查找排序相关面试题

题目

给两个字符串s和t，判断t是否为s的重新排列后组成的单词

- s = "anagram", t = "nagaram", return true.
- s = "rat", t="car", return false.

sort

1. 转换成列表，sort排序(改变原始列表)，时间复杂度$O(nlogn)$

```
class Solution:
    def isAnagram(self, s, t):
        ss = list(s)
        tt = list(t)
        ss.sort()
        tt.sort()
        return ss == tt
```

sorted

2. 转换成列表，sorted排序（生成新列表），时间复杂度与上述类似，存在空间复杂度

```
class Solution:
    def isAnagram(self, s, t):
        return sorted(list(s)) == sorted(list(t))
```

dict

3. 转换为字典，计数，时间复杂度$O(n)$

```
class Solution:
    def isAnagram(self, s, t):
        dict1 = {} #类似('a':1,'b':2)
        dict2 = {}
        for ch in s:
            dict1[ch] = dict1.get(ch, 0) + 1
            #get：如果ch在dict的key里面，则获得对应value，否则建立key，赋值value为0
        for ch in t:
            dict2[ch] = dict2.get(ch, 0) + 1
        return dict1 == dict2
```

题目

给定一个m*n的二维列表查找一个数是否存在。列表有下列特性：

- 每一行的列表从左到右已经排序好。
- **每一行第一个数比上一行最后一个数大。**

线性查找

```
class Solution:
    def searchMatrix(self, matrix, target):
        for line in matrix: # O(m) m层
            if target in line: # O(n)
                return True
        return False
```

二分查找

```
from typing import List
```

```
class Solution:
    def searchMatrix(self, matrix: List[List[int]], target: int) -> bool:
        h = len(matrix) #长，几层
        if h == 0: # 注意边界条件:[]
            return False
        w = len(matrix[0]) #宽，几列
        if w == 0:
            return False # [[],[],[]] , 3层0列

        left = 0
        right = w * h - 1
        '''
        找下标：
        0 1 2 3
        4 5 6 7
        8 9 10 11
        i = num //4 (行); j = num % 4 (列)
        '''
        while left <= right:
            mid = (left + right) //2
            i = mid // w
            j = mid % w
            if matrix[i][j] == target:
                return True
            elif matrix[i][j] > target:
                right = mid - 1
            else:
                left = mid + 1
        else:
            return False
```

题目

给定一个列表和一个整数，设计算法找到两个数的下标，使得两个数之和为给定的整数。保证肯定仅有一个结果。

- 例如，列表[1,2,5,4]与目标整数3，1+2=3，结果为(0,1)。

两层for循环

```
class Solution:
    def twoSum(self, nums: List[int], target: int) -> List[int]:
        n = len(nums)
        for i in range(n):
            for j in range(i+1,n):
                if nums[i] + nums[j] == target:
                    return sorted([i,j])
```

根据一个数查找另一个数

```
class Solution:
    def twoSum(self, nums: List[int], target: int) -> List[int]:
        n = len(nums)
        for i in range(n):
            val = target - nums[i]
            print(val)
            if val in nums[i+1:n]:
                j = nums[i+1:n].index(val) + i + 1
                return sorted([i,j])
```

```
result = Solution()
result.twoSum([2,7,7,15],9)
```

```
7
```

```
[0, 1]
```

二分查找(前提：nums有序)

```
class Solution:
    def binary_search(self, li, left, right, val):
        while left <= right: #候选区有值
            mid = (left + right) // 2
            if li[mid] == val:
                return mid
            elif li[mid] > val:
                right = mid - 1
            else:
                left = mid + 1
        else:
            return None

    def twoSum(self, nums: List[int], target: int) -> List[int]:
        for i in range(len(nums)):
            a = nums[i]
            b = target - a
            if b >= a:
                j = self.binary_search(nums, i+1, len(nums)-1, b)
            else:
```

```
                j = self.binary_search(nums, 0, i-1, b)
            if j:
                break
        return sorted([i, j])
```

二分查找（无序）

新建二维列表：使用enumerate，时间复杂度：$O(nlogn)$

```
class Solution:
    def binary_search(self, li, left, right, val):
        while left <= right: #候选区有值
            mid = (left + right) // 2
            if li[mid][0] == val:
                return mid
            elif li[mid][0] > val:
                right = mid - 1
            else:
                left = mid + 1
        else:
            return None

    def twoSum(self, nums: List[int], target: int) -> List[int]:
        new_nums = [[num, i] for i, num in enumerate(nums)]
        new_nums.sort(key = lambda x:x[0])

        for i in range(len(new_nums)):
            a = new_nums[i][0]
            b = target - a
            if b >= a:
                j = self.binary_search(new_nums, i+1, len(new_nums)-1, b)
            else:
                j = self.binary_search(new_nums, 0, i-1, b)
            if j:
                break
        return sorted([new_nums[i][1], new_nums[j][1]])
```

```
nums = [1,5,2,4]
new_nums = [[num, i] for i, num in enumerate(nums)]
new_nums.sort(key = lambda x:x[0])
new_nums
```

```
[[1, 0], [2, 2], [4, 3], [5, 1]]
```

```
new_nums[2][0]
```

```
4
```

```
result = Solution()
result.twoSum([7,2,5,15],9)
```

```
[0, 1]
```