对链表进行插入排序。

插入排序的动画演示如上。从第一个元素开始，该链表可以被认为已经部分排序（用黑色表示）。
每次迭代时，从输入数据中移除一个元素（用红色表示），并原地将其插入到已排好序的链表中。

插入排序算法：

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。

示例 1：

输入： `4->2->1->3`

输出： `1->2->3->4`

示例 2：

输入： `-1->5->3->4->0`

输出： `-1->0->3->4->5`

对链表进行插入排序的具体过程如下。

首先判断给定的链表是否为空，若为空，则不需要进行排序，直接返回。

创建哑节点 dummyHead，令 dummyHead.next = head。引入哑节点是为了便于在 head 节点之前插入节点。

维护 lastSorted 为链表的已排序部分的最后一个节点，初始时 lastSorted = head。

维护 curr 为待插入的元素，初始时 curr = head.next。

比较 lastSorted 和 curr 的节点值。

若 lastSorted.val <= curr.val，说明 curr 应该位于 lastSorted 之后，将 lastSorted 后移一位，curr 变成新的 lastSorted。

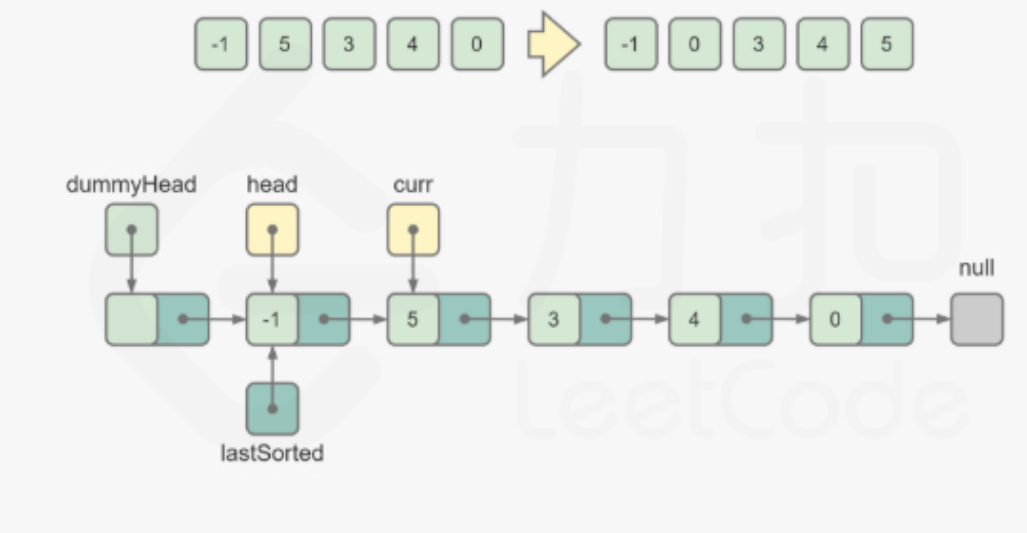
否则，从链表的头节点开始往后遍历链表中的节点，寻找插入 curr 的位置。令 prev 为插入 curr 的位置的前一个节点，进行如下操作，完成对 curr 的插入：

