitian/Desktop/Python/Leetcode/Leetcode_array.ipynb Cell 28 line
7
      <a href='vscode-notebook-cell:/Users/xingqitian/Desktop/Python/Leetcod
e/Leetcode_array.ipynb#Y135sZmlsZQ%3D%3D?line=4'>5</a> k = len(nums)
      <a href='vscode-notebook-cell:/Users/xingqitian/Desktop/Python/Leetcod
e/Leetcode_array.ipynb#Y135sZmlsZQ%3D%3D?line=5'>6</a> for i in range(n):
----> <a href='vscode-notebook-cell:/Users/xingqitian/Desktop/Python/Leetcod
e/Leetcode_array.ipynb#Y135sZmlsZQ%3D%3D?line=6'>7</a>     while nums[i] ==
val:
      <a href='vscode-notebook-cell:/Users/xingqitian/Desktop/Python/Leetcod
e/Leetcode_array.ipynb#Y135sZmlsZQ%3D%3D?line=7'>8</a>         del nums[i]
      <a href='vscode-notebook-cell:/Users/xingqitian/Desktop/Python/Leetcod
e/Leetcode_array.ipynb#Y135sZmlsZQ%3D%3D?line=8'>9</a>         k -= 1

IndexError: list index out of range
```

法2: 双指针 fast, slow

双指针有left right, fast slow, 后面有专门的章节讲。

双指针法（快慢指针法）： 通过一个快指针和慢指针在一个for循环下完成两个for循环的工作。

定义快慢指针

快指针：fast always move to the right every time!

慢指针：slow 的移动是有条件的。
在本题中，只有当nums[fast] != val的时候slow才会往右移动一格。换句话说，**如果nums[fast] == val，那么slow就不动，默默等待被替换。**

In [20]:
```python
# 写法1: use while loop

class Solution:
    def removeElement(self, nums: list[int], val: int) -> int:

        slow, fast = 0, 0

        while fast < len(nums):
            if nums[fast] != val:
                nums[slow] = nums[fast]
                # 如果nums[fast] == val, 那么slow就不动, 默默等待被替换
                slow += 1
            # fast moves to the right every time
            fast += 1

        # slow指针相当于在计数有多少个number != val
        return slow

# test
nums = [1,3,5,6]
val = 3
s = Solution()
s.removeElement(nums, val)
```

Out[20]:
```
3
```

再次强调快慢指针的精妙之处：

- fast index相当于在遍历数组，
- slow index在nums[fast] == val的时候不动，默默等待被替换。另外，slow指针相当于在计数有多少个number != val。

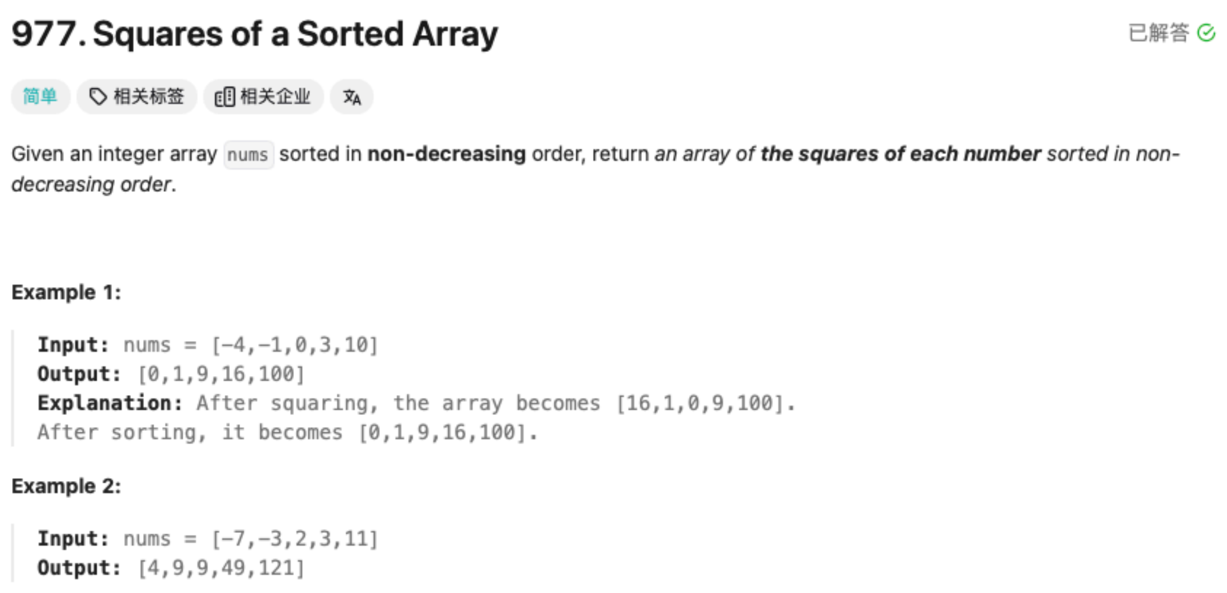
In [21]:
```python
# 写法2: use for loop

class Solution:
    def removeElement(self, nums: list[int], val: int) -> int:

        slow, fast = 0, 0

        # fast 遍历数组的功能在这里体现, 然后不需要 fast += 1 了
        for fast in range(len(nums)):
            if nums[fast] != val:
                nums[slow] = nums[fast]
                # 如果nums[fast] == val, 那么slow就不动, 默默等待被替换
                slow += 1

        # slow指针相当于在计数有多少个number != val
        return slow
```

重点：fast 指针遍历数组的功能在这里体现，然后不需要 fast += 1 了

时间复杂度: O(N)
空间复杂度：O(1)

Day2 (Jan 11)| 977.有序数组的平方, 209.长度最小的子数组, 325, 59.螺旋矩阵II

LC977 Squares of a Sorted Array


977. Squares of a Sorted Array

已解答

简单 相关标签 相关企业

Given an integer array `nums` sorted in **non-decreasing** order, return *an array of **the squares of each number** sorted in non-decreasing order*.

Example 1:

```
Input: nums = [-4,-1,0,3,10]
Output: [0,1,9,16,100]
Explanation: After squaring, the array becomes [16,1,0,9,100].
After sorting, it becomes [0,1,9,16,100].
```

Example 2:

