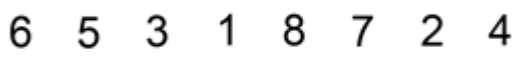
147. Insertion Sort List

Given the `head` of a singly linked list, sort the list using **insertion sort**, and return *the sorted list's head*.

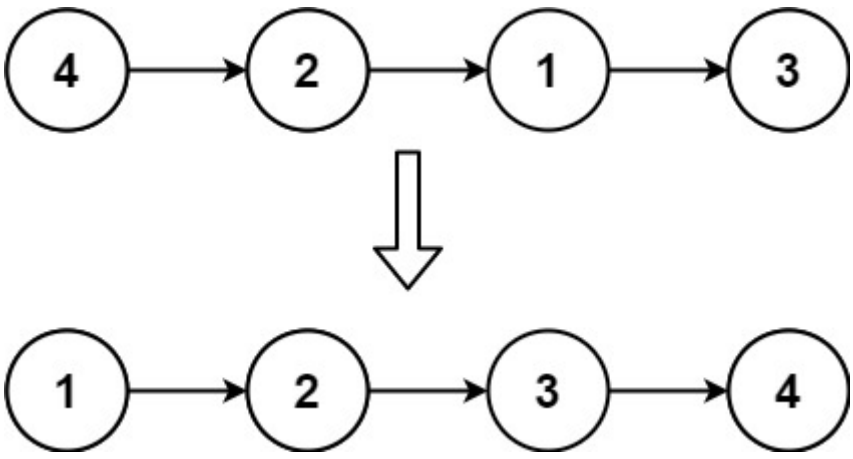
The steps of the **insertion sort** algorithm:

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.

The following is a graphical example of the insertion sort algorithm. The partially sorted list (black) initially contains only the first element in the list. One element (red) is removed from the input data and inserted in-place into the sorted list with each iteration.

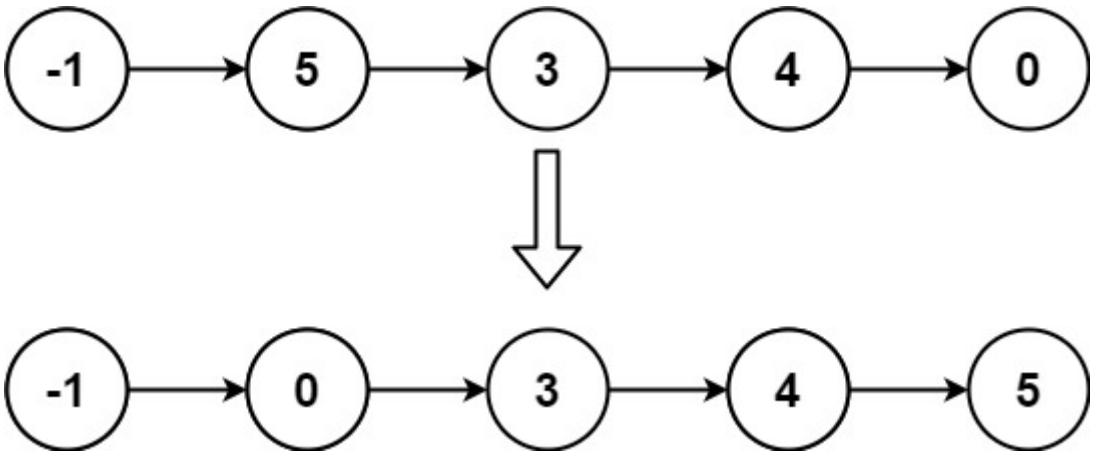


Example 1:




```
Input: head = [4,2,1,3]
Output: [1,2,3,4]
```

Example 2:




```
Input: head = [-1,5,3,4,0]
Output: [-1,0,3,4,5]
```

Constraints:

- The number of nodes in the list is in the range `[1, 5000]`.
- `-5000 <= Node.val <= 5000`

```
class Solution:
    def insertionSortList(self, head: Optional[ListNode]) ->
Optional[ListNode]:
        if not head or not head.next:
            return head

        # Use dummy_head will help us to handle insertion before head easily
        dummy_head = ListNode(val=-5000, next=head)
        last_sorted = head # last node of the sorted part
        cur = head.next # cur is always the next node of last_sorted
        while cur:
            if cur.val >= last_sorted.val:
                last_sorted = last_sorted.next
            else:
                # Search for the position to insert
                prev = dummy_head
                while prev.next.val <= cur.val:
                    prev = prev.next

                # Insert
                last_sorted.next = cur.next
                cur.next = prev.next
                prev.next = cur

            cur = last_sorted.next
```

```
        return dummy_head.next
```