插入排序刷题路线

```
// 插入排序主循环：从第二个元素开始遍历（索引1到n-1）
for (int i = 1; i < n; i++) {
    // 当前待插入的元素值
    int key = nums[i];
    // j指向当前元素的前一个位置（已排序部分的末尾）
    int j = i - 1;

    // 从后向前遍历已排序部分，找到key应该插入的位置
    // 条件：j不越界且nums[j] > key（需要将大于key的元素后移）
    while (j >= 0 && nums[j] > key) {
        // 将大于key的元素向后移动一位，为key腾出空间
        nums[j + 1] = nums[j];
        // 向前移动指针，继续比较前一个元素
        j--;
    }
    // 找到正确位置（j+1），将key插入到该位置
    nums[j + 1] = key;
}
```

力扣

排序数组

给你一个整数数组 `nums`，请你将该数组升序排列。

你必须在 不使用任何内置函数 的情况下解决问题，时间复杂度为 $O(n \log n)$，并且空间复杂度尽可能小。

示例 1：

- 输入：`nums = [5, 2, 3, 1]`
- 输出：`[1, 2, 3, 5]`
- 解释：数组排序后，某些数字的位置没有改变（例如，2 和 3），而其他数字的位置发生了改变（例如，1 和 5）。

示例 2：

- 输入：`nums = [5, 1, 1, 2, 0, 0]`
- 输出：`[0, 0, 1, 1, 2, 5]`
- 解释：请注意，`nums` 的值不一定唯一。

提示：

- $1 \leq \text{nums.length} \leq 5 \times 10^4$
- $-5 \times 10^4 \leq \text{nums[i]} \leq 5 \times 10^4$

```
class Solution {
public:
    vector<int> sortArray(vector<int>& nums) {
        int n = nums.size();
        for (int i = 1; i < n; i++) {
            int key = nums[i];
            int j = i - 1;
```

```
            while (j >= 0 && nums[j] > key) {
                nums[j + 1] = nums[j];
                j--;
            }
            nums[j + 1] = key;
        }
        return nums;
    }
};
```

对链表进行插入排序

题目描述： 给定单个链表的头 `head` ，使用 插入排序 对链表进行排序，并返回 *排序后链表的头* 。

插入排序 算法的步骤：

1. 插入排序是迭代的，每次只移动一个元素，直到所有元素可以形成一个有序的输出列表。
2. 每次迭代中，插入排序只从输入数据中移除一个待排序的元素，找到它在序列中适当的位置，并将其插入。
3. 重复直到所有输入数据插入完为止。

下面是插入排序算法的一个图形示例。部分排序的列表(黑色)最初只包含列表中的第一个元素。每次迭代时，从输入数据中删除一个元素(红色)，并就地插入已排序的列表中。

对链表进行插入排序。

```
链表 4->2->1->3
[4]->2->1->3
[2]->[4]->1->3
[1]->[2]->[4]->3
[1]->[2]->[3]->[4]
链表 1->2->3->4
```

示例 1： [图示：输入链表 4->2->1->3，输出链表 1->2->3->4]

- 输入： `head = [4,2,1,3]`
- 输出： `[1,2,3,4]`

示例 2： [图示：输入链表 -1->5->3->4->0，输出链表 -1->0->3->4->5]

- 输入： `head = [-1,5,3,4,0]`
- 输出： `[-1,0,3,4,5]`

提示：

- 列表中的节点数在 `[1, 5000]` 范围内
- `-5000 <= Node.val <= 5000`

```
class Solution {
public:
    ListNode* insertionSortList(ListNode* head) {
        // 边界情况：如果链表为空或只有一个节点，直接返回，无需排序
        if(head == nullptr || head->next == nullptr){
            return head;
        }

        // 创建哑节点(dummy node)，简化头节点的插入操作
        // 哑节点的值不重要，它的next指向原始链表的头节点
        ListNode* dummy = new ListNode(0);
```

```
        dummy->next = head;

        // 初始化指针：lastSorted指向已排序部分的最后一个节点
        // curr指向当前待排序的节点（从第二个节点开始）
        ListNode* lastSorted = head;
        ListNode* curr = head->next;

        // 遍历链表，对每个节点进行插入排序
        while(curr != nullptr){
            // 情况1：当前节点的值 >= 已排序部分的最后一个节点的值
            // 说明当前节点应该位于已排序部分的末尾，直接扩展已排序部分
            if(lastSorted->val <= curr->val){
                lastSorted = lastSorted->next;  // 将lastSorted向后移动一位
            }
            // 情况2：当前节点的值 < 已排序部分的最后一个节点的值
            // 需要在已排序部分中找到正确的插入位置
            else{
                // 从头开始（哑节点开始）寻找插入位置
                // 找到第一个值大于当前节点值的前一个节点
                ListNode* prev = dummy;
                // 当prev->next的值 <= curr->val时，继续向后移动prev
                while (prev->next->val <= curr->val) {
                    prev = prev->next;
                }
                // 此时prev->next的值大于curr->val，所以curr应该插入到prev之后

                // 步骤1：将curr从当前位置移除
                lastSorted->next = curr->next;  // 跳过curr节点

                // 步骤2：将curr插入到prev之后
                curr->next = prev->next;  // curr指向prev原来的下一个节点
                prev->next = curr;        // prev指向curr，完成插入

                // 注意：lastSorted不需要移动，因为curr被移到了前面
                // 但已排序部分的末尾仍然是lastSorted（它指向下一个待处理节点）
            }
            // 更新curr为已排序部分的下一个待排序节点
            curr = lastSorted->next;
        }
        // 返回排序后的链表（哑节点的下一个节点）
        return dummy->next;
    }
};
```