```
Input: nums = [-7,-3,2,3,11]
Output: [4,9,9,49,121]
```

题目建议： 本题关键在于理解双指针思想

题目链接：https://leetcode.cn/problems/squares-of-a-sorted-array/

文章讲解：https://programmercarl.com/0977.%E6%9C%89%E5%BA%8F%E6%95%B0%E7%BB%

视频讲解： https://www.bilibili.com/video/BV1QB4y1D7ep

法1: 暴力排序

每个数平方之后，排个序

这个时间复杂度是 O(n + nlogn)， 可以说是O(nlogn)的时间复杂度，但为了和下面双指针法算法时间复杂度有鲜明对比，我记为 O(n + nlog n)。

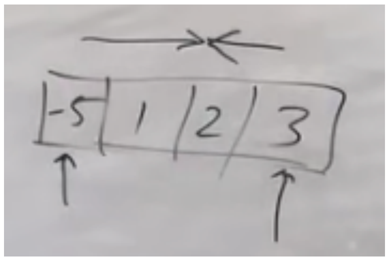
In [3]:
```python
class Solution:
    def sortedSquares(self, nums: list[int]) -> list[int]:
        # method1: 暴力解法 -- 先平方，再排序

        for i in range(len(nums)):
            nums[i] = nums[i] ** 2
        nums.sort()
        return nums
```

In [6]:
```python
# 另一种写法

class Solution:
    def sortedSquares(self, nums: list[int]) -> list[int]:
        return sorted(x*x for x in nums)
```

法2: 双指针 -- left，right

怎么想到双指针?

- 数组其实是有序的， 只不过负数平方之后可能成为最大数了。那么数组平方的最大值就在数组的两端，不是最左边就是最右边，不可能是中间。
- 此时可以考虑双指针法了，而且使用left和right指针，从两边逐步向中间合拢的过程。
- 因为最大值肯定在两端，所以由大到小写出新的数组。



In [4]:
```
class Solution:
    def sortedSquares(self, nums: list[int]) -> list[int]:
        # 法2: 双指针（自己写的）
        res = []
        left, right = 0, len(nums) - 1

        while left <= right:
            if nums[left] ** 2 > nums[right] ** 2:
                res.append(nums[left] ** 2)
                left += 1
            else:
                res.append(nums[right] ** 2)
                right -= 1

        # now res is of descending order, reverse it
        res.reverse() # O(N)的操作

        return res

# test
nums = [-4,-1,0,3,10]
s = Solution()
s.sortedSquares(nums)
```

Out[4]:
```
[0, 1, 9, 16, 100]
```

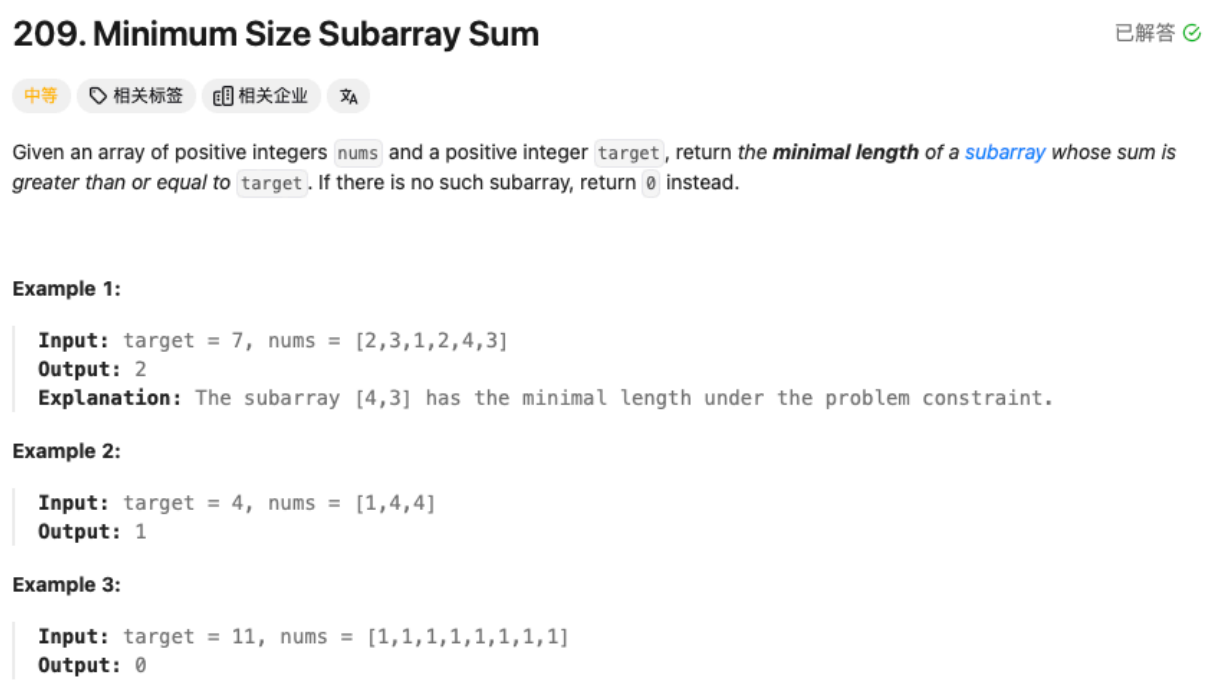
自己写的代码思路很清晰，但是如果不用reverse()这个库函数的话，就得看随想录的写法。要想不用reverse，就得多一个存放结果的指针，每次向前平移一位。

In [7]:
```
class Solution:
    def sortedSquares(self, nums: list[int]) -> list[int]:
        # 法2: 双指针（随想录）
        res = [0] * len(nums) # 需要提前定义列表，存放结果
        left, right = 0, len(nums) - 1
        i = len(nums) - 1      # 存放位置

        while left <= right:
            if nums[left] ** 2 > nums[right] ** 2:
                res[i] = nums[left] ** 2
                left += 1
            else:
                res[i] = nums[right] ** 2
                right -= 1

            # 从后往前填充列表
            i -= 1

        return res
```

这个的时间复杂度为O(n)，相对于暴力排序的解法O(n + nlog n)还是提升不少的。

LC209 Minimum Size Subarray Sum

题目建议： 本题关键在于理解滑动窗口，这个滑动窗口看文字讲解 还挺难理解的，建议大家先看视频讲解。


209. Minimum Size Subarray Sum

已解答

中等 相关标签 相关企业

Given an array of positive integers `nums` and a positive integer `target`, return *the **minimal length** of a subarray whose sum is greater than or equal to* `target`. If there is no such subarray, return `0` instead.

Example 1:

```
Input: target = 7, nums = [2,3,1,2,4,3]
Output: 2
Explanation: The subarray [4,3] has the minimal length under the problem constraint.
```

Example 2:

```
Input: target = 4, nums = [1,4,4]
Output: 1
```

Example 3:

```
Input: target = 11, nums = [1,1,1,1,1,1,1,1]
Output: 0
```

法1: 暴力

在暴力解法中，是一个for循环滑动窗口的起始位置(start)，一个for循环为滑动窗口的终止位置(end)，用两个for循环 完成了一个不断搜索区间的过程

法2: 滑动窗口（from 随想录）

文章讲解：https://programmercarl.com/0209.%E9%95%BF%E5%BA%A6%E6%9C%80%E5%B0%

视频讲解：https://www.bilibili.com/video/BV1tZ4y1q7XE

所谓**滑动窗口（sliding window)**，就是不断的调节子序列的起始位置和终止位置，从而得出我们要想的结果。本质也是双指针，只不过是取这两个指针之间的部分，像一个在滑动的窗口。

所以现在我们有**三种双指针**：
1）left, right
2) slow, fast
3) start, end (sliding window)

思路：

用一个for loop 去做两个for loop的事情（有人说就是两层for loop + 剪枝）

`for j in range(len(nums)):`

- 这里的 j 表示起始位置还是终止位置?
  - 如果j表示起始位置，那么终止位置依然要把所有位置遍历一遍，跟暴力解法没区别
  - 所以j只能表示**终止位置**
  - 本题的重点在于如何移动起始位置

In [9]:
```
'''
写一个伪代码

res = max
i = 0 # 初始位置i
for j in range(len(nums)):
    # 把nums[j]加到res里面
    sum += nums[j]