lastSorted.next = curr.next
curr.next = prev.next
prev.next = curr

令 curr = lastSorted.next，此时 curr 为下一个待插入的元素。

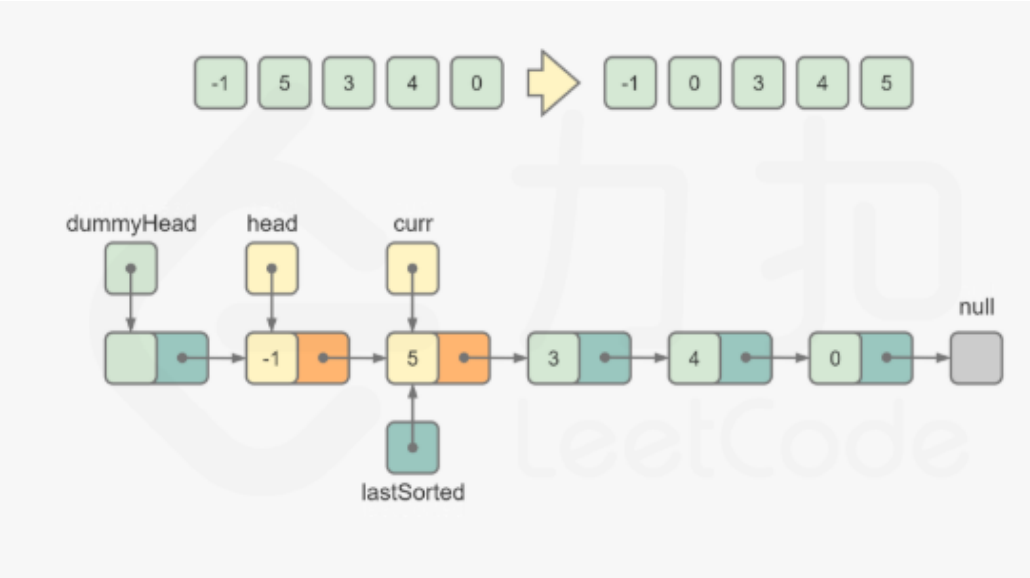
重复第 5 步和第 6 步，直到 curr 变成空，排序结束。

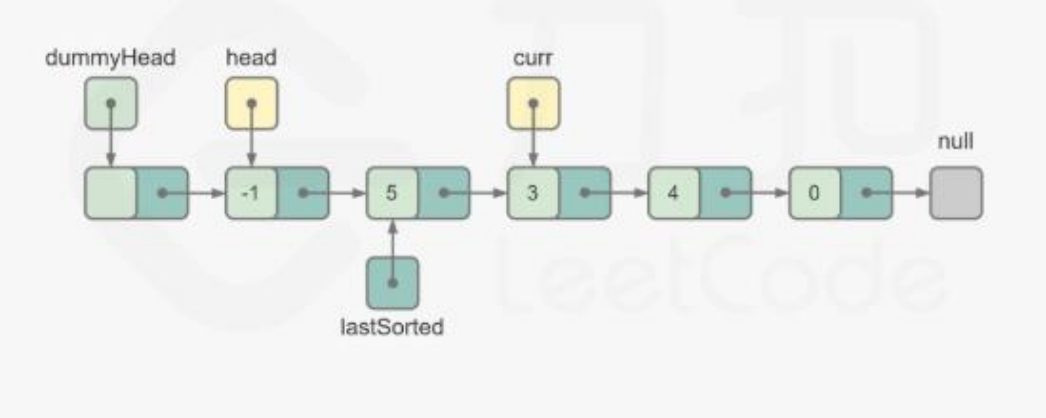
返回 dummyHead.next，为排序后的链表的头节点。




```
struct ListNode* insertionSortList(struct ListNode* head){
    if(head==NULL)
        return NULL;
    struct ListNode* tmp=(struct ListNode*)malloc(sizeof(struct ListNode));
    tmp->val=0;
    tmp->next=head;
    struct ListNode* lastsorted=head;
    struct ListNode* cur=head->next;
    while(cur)
    {
        if(lastsorted->val<=cur->val)//如果维护的排序满足排序要求，直接后移，不
需要调整
        {
            lastsorted=lastsorted->next;
        }
        else//调整直到合适的位置处
        {
            struct ListNode* prev=tmp;
            while(prev->next->val<=cur->val)//找到插入节点的位置
            {
                prev=prev->next;
```

