# 题目

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。

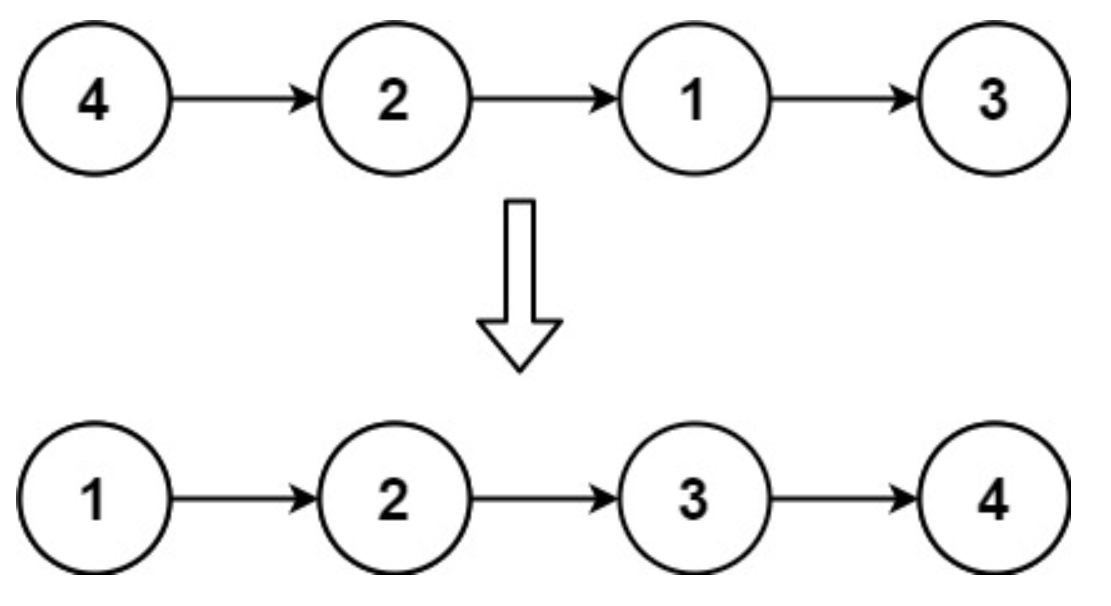
3、重复直到所有输入数据插入完为止。

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。



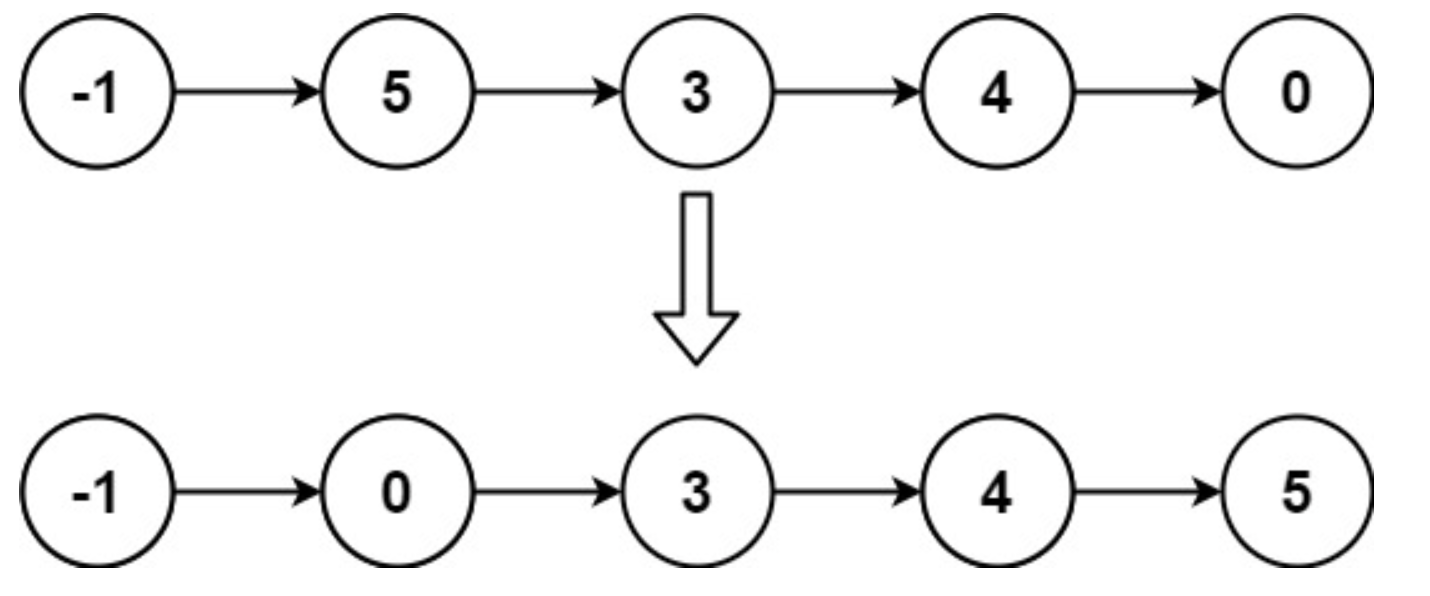
示例 1：



输入: head = [4,2,1,3]

输出: [1,2,3,4]

示例 2：



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

# 分析

## 方法一：插入排序

思路：

插入排序的思想是逐个将元素插入到已经排好序的部分中，直到整个序列都有序。在链表中，我们可以维护一个已排序部分的头节点 `dummy`，初始时指向空，然后遍历原链表的节点，将每个节点插入到已排序部分的合适位置。

具体步骤如下：

1、创建一个虚拟头节点 dummy，指向空。

2、遍历原链表，对于每个节点：

- 从 dummy 开始遍历已排序部分，找到第一个比当前节点大的节点 prev，将当前节点插入到 prev 后面。

- 如果没有找到比当前节点大的节点，说明当前节点应该插入到已排序部分的末尾，直接插入到最后即可。

3、返回 dummy->next，即为排序后的链表头节点。

以下是一个可能的实现：

class Solution {

public:

ListNode\* insertionSortList(ListNode\* head) {

if (!head || !head->next) {

return head;

}

ListNode\* dummy = new ListNode(0); // 虚拟头节点

ListNode\* cur = head; // 当前待插入节点

while (cur) {

ListNode\* prev = dummy;

ListNode\* next = cur->next; // 保存下一个待插入节点

// 找到第一个大于当前节点的节点

while (prev->next && prev->next->val < cur->val) {

prev = prev->next;

}

// 插入当前节点到已排序部分

cur->next = prev->next;

prev->next = cur;

cur = next; // 继续处理下一个待插入节点

}

return dummy->next;

}

};

这个实现中，我们维护了一个虚拟头节点 `dummy`，使得插入操作变得简单。对于每个待插入节点 `cur`，我们从虚拟头节点开始遍历已排序部分，找到合适的位置插入。