    **while** sum >= target: # 这里用while因为要持续向后移动起始位置，更新滑动窗口长
        sublen = j - i + 1
        # update res
        res = min(res, sublen)
        # i往右移动
        sum -= nums[i]
        i += 1

return res
'''
```

Out[9]:
```
'\n写一个伪代码\n\nres = max\ni = 0 # 初始位置i\nfor j in range(len(nums)):\n
# 把nums[j]加到res里面\n     sum += nums[j]\n\n     while sum >= target:\n
sublen = j - i + 1\n         # update res\n         res = min(res, sublen)\n
# i往右移动\n         sum -= nums[i]\n         i += 1\n\nreturn res\n'
```

Remark: **while** sum >= target，用while不用if，原因是yao持续向后移动起始位置，更新滑动窗口长度来找到最小。

e.g. nums = [1, 1, 1, 1, 100], target = 100

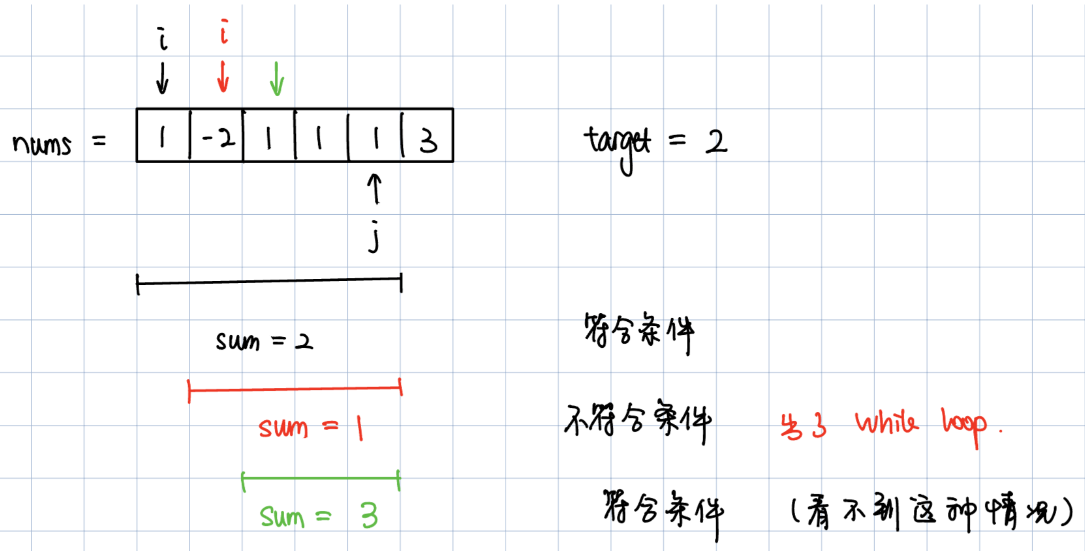
In [10]:
```python
class Solution:
    def minSubArrayLen(self, target: int, nums: list[int]) -> int:

        # 法2: 滑动窗口
        if sum(nums) < target:
            return 0

        res = len(nums)
        sum_ = 0
        i = 0 # start pointer

        for j in range(len(nums)): # end pointer
            sum_ += nums[j]

            # 持续向后移动起始位置，以更新滑动窗口的大小
            while sum_ >= target:
                sublen = j - i + 1
                # update res to be the minimum length
                res = min(res, sublen)

                # start pointer i moves one step to the right
                sum_ -= nums[i]
                i += 1

        return res
```

Comments:

1. Here, `nums` and `target` are both **positive integers,** 所以滑动窗口适用。
2. 如果 `nums` 可以有负数的话，问题会复杂很多。如果我们还用这种方法就会出问题，比如下面这个例子。
    - 最开始，`i = 0`, `j = 0`
    - 然后，`i = 0`, `j` 逐渐搜索到 `j = 4`, `sum = 2` 此时进入 while loop
        - `sublen = 5`, `res = 5`
    - 然后 `i = 1`, `sum = 1` 直接出了while loop
        - 也就是说我们的程序认为在以 `j = 4` 结尾的滑动窗口中，满足条件的最短子数组长度是5
        - 这显然是错误的，因为如果把 `i` 再往后一位的话，`sum = 3` （又会增加），所以满足条件的最短子数组长度是2
3. 我的评价是：由于 `nums` and `target` are both positive integers，所以当 `j` 不动时，不断移动 `i` 使我们的区间变短，**当前的sum一定是越来越小的**。所以我们只要一碰到 `sum < target`，就立刻跳出while loop，此时我们**能保证res = 以 `j` 结尾的subarray中，满足条件的最短子数组长度**。然后终止位置 `j` 向后移动一位，继续寻找以下一个 `j` 结尾的最短的窗口长度。

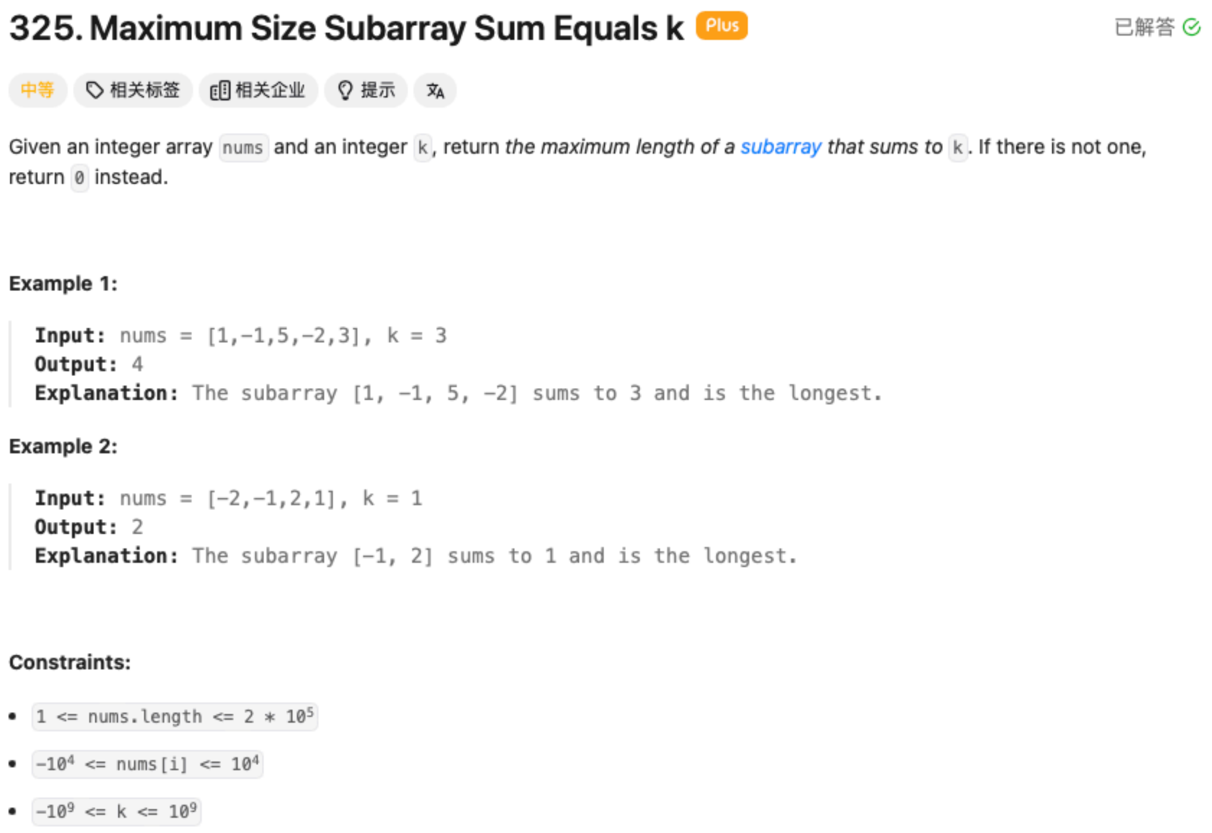
    所以这道题的思路很明确：j一个一个遍历数组，对于每一个 `j`，找到以 `j` 结尾的subarray中, 满足条件的最短子数组长度.

    如果没有positive 这个条件，不断移动 `i` 使我们的区间变短，当前sum不一定是越来越小的（这里是2，1，3），所以这个方法不行。



这里我想补充一道看起来从题面上很类似的题，但解法上完全不同。

LC325 Maximum Size Subarray Sum Equals k


325. Maximum Size Subarray Sum Equals k Plus 已解答

中等 相关标签 相关企业 提示

Given an integer array `nums` and an integer `k`, return *the maximum length of a subarray that sums to* `k`. If there is not one, return `0` instead.

Example 1:

