Problem

Sort List

Sort a linked list in $O(n \log n)$ time using constant space complexity.

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
```

Solution

这道题主要考察的是单链表的排序问题，看到题目对于时间复杂度的要求，可以想到基本排序算法中的快速排序和归并排序（这两个算法都是基于分治策略的排序算法），实际上，对于单链表而言，使用归并排序在绝大多数情况可以获得更好的性能

GitHub传送门

快速排序

```
class Solution {
private:
    ListNode* partition(ListNode* key,ListNode* end){
        auto p1=key;
        auto p2 = key->next;
        while(p2!=end){
            if(p2->val<key->val){
                p1 = p1->next;
                auto temp = p1->val;
                p1->val = p2->val;
                p2->val = temp;
            }
            p2 = p2->next;
        }
        if(p1!=key){
            auto temp=key->val;
            key->val=p1->val;
            p1->val = temp;
        }
        return p1;
    }
    void quickSort(ListNode* head,ListNode* end){
        if(head!=end){
            auto node = partition(head, end);
            quickSort(head, node);
            quickSort(node->next, end);
        }
    }

public:
    ListNode* sortList(ListNode* head) {
        quickSort(head, nullptr);
        return head;
    }
};
```

归并排序

```
class Solution
{
  private:
    ListNode *getMiddle(ListNode *head)
    {
        if (head == nullptr)
            return head;
        auto slow = head, quick = head;
        while (quick->next != nullptr && quick->next=->next != nullptr)
        {
            slow = slow->next;
            quick = quick->next->next;
        }
        return slow;
    }

    ListNode *mergeList(ListNode *left, ListNode *right)
    {
        ListNode head(0);
        auto headPtr = &head;
        while (left != nullptr && right != nullptr)
        {
            if (left->val < right->val)
            {
                headPtr->next = left;
                left = left->next;
            }
            else
            {
                headPtr->next = right;
                right = right->next;
            }
            headPtr = headPtr->next;
        }
        headPtr->next = left != nullptr ? left : right;
        return head.next;
    }

    ListNode* mergeSort(ListNode* head){
        if(head==nullptr||head->next==nullptr)
            return head;
        auto mid = getMiddle(head);
        auto sortedPost = mergeSort(mid->next);
        mid->next = nullptr;
        auto sortedPre = mergeSort(head);
        return mergeList(sortedPre, sortedPost);
    }

  public:
    ListNode *sortList(ListNode *head)
    {
        if (head == nullptr)
            return head;
        return mergeSort(head);
```

```
    }
};
```