AI1804 算法设计与分析
第2次课上练习：算法设计与应用

练习说明：

- 本次练习共4道题，总时间90分钟（两节课）
- 每道题都需要用Python编程实现
- 重点关注算法设计与应用，而非数据结构实现
- 为每道题提供了代码框架，请在指定位置填写代码
- 每道题的main函数中包含测试用例，可以验证你的实现

问题2-1：基于数组的基础算法应用

子问题1-1：Two Sum问题

找到数组中和为目标值的两个数，返回它们的索引。

子问题1-2：最大子数组和

找到数组中连续子数组的最大和。

定义：连续子数组是指数组中相邻元素组成的子序列。 例如：数组[1, -3, 2, 1, -1]中，[2, 1]是连续子数组，[1, 2, -1]不是连续子数组。 目标：找到所有连续子数组中和最大的那个。

子问题1-3：寻找峰值元素

找到数组中的峰值元素（大于相邻元素的元素），返回其索引。

子问题1-4：多数元素

找到数组中出现次数大于n/2的元素。

代码框架：

```
def two_sum_sorted(arr, target):
    """找到数组中和为目标值的两个数，返回它们的索引"""
    # TODO: 实现Two Sum算法
    pass

def find_max_subarray_sum(arr):
    """找到数组中连续子数组的最大和"""
    # TODO: 实现最大子数组和算法
    pass

def find_peak_element(arr):
```

```
    """找到数组中的峰值元素，返回其索引"""
    # TODO: 实现峰值查找算法
    pass

def find_majority_element(arr):
    """找到数组中出现次数大于n/2的元素"""
    # TODO: 实现多数元素查找算法
    pass

# 测试代码
arr = [2, 7, 11, 15]
result = two_sum_sorted(arr, 9)
print(f"Two Sum结果: {result}")
```

问题2-2：集合操作实现

基于排序数组实现一个集合数据结构（元素具有唯一性，不能重复），支持以下操作：

- `insert(x)`：向集合中插入元素x
- `delete(x)`：从集合中删除元素x
- `find(x)`：查找元素x是否在集合中

代码框架：

```
class SortedArraySet:
    def __init__(self):
        """初始化集合"""
        self._data = []  # 存储数据的有序列表

    def _binary_search(self, x):
        """二分查找元素x的位置，返回(found, index)"""
        # TODO: 实现二分查找算法
        pass

    def insert(self, x):
        """向集合中插入元素x"""
        # TODO: 实现插入操作
        pass

    def delete(self, x):
        """从集合中删除元素x，返回True如果成功，False如果元素不存在"""
        # TODO: 实现删除操作
        pass

    def find(self, x):
        """查找元素x是否在集合中"""
        # TODO: 实现查找操作
        pass

    # 其他方法...

    def size(self):
        """返回集合大小"""
        return len(self._data)

    def to_list(self):
        """返回集合的有序列表表示"""
        return self._data.copy()

# 测试代码
s = SortedArraySet()
```

```
for x in [5, 2, 8, 1, 9, 3, 1, 2]:
    s.insert(x)
print(f"集合: {s.to_list()}")
```

问题2-3：归并排序的应用

实现归并排序算法，并用它解决逆序对统计问题。

逆序对是指数组中的两个元素，其中前面的元素大于后面的元素。 例如：数组[3, 1, 4, 2]中的逆序对有：(3,1), (3,2), (4,2)，共3个。

代码框架：

```
def merge_sort_with_inversion_count(arr):
    """
    归并排序并统计逆序对数量
    参数: arr - 待排序的数组
    返回: (sorted_arr, inversion_count) - 排序后的数组和逆序对数量
    """

    def merge(left, right):
        """合并两个有序数组，并统计跨越逆序对"""
        # TODO: 实现合并逻辑，同时统计逆序对
        pass

    def merge_sort_helper(arr):
        """递归实现归并排序并统计逆序对"""
        # TODO: 实现归并排序的递归逻辑
        pass

    return merge_sort_helper(arr)

def count_inversions_naive(arr):
    """
    朴素方法统计逆序对（用于验证结果）
    时间复杂度: O(n^2)
    """
    count = 0
    n = len(arr)
    for i in range(n):
        for j in range(i + 1, n):
            if arr[i] > arr[j]:
                count += 1
    return count

# 测试代码
arr = [3, 1, 4, 2]
sorted_arr, inv_count = merge_sort_with_inversion_count(arr)
print(f"排序后: {sorted_arr}, 逆序对数量: {inv_count}")
```