```
Input: nums = [1,-1,5,-2,3], k = 3
Output: 4
Explanation: The subarray [1, -1, 5, -2] sums to 3 and is the longest.
```

Example 2:

```
Input: nums = [-2,-1,2,1], k = 1
Output: 2
Explanation: The subarray [-1, 2] sums to 1 and is the longest.
```

Constraints:

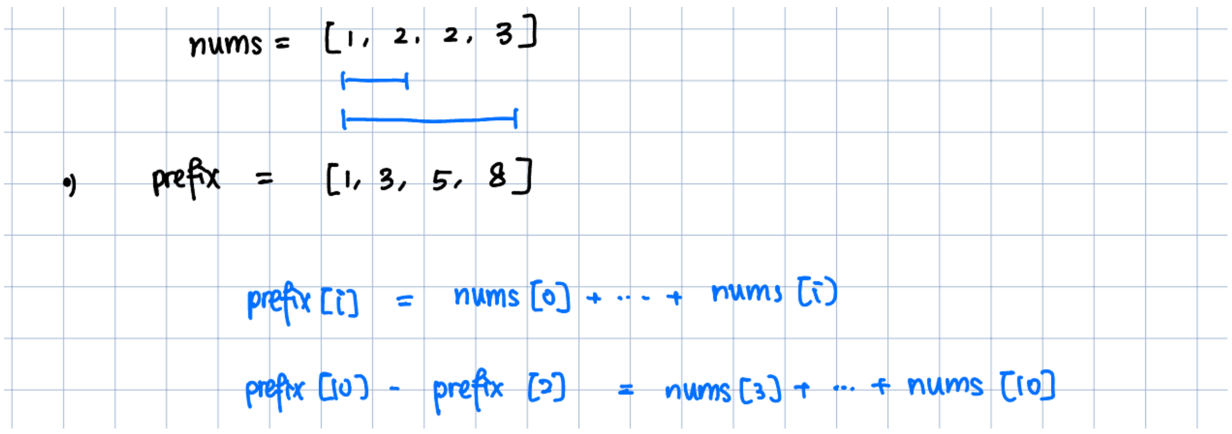
- $1 <= nums.length <= 2 * 10^5$
- $-10^4 <= nums[i] <= 10^4$
- $-10^9 <= k <= 10^9$

这道题就不能用刚才LC209的滑动窗口了，因为这里的nums 和 k 都可以是负数，根据类似的逻辑，滑动窗口不适用。

方法: prefixsum + hashtable

from Leetcode 题解，每次一用prefixsum就特别难想，还是上图吧。

Preliminary: What is **prefixsum**?



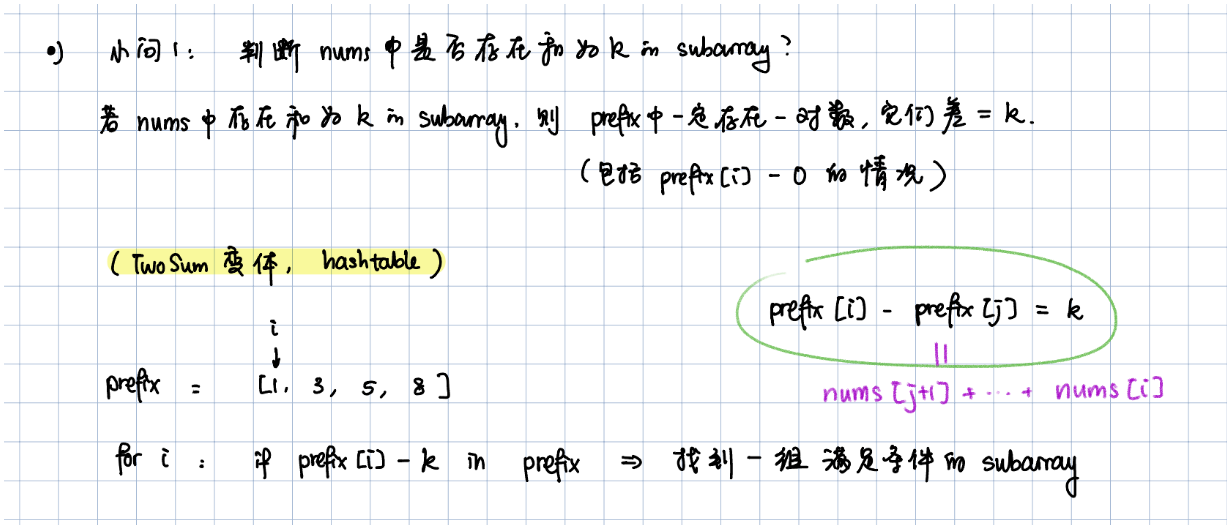
nums = [1, 2, 2, 3]

prefix = [1, 3, 5, 8]

prefix [i] = nums [0] + ··· + nums [i]

prefix [10] - prefix [2] = nums [3] + ··· + nums [10]

prefixsum 通常都是用来做差，和我们的程序目标产生相关性的。



•) 小问 1：判断 nums 中是否存在和为 k 的 subarray？

若 nums 中存在和为 k 的 subarray，则 prefix 中一定存在一对数，它们差 = k.

（包括 prefix [i] - 0 的情况）

（TwoSum 变体，hashtable）

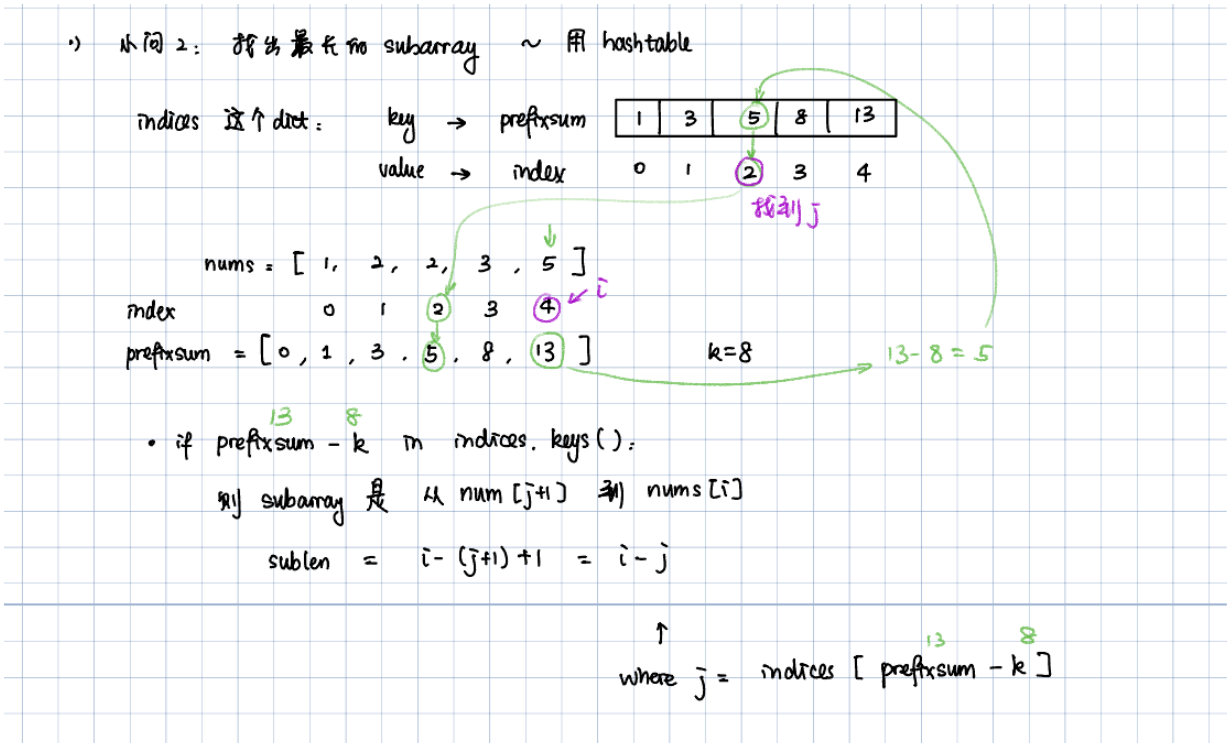
prefix [i] - prefix [j] = k

‖

nums [j+1] + ··· + nums [i]

prefix = [1, 3, 5, 8]

for i : if prefix [i] - k in prefix ⇒ 找到一组满足条件的 subarray



•) 小问 2：找出最长的 subarray ~ 用 hashtable

indices 这个 dict：key → prefixsum [1 | 3 | 5 | 8 | 13]

value → index 0 1 2 3 4

找到 j

nums = [1, 2, 2, 3, 5]

index 0 1 2 3 4 ← i

prefixsum = [0, 1, 3, 5, 8, 13]   k=8   13 - 8 = 5

• if prefixsum - k in indices.keys():

