50. . 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

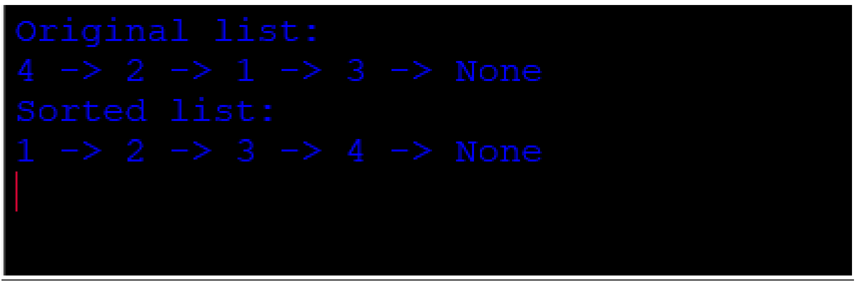
list (black) initially contains only the first element in the list. One element (red) is removed

from the input data and inserted in-place into the sorted list with each iteration.

Code:

```python
class ListNode:
    def __init__(self, val=0, next=None):
        self.val = val
        self.next = next

def insertionSortList(head):
    dummy = ListNode(0)
    current = head

    while current:
        prev_node = dummy
        next_node = dummy.next
        while next_node:
            if next_node.val > current.val:
                break
            prev_node = next_node
            next_node = next_node.next

        temp = current.next
        current.next = next_node
        prev_node.next = current
        current = temp

    return dummy.next
def print_list(head):
    current = head
    while current:
        print(current.val, end=" -> ")
        current = current.next
    print("None")
node1 = ListNode(4)
node2 = ListNode(2)
node3 = ListNode(1)
node4 = ListNode(3)
node1.next = node2
node2.next = node3
node3.next = node4

print("Original list:")
print_list(node1)

sorted_head = insertionSortList(node1)
print("Sorted list:")
print_list(sorted_head)
```

Output:



```
Original list:
4 -> 2 -> 1 -> 3 -> None
Sorted list:
1 -> 2 -> 3 -> 4 -> None
```

Time Complexity:

- $T(n)= O(n^2)$