问题2-4：医院诊室分配系统

问题背景： 医院有k个诊室，n个病人挂号（各自有号码1到n）。病人按到达顺序形成链表，但到达顺序与号码顺序不一致。需要将病人分配到k个诊室，使得各诊室负载均衡，且每个诊室内病人按号码从小到大排序。最后，所有诊室的病人合并成一个有序队列去取药。

任务要求： 实现一个完整的HospitalSystem类，核心方法包括：

- assign_patients_to_clinics()：分配病人到诊室（负载均衡+有序维护）
- merge_clinic_queues()：使用分治法合并k个诊室队列
- process_hospital_queue()：综合运用所有技术的主流程

代码框架：

```
class Patient:
    """病人类"""
    def __init__(self, patient_id, name):
        self.patient_id = patient_id  # 挂号号码
        self.name = name

    def __str__(self):
        return f"病人{self.patient_id}({self.name})"

class PatientNode:
    """病人链表节点"""
    def __init__(self, patient, next=None):
        self.patient = patient
        self.next = next

class HospitalSystem:
    """医院诊室分配系统"""
    def __init__(self, num_clinics):
        self.num_clinics = num_clinics
        self.clinics = [[] for _ in range(num_clinics)]  # k个诊室（数组）
        self.final_queue = []  # 最终取药队列

    def assign_patients_to_clinics(self, patient_list_head):
        """将病人分配到各个诊室，要求负载均衡且每个诊室内按号码有序"""
        # TODO: 实现病人分配算法
        pass

    def merge_clinic_queues(self):
        """合并所有诊室的队列为最终取药队列，要求使用分治法"""
        # TODO: 实现k路归并算法
        pass

    def process_hospital_queue(self, patient_list_head):
        """主处理流程 - 综合运用所有技术（已实现，展示整体流程）"""
```

```
        print("开始医院排队系统处理...")

        # 步骤1: 统计病人信息（Linked List遍历）
        print("步骤1: 统计到达病人")
        total_patients = self._count_patients(patient_list_head)
        print(f"   总病人数: {total_patients}")

        # 步骤2: 分配病人到诊室（Linked List + Array + Sorting）
        print("步骤2: 分配病人到诊室（负载均衡+有序维护）")
        self.assign_patients_to_clinics(patient_list_head)

        # 显示分配结果
        for i, clinic in enumerate(self.clinics):
            patient_ids = [p.patient_id for p in clinic]
            print(f"   诊室{i+1}: {len(clinic)}人, 号码: {patient_ids}")

        # 步骤3: 合并诊室队列（Divide & Conquer + Recursion）
        print("步骤3: 合并诊室队列（k路归并）")
        self.merge_clinic_queues()

        final_ids = [p.patient_id for p in self.final_queue]
        print(f"   最终取药队列: {final_ids}")

        print("处理完成！")
        return {
            'total_patients': total_patients,
            'clinic_loads': [len(clinic) for clinic in self.clinics],
            'final_queue_size': len(self.final_queue),
            'is_final_sorted': all(self.final_queue[i].patient_id <=
                                   self.final_queue[i+1].patient_id
                                   for i in range(len(self.final_queue)-1))
        }

    def _count_patients(self, head):
        """统计病人总数（已实现的辅助方法）"""
        count = 0
        current = head
        while current:
            count += 1
            current = current.next
        return count

# 测试代码
# 创建病人到达序列（号码有序但到达顺序随机）
patients_data = [(5,"王五"), (2,"李二"), (8,"赵八"), (1,"张一"), (7,"孙七")]

# 构建链表
head = PatientNode(Patient(patients_data[0][0], patients_data[0][1]))
```

```
current = head
for patient_id, name in patients_data[1:]:
    current.next = PatientNode(Patient(patient_id, name))
    current = current.next

# 创建3个诊室的医院系统
hospital = HospitalSystem(3)

# 执行处理流程
report = hospital.process_hospital_queue(head)
print(f"处理结果: {report}")
```

提交要求：

- 完成所有TODO部分的代码实现
- 确保所有测试用例都能正确运行
- 重点关注算法的时间复杂度分析
- 理解每个算法的核心思想和适用场景
- 代码要有适当的注释说明关键逻辑

学习目标：

- 掌握基于数组的基础算法设计技巧（双指针、分治、递归）
- 理解二分查找在各种场景中的应用
- 深入理解分治思想在数组问题中的应用
- 学会将基础算法应用到实际问题中
- 培养算法复杂度分析能力

实际应用价值：

- 双指针技巧：数组处理、数据匹配
- 二分查找：数据库索引、搜索优化
- 分治算法：大数据处理、并行计算
- 排序算法：数据预处理、优化搜索
- 递归思想：问题分解、算法设计