则 subarray 是 从 num [j+1] 到 nums [i]

sublen = i - (j+1) + 1 = i - j

↑

where j = indices [prefixsum - k]

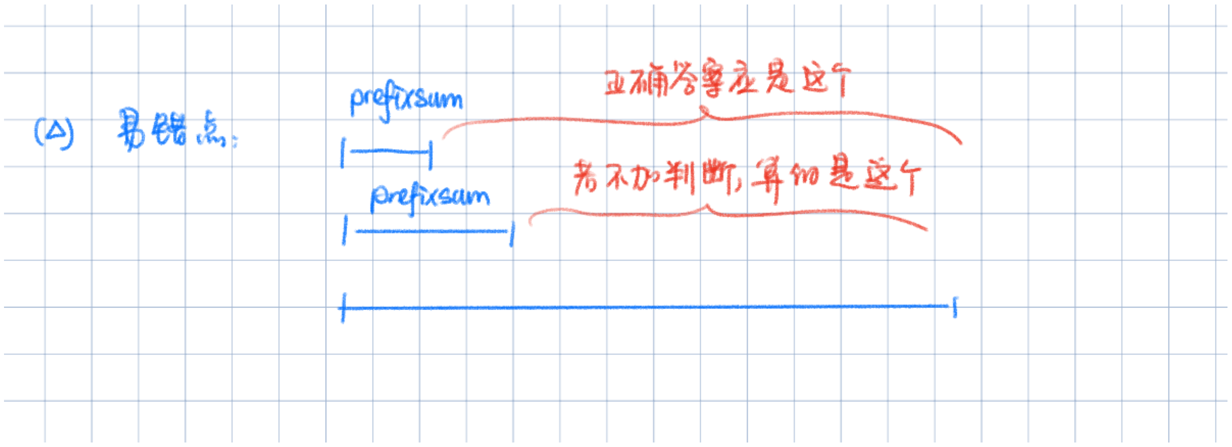
所以coding思路：

1. 初始化三个变量:
    - 一个整数 `prefixSum` 记录 `nums` 的前缀和，初始设为 `0` 。
    - 一个整数 `longestSubarray` 来记录和为 `k` 的最长子数组的长度，初始设为0。
    - 一个哈希映射 `indices` ，这个哈希映射的键是迄今为止见过的前缀和，而值是每个键第一次出现的索引。
2. 遍历 `nums` 。在每个索引 `i` ，将 `nums[i]` 加到 `prefixSum` 上。然后，进行以下检查:
    - 如果 `prefixSum == k` ，那意味着到这个索引位置的数组的和等于 `k` 。更新 `longestSubarray = i + 1` (因为 `i` 是从 0 开始的索引)
    - 如果 `prefixSum - k` 存在于 `indices` 中，那意味着存在一个和为 `k` 的子数组结束在当前的 `i` 。长度将是 `i - indices[prefixSum - k]` 。如果这个长度大于 `longestSubarray` ，更新 `longestSubarray` 。
    - 如果 当前 `prefixSum` 还没有在 `indices` 中出现过，那么设定 `indices[prefixSum] = i` 。只有在它还不存在的时候做这个操作，因为我们只想要这个前缀和最早出现的实例。
3. 返回 `longestSubarray` 。

In [ ]:
```python
# solution
class Solution:
    def maxSubArrayLen(self, nums: list[int], k: int) -> int:
        # prefixsum + hashtable
        prefixsum = 0
        res = 0
        indices = {} # dict, hashtable: store the value of prefixsum and its

        for i in range(len(nums)):
            prefixsum += nums[i]

            # 如果prefixsum不在indices里面，那么就加进去
            # 如果已经在了，那么就不用管了，因为我们要找的是最长的subarray  (见下面rem
            if prefixsum not in indices.keys():
                indices[prefixsum] = i # key is the prefixsum, value is the

