class Solution {

public:

ListNode\* sortList(ListNode\* head) {

if (head == nullptr || head->next == nullptr) {

return head; // Already sorted or empty

}

// 1. Split the list into two halves

ListNode\* mid = getMid(head);

ListNode\* left = head;

ListNode\* right = mid->next;

mid->next = nullptr; // Break the link

// 2. Recursively sort the two halves

left = sortList(left);

right = sortList(right);

// 3. Merge the sorted halves

return merge(left, right);

}

private:

ListNode\* getMid(ListNode\* head) {

ListNode\* slow = head;

ListNode\* fast = head;

while (fast->next != nullptr && fast->next->next != nullptr) {

slow = slow->next;

fast = fast->next->next;

}

return slow;

}

ListNode\* merge(ListNode\* left, ListNode\* right) {

ListNode\* dummy = new ListNode(0);

ListNode\* current = dummy;

while (left != nullptr && right != nullptr) {

if (left->val <= right->val) {

current->next = left;

left = left->next;

} else {

current->next = right;

right = right->next;

}

current = current->next;

}

// Add any remaining nodes

if (left != nullptr) {

current->next = left;

}

if (right != nullptr) {

current->next = right;

}

ListNode\* sortedHead = dummy->next;

delete dummy;

return sortedHead;

}

};