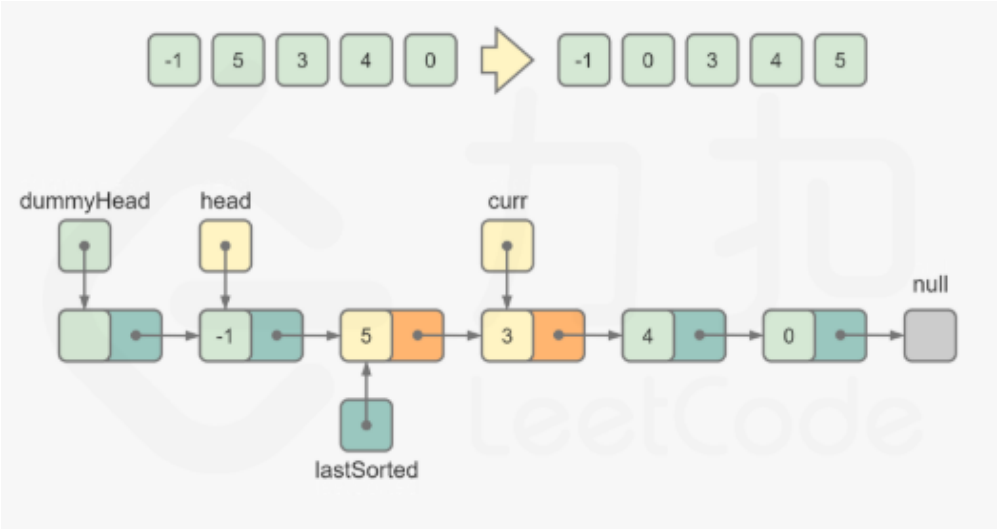
```
            }
            //此时 prev 位于插入位置的前一节点
            //开始调整
            lastsorted->next=cur->next;
            cur->next=prev->next;
            prev->next=cur;
        }
        cur=lastsorted->next;
    }
    return tmp->next;
}
```