            # 数组从0到i的和等于k
            if prefixsum == k:
                sublen = i+1

            # 数组从indices[prefixsum - k]+1 到i 的和等于k
            if prefixsum - k in indices.keys():
                sublen = i - indices[prefixsum - k]
                # update res
                res = max(res, sublen)

        return res
```

Comment: 这里跟two sum 那道题很像的一点是hashtable的巧用 -- **key是某个东西的数值，value反而是index**。这是因为我们想通过prefixsum的值来找它在prefixsum数组中的位置，进而找到满足条件的subarray的初始和终止位置。当我们想通过A来找B时，应该把A设成dictionary 的 key，B设成value。

Remark:



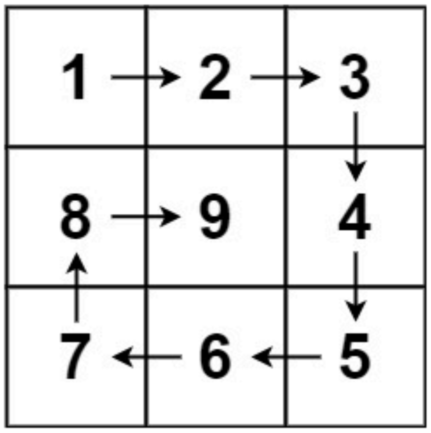
LC59 Spiral Matrix II

59. Spiral Matrix II

中等 相关标签 相关企业

Given a positive integer `n`, generate an `n x n` `matrix` filled with elements from `1` to `n²` in spiral order.

Example 1:



```
Input: n = 3
Output: [[1,2,3],[8,9,4],[7,6,5]]
```

本题并不涉及到什么算法，就是模拟过程，但却十分考察对代码的掌控能力。

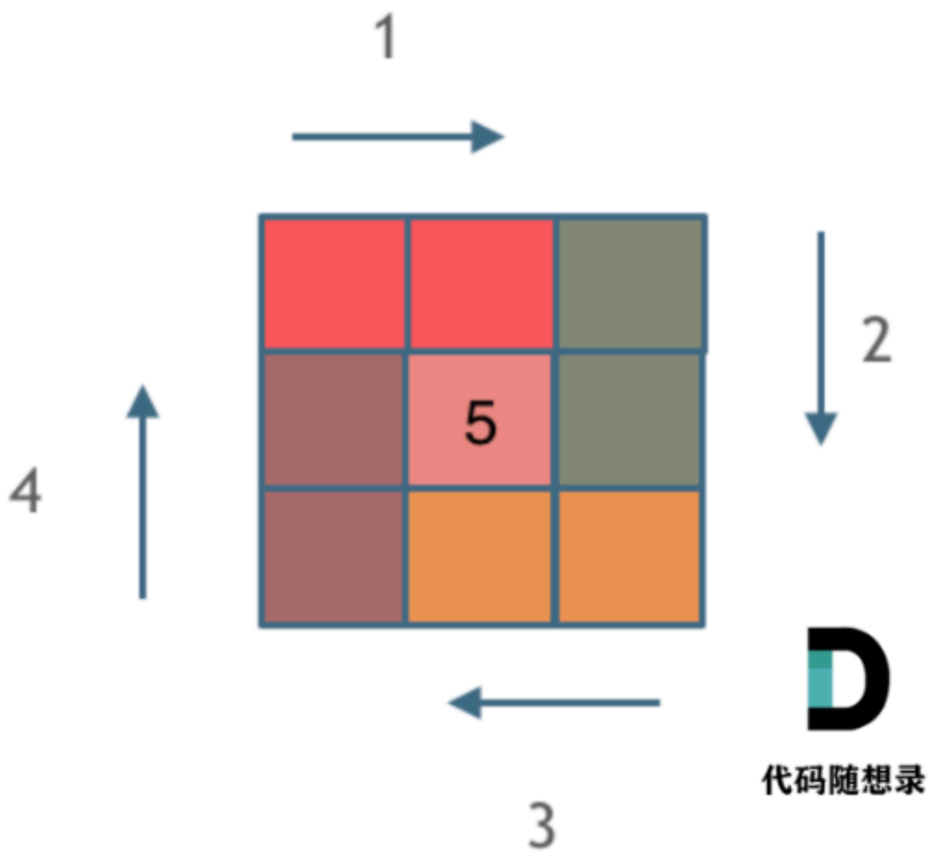
而求解本题依然是要坚持循环不变量原则。

模拟顺时针画矩阵的过程:

填充上行从左到右
填充右列从上到下
填充下行从右到左
填充左列从下到上
由外向内一圈一圈这么画下去。

这里一圈下来，我们要画每四条边，这四条边怎么画，每画一条边都要坚持一致的左闭右开，或者左开右闭的原则，这样这一圈才能按照统一的规则画下来。

那么我按照左闭右开的原则，来画一圈，大家看一下:



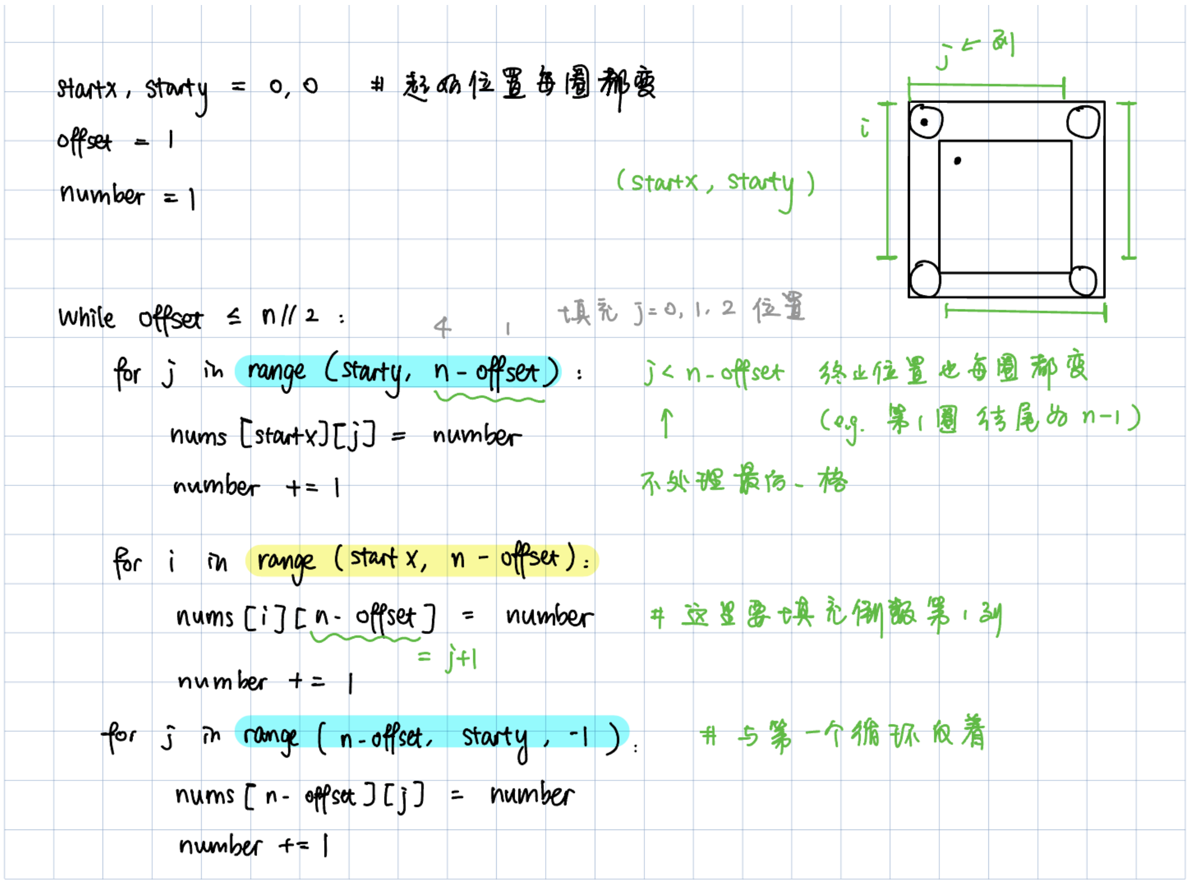
这里每一种颜色，代表一条边，我们遍历的长度，可以看出每一个拐角处的处理规则，拐角处让给新的一条边来继续画。

这也是坚持了每条边左闭右开的原则。

继续思路：我们设一圈 = 画了四条边，那么一共要画多少圈呢？答案是 `n // 2` 圈。这是因为矩阵的边长 = n, 每填充一圈就会少左右两边的格子，所以长度 / 2（举一个 $n = 4$ 的例子就明白了）。
这里还有一个小点要注意：

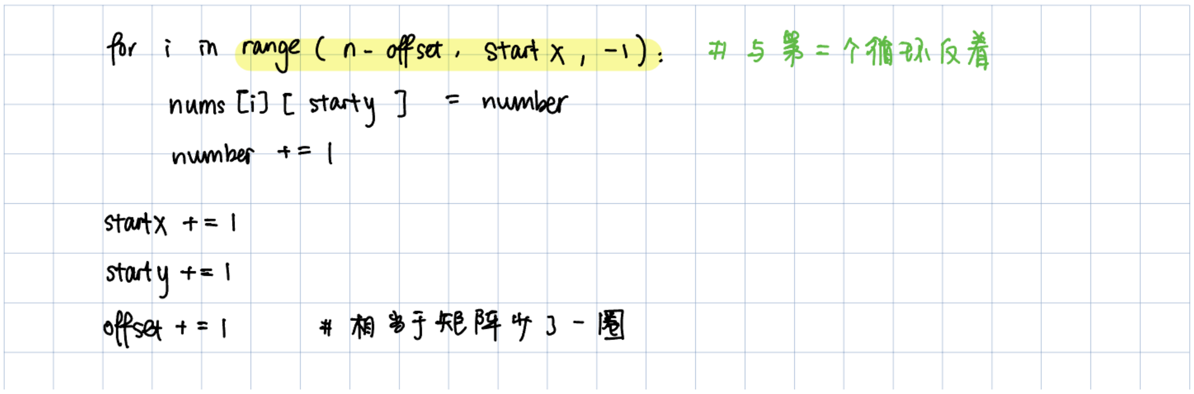
- 当n为偶数的时候正好填充完（e.g. $n = 4$）
- 当n为奇数的时候，最后会剩下最中间的一个格子（e.g. $n = 3$），所以要加一个判断语句处理这种情况（`mid = n // 2`）。

伪代码：注意这里i代表行index，j代表列index，`(i, j)` 和 `(startx, starty)` 对应。



j ← 列
i
(startx, starty)

