1. Insertion Sort List

Sort a linked list using insertion sort.

 A graphical example of insertion sort. The partial sorted list (black) initially contains only the first element in the list. With each iteration one element (red) is removed from the input data and inserted in-place into the sorted list

**Algorithm of Insertion Sort:**

1. Insertion sort iterates, consuming one input element each repetition, and growing a sorted output list.
2. At each iteration, insertion sort removes one element from the input data, finds the location it belongs within the sorted list, and inserts it there.
3. It repeats until no input elements remain.

**Example 1:**

Input: 4->2->1->3  
Output: 1->2->3->4

**Example 2:**

Input: -1->5->3->4->0  
Output: -1->0->3->4->5

**解** 插入排序

/\*\*  
 \* Definition for singly-linked list.  
 \* struct ListNode {  
 \* int val;  
 \* ListNode \*next;  
 \* ListNode() : val(0), next(nullptr) {}  
 \* ListNode(int x) : val(x), next(nullptr) {}  
 \* ListNode(int x, ListNode \*next) : val(x), next(next) {}  
 \* };  
 \*/  
class Solution {  
public:  
 ListNode\* insertionSortList(ListNode\* head) {  
 if(head == NULL || head->next == NULL)return head;  
 ListNode \*hh = new ListNode(INT\_MIN, head);  
 ListNode \*q = head->next, \*p = head;  
 while(q){  
 ListNode \*tmp = hh->next, \*pre = hh;  
 while(tmp != q && tmp->val < q->val){  
 pre = tmp;  
 tmp = tmp->next;  
 }  
 if(tmp != q){  
 p->next = q->next;  
 q->next = tmp;  
 pre->next = q;  
 q = p->next;  
 }else{  
 p = p->next;  
 q = q->next;  
 }  
 }  
 return hh->next;  
 }  
};