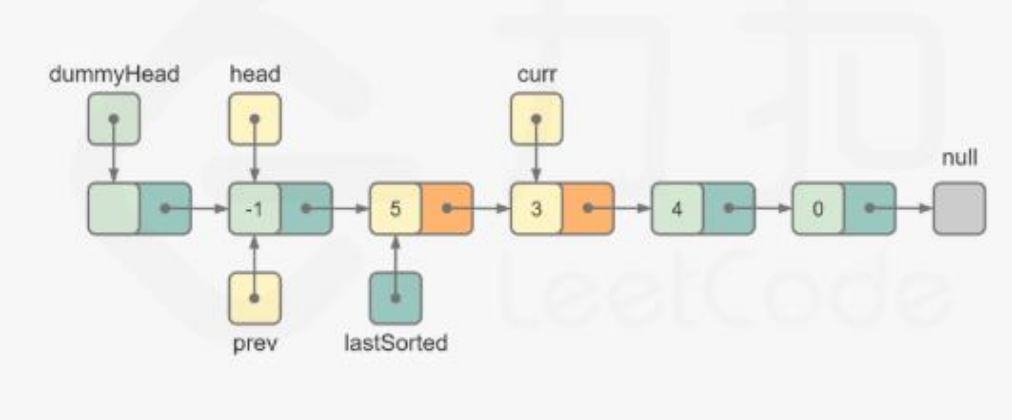


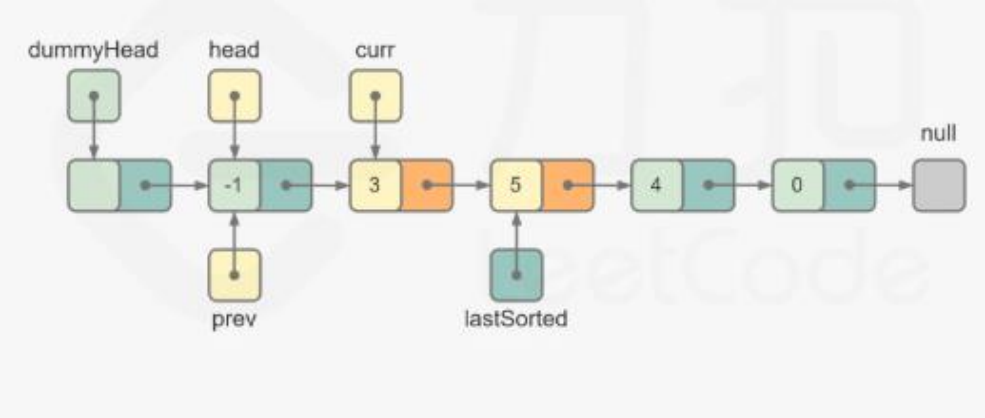


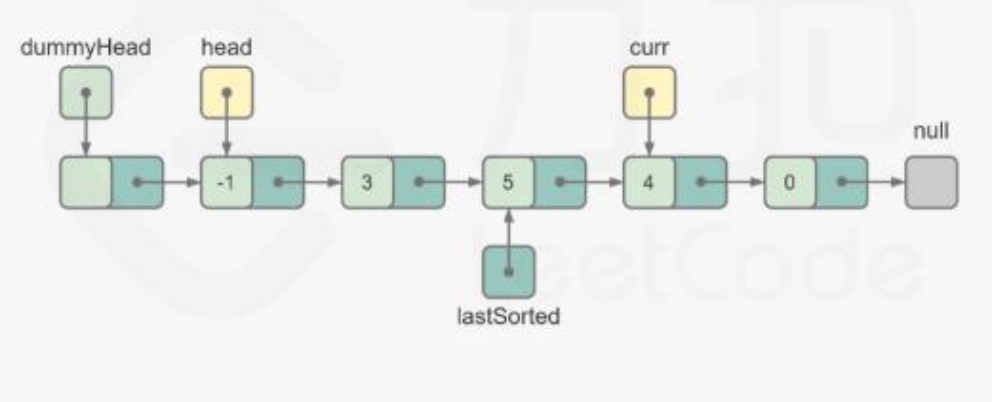

```
struct ListNode* insertionSortList(struct ListNode* head){
    if(head==NULL)
        return NULL;
    struct ListNode* tmp=(struct ListNode*)malloc(sizeof(struct ListNode));
    tmp->val=0;
    tmp->next=head;
    struct ListNode* lastsorted=head;
```

```
    struct ListNode* cur=head->next;
    while(cur)
    {
        if(lastsorted->val<=cur->val)//如果维护的排序满足排序要求，直接后移，不
需要调整
        {
            lastsorted=lastsorted->next;
        }
        else//调整直到合适的位置处
        {
            struct ListNode* prev=tmp;
            while(prev->next->val<=cur->val)//找到插入节点的位置
            {
                prev=prev->next;
            }
            //此时 prev 位于插入位置的前一节点
            //开始调整
            lastsorted->next=cur->next;
            cur->next=prev->next;
            prev->next=cur;
        }
        cur=lastsorted->next;
    }
    return tmp->next;
}
```










```
struct ListNode* insertionSortList(struct ListNode* head){
    if(head==NULL)
        return NULL;
    struct ListNode* tmp=(struct ListNode*)malloc(sizeof(struct ListNode));
    tmp->val=0;
    tmp->next=head;
    struct ListNode* lastsorted=head;
    struct ListNode* cur=head->next;
    while(cur)
    {
        if(lastsorted->val<=cur->val)//如果维护的排序满足排序要求，直接后移，不
需要调整
```

```
        {
            lastsorted=lastsorted->next;
        }
        else//调整直到合适的位置处
        {
            struct ListNode* prev=tmp;
            while(prev->next->val<=cur->val)//找到插入节点的位置
            {
                prev=prev->next;
            }
            //此时 prev 位于插入位置的前一节点
            //开始调整
            lastsorted->next=cur->next;
            cur->next=prev->next;
            prev->next=cur;
        }
        cur=lastsorted->next;
    }
    return tmp->next;
}
```