```
startx, starty = 0, 0    # 起始位置每圈都变
offset = 1
number = 1

while offset ≤ n//2 :                               填充 j=0, 1, 2 位置
    for j in range (starty, n - offset) :           j< n-offset   终止位置也每圈都变
        nums [startx][j] = number                   ↑             (e.g. 第1圈 结尾为 n-1)
        number += 1                                 不处理最后一格

    for i in range (start x, n - offset):
        nums [i] [n- offset] = number               # 这里要填充倒数第1列
                  = j+1
        number += 1
    for j in range ( n-offset, starty , -1 ):       # 与第一个循环反着
        nums [ n- offset ] [j] = number
        number += 1
```



```
    for i in range ( n - offset , start x , -1):    # 与第二个循环反着
        nums [i] [ starty ] = number
        number += 1

    startx += 1
    starty += 1
    offset += 1          # 相当于矩阵少了一圈
```

```
In [3]: class Solution:
            def generateMatrix(self, n: int) -> list[list[int]]:
                nums = [[0 for _ in range(n)] for _ in range(n)]
                startx, starty = 0, 0
                mid = n // 2
                offset = 1 # 这是第几圈
                number = 1 # 要填充的数字


                while offset <= n // 2: # 从第一圈开始，一共n // 2圈
                    # 填充上行从左到右，左闭右开
                    for j in range(starty, n - offset, +1):
                        nums[startx][j] = number
                        number += 1
                    # 填充右列从上到下
                    for i in range(startx, n - offset, +1):
                        nums[i][n - offset] = number
                        number += 1
                    # 填充下行从右到左
                    for j in range(n - offset, starty, -1):
                        nums[n - offset][j] = number
                        number += 1
                    # 填充左列从下到上
                    for i in range(n - offset, startx, -1):
                        nums[i][starty] = number
                        number += 1

                    startx += 1
                    starty += 1
                    offset += 1

                # n是奇数时，中间剩一个格子没填充
                if n % 2 != 0:
                    nums[mid][mid] = number

                return nums
```

总结

1. List vs. Array

Python List

- A **list** is a built-in Python data structure that can hold items of different data types.
- Lists are dynamic; they can grow or shrink, and elements can be of **any type**, including other lists.
- They come with a variety of built-in methods for manipulation such as adding `append()`, removing `remove()`, and changing `extend()` `pop()` elements, and more.
- In Python, lists are created using square brackets `[]`, like so: `my_list = [1,`

`'two', 3.0]`.

Python Array

In Python, an **array** is a data structure available through the `array` module that stores elements of **the same data type**:

- An array is more memory efficient and can offer better performance for certain tasks compared to lists, especially when dealing with a large number of elements.
- Arrays are particularly useful when you're working with **numerical data** and require efficient storage and manipulation of elements.
- To use arrays in Python, you must import the `array` module with the statement `import array`.

For example, creating an array of integers with the `array` module looks like this:

```
from array import array
int_array = array('i', [1, 2, 3, 4])
```

Python Array in LeetCode

- In LeetCode problems, when "array" is mentioned, it usually refers to a concept similar to lists, but the term is used in a more general, language-agnostic way.
- Unlike in Python's `array` module where arrays are homogenous, in LeetCode "array" problems, you can use Python lists without importing any module.
- The "array" problems are often about manipulating indexed collections of items, typically assuming they are of the same type for simplicity.
- In LeetCode, when you're solving "array" problems, you just use a Python list as you normally would.

For example, a LeetCode problem might ask you to manipulate an array of integers. In Python, you would use a list:

```
# LeetCode "array" problem using a Python list
nums = [2, 7, 11, 15]
```

需要两点注意的是

- 数组下标都是从0开始的。Indexing Starts at Zero.
- 数组内存空间的地址是连续的。正是因为数组的在内存空间的地址是连续的，所以我们在删除或者增添元素的时候，就难免要移动其他元素的地址
  (**Contiguous Memory Allocation**: While Python lists are not arrays in the traditional sense, they behave similarly in that they allow for efficient indexing. The difference is that **Python lists** can grow or shrink dynamically, and they can hold items of different types.)

2. 三种方法

1. **二分法**：特殊的双指针-- left，right & mid
2. **双指针**：
    - **left，right** （也有不是二分法的左右指针）
    - **fast, slow** (fast 永远遍历整个数组)
      双指针法（快慢指针法）在数组和链表的操作中是非常常见的，很多考察数组和链表操作的面试题，都使用双指针法。
    - **start, end**(滑动窗口)
3. **滑动窗口**（本质还是双指针，只不过是取start, end 中间的部分）

https://code-thinking-1253855093.file.myqcloud.com/pics/数组总结.png

In [ ]:

