

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.

AIM: To find the Insertion sort For a linked list

PROGRAM:

```
class ListNode:
```

```
    def __init__(self, val=0, next=None):
```

```
        self.val = val
```

```
        self.next = next
```

```
def insertionSortList(head):
```

```
    dummy = ListNode(0)
```

```
    dummy.next = head
```

```
    curr = head
```

```
    while curr and curr.next:
```

```
        if curr.val > curr.next.val:
```

```
            prev = dummy
```

```
            while prev.next.val < curr.next.val:
```

```
                prev = prev.next
```

```
            temp = curr.next
```

```
            curr.next = temp.next
```

```
            temp.next = prev.next
```

```
            prev.next = temp
```

```
        else:
```

```
curr = curr.next
```

```
return dummy.next
```

```
def printLinkedList(head):
```

```
    while head:
```

```
        print(head.val, end=" ")
```

```
        head = head.next
```

```
    print()
```

```
head = ListNode(4)
```

```
head.next = ListNode(2)
```

```
head.next.next = ListNode(1)
```

```
head.next.next.next = ListNode(3)
```

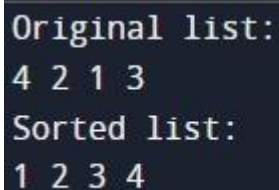
```
print("Original list:")
```

```
printLinkedList(head)
```

```
sorted_head = insertionSortList(head)
```

```
print("Sorted list:")
```

```
printLinkedList(sorted_head)
```



```
Original list:  
4 2 1 3  
Sorted list:  
1 2 3 4
```

OUTPUT:

TIME COMPLEXITY: $O(n^2)$