```
```

In [ ]:

```
```

In [4]:

```
pip install nbconvert
```

```
Requirement already satisfied: nbconvert in /Users/xingqitian/Library/Python
/3.10/lib/python/site-packages (7.11.0)
Requirement already satisfied: beautifulsoup4 in /Library/Frameworks/Python.
framework/Versions/3.10/lib/python3.10/site-packages (from nbconvert) (4.11.
1)
Requirement already satisfied: bleach!=5.0.0 in /Users/xingqitian/Library/Py
thon/3.10/lib/python/site-packages (from nbconvert) (6.1.0)
Requirement already satisfied: defusedxml in /Users/xingqitian/Library/Pytho
n/3.10/lib/python/site-packages (from nbconvert) (0.7.1)
Requirement already satisfied: jinja2>=3.0 in /Library/Frameworks/Python.fra
mework/Versions/3.10/lib/python3.10/site-packages (from nbconvert) (3.1.2)
Requirement already satisfied: jupyter-core>=4.7 in /Users/xingqitian/Librar
y/Python/3.10/lib/python/site-packages (from nbconvert) (5.5.0)
Requirement already satisfied: jupyterlab-pygments in /Users/xingqitian/Libr
ary/Python/3.10/lib/python/site-packages (from nbconvert) (0.2.2)
Requirement already satisfied: markupsafe>=2.0 in /Library/Frameworks/Python
.framework/Versions/3.10/lib/python3.10/site-packages (from nbconvert) (2.1.
1)
Requirement already satisfied: mistune<4,>=2.0.3 in /Users/xingqitian/Librar
y/Python/3.10/lib/python/site-packages (from nbconvert) (3.0.2)
Requirement already satisfied: nbclient>=0.5.0 in /Users/xingqitian/Library/
Python/3.10/lib/python/site-packages (from nbconvert) (0.9.0)
Requirement already satisfied: nbformat>=5.7 in /Users/xingqitian/Library/Py
thon/3.10/lib/python/site-packages (from nbconvert) (5.9.2)
Requirement already satisfied: packaging in /Users/xingqitian/Library/Python
/3.10/lib/python/site-packages (from nbconvert) (21.3)
Requirement already satisfied: pandocfilters>=1.4.1 in /Users/xingqitian/Lib
rary/Python/3.10/lib/python/site-packages (from nbconvert) (1.5.0)
Requirement already satisfied: pygments>=2.4.1 in /Users/xingqitian/Library/
Python/3.10/lib/python/site-packages (from nbconvert) (2.13.0)
```

```
Requirement already satisfied: tinycss2 in /Users/xingqitian/Library/Python/
3.10/lib/python/site-packages (from nbconvert) (1.2.1)
Requirement already satisfied: traitlets>=5.1 in /Users/xingqitian/Library/P
ython/3.10/lib/python/site-packages (from nbconvert) (5.13.0)
Requirement already satisfied: six>=1.9.0 in /Users/xingqitian/Library/Pytho
n/3.10/lib/python/site-packages (from bleach!=5.0.0->nbconvert) (1.16.0)
Requirement already satisfied: webencodings in /Library/Frameworks/Python.fr
amework/Versions/3.10/lib/python3.10/site-packages (from bleach!=5.0.0->nbco
nvert) (0.5.1)
Requirement already satisfied: platformdirs>=2.5 in /Users/xingqitian/Librar
y/Python/3.10/lib/python/site-packages (from jupyter-core>=4.7->nbconvert) (
4.0.0)
Requirement already satisfied: jupyter-client>=6.1.12 in /Users/xingqitian/L
ibrary/Python/3.10/lib/python/site-packages (from nbclient>=0.5.0->nbconvert
) (7.4.9)
Requirement already satisfied: fastjsonschema in /Users/xingqitian/Library/P
ython/3.10/lib/python/site-packages (from nbformat>=5.7->nbconvert) (2.18.1)
Requirement already satisfied: jsonschema>=2.6 in /Users/xingqitian/Library/
Python/3.10/lib/python/site-packages (from nbformat>=5.7->nbconvert) (4.19.2
)
Requirement already satisfied: soupsieve>1.2 in /Library/Frameworks/Python.f
ramework/Versions/3.10/lib/python3.10/site-packages (from beautifulsoup4->nb
convert) (2.3.2.post1)
Requirement already satisfied: pyparsing!=3.0.5,>=2.0.2 in /Users/xingqitian
/Library/Python/3.10/lib/python/site-packages (from packaging->nbconvert) (3
.0.9)
Requirement already satisfied: attrs>=22.2.0 in /Users/xingqitian/Library/Py
thon/3.10/lib/python/site-packages (from jsonschema>=2.6->nbformat>=5.7->nbc
onvert) (23.1.0)
Requirement already satisfied: jsonschema-specifications>=2023.03.6 in /User
s/xingqitian/Library/Python/3.10/lib/python/site-packages (from jsonschema>=
2.6->nbformat>=5.7->nbconvert) (2023.7.1)
Requirement already satisfied: referencing>=0.28.4 in /Users/xingqitian/Libr
ary/Python/3.10/lib/python/site-packages (from jsonschema>=2.6->nbformat>=5.
7->nbconvert) (0.30.2)
Requirement already satisfied: rpds-py>=0.7.1 in /Users/xingqitian/Library/P
ython/3.10/lib/python/site-packages (from jsonschema>=2.6->nbformat>=5.7->nb
convert) (0.12.0)
Requirement already satisfied: entrypoints in /Users/xingqitian/Library/Pyth
on/3.10/lib/python/site-packages (from jupyter-client>=6.1.12->nbclient>=0.5
.0->nbconvert) (0.4)
Requirement already satisfied: nest-asyncio>=1.5.4 in /Users/xingqitian/Libr
ary/Python/3.10/lib/python/site-packages (from jupyter-client>=6.1.12->nbcli
ent>=0.5.0->nbconvert) (1.5.5)
Requirement already satisfied: python-dateutil>=2.8.2 in /Users/xingqitian/L
ibrary/Python/3.10/lib/python/site-packages (from jupyter-client>=6.1.12->nb
client>=0.5.0->nbconvert) (2.8.2)
Requirement already satisfied: pyzmq>=23.0 in /Users/xingqitian/Library/Pyth
on/3.10/lib/python/site-packages (from jupyter-client>=6.1.12->nbclient>=0.5
.0->nbconvert) (24.0.1)
Requirement already satisfied: tornado>=6.2 in /Users/xingqitian/Library/Pyt
hon/3.10/lib/python/site-packages (from jupyter-client>=6.1.12->nbclient>=0.
5.0->nbconvert) (6.2)
Note: you may need to restart the kernel to use updated packages.
```

In [11]:
```
# run in terminal
jupyter nbconvert /Users/xingqitian/Desktop/Python/Leetcode/Ch1_Array.ipynb
```

```
  Cell In [11], line 2
    jupyter nbconvert /Users/xingqitian/Desktop/Python/Leetcode/随想录Ch1数组
.ipynb --to python
            ^
SyntaxError: invalid syntax
```

In [ ]: