MergeSort in C++

```
#include <iostream>

using namespace std;

// Definition for a singly-linked list node
struct ListNode {
    int data;
    ListNode* next;

    ListNode(int x) {
        data = x;
        next = nullptr;
    }
};

// Function to merge two sorted linked lists
ListNode* merge(ListNode* h1, ListNode* h2) {
    if (h1 == nullptr) return h2;
    if (h2 == nullptr) return h1;

    ListNode* ans = nullptr;
    ListNode* t = nullptr;

    if (h1->data < h2->data) {
        ans = h1;
        t = h1;
        h1 = h1->next;
    } else {
        ans = h2;
        t = h2;
        h2 = h2->next;
    }

    while (h1 != nullptr && h2 != nullptr) {
        if (h1->data < h2->data) {
            t->next = h1;
            t = t->next;
            h1 = h1->next;
        } else {
            t->next = h2;
            t = t->next;
            h2 = h2->next;
        }
    }

    if (h1 != nullptr) t->next = h1;
    if (h2 != nullptr) t->next = h2;

    return ans;
}

// Function to find the middle of the linked list
ListNode* mid(ListNode* h) {
    ListNode* slow = h;
    ListNode* fast = h;

    while (fast != nullptr && fast->next != nullptr) {
        slow = slow->next;
        fast = fast->next->next;
    }
```

Dry Run — Function Calls Breakdown:

1. Initial Call:

```
mergeSort(4 -> 2 -> 1 -> 3)
```

Midpoint = 1 (list breaks into):

- h1 = 4 -> 2
- h2 = 1 -> 3

2. Recursive Breakdown:

Level	Call	Mid Node	Left Part	Right Part
1	mergeSort(4->2->1->3)	1	4->2	1->3
2	mergeSort(4->2)	2	4	2
2	mergeSort(1->3)	3	1	3

3. Merge Steps (Bottom-Up):

Step	Merge Call	Output
1	merge(4, 2)	2 -> 4
2	merge(1, 3)	1 -> 3
3	merge(2->4, 1->3)	1 -> 2 -> 3 -> 4

✓ Final Output:

```
Sorted Linked List: 1 -> 2 -> 3 -> 4
```

```
    return slow;
}

// Function to perform merge sort on the linked list
ListNode* mergeSort(ListNode* h1) {
    if (h1 == nullptr || h1->next == nullptr) return h1;

    ListNode* m = mid(h1);
    ListNode* h2 = m->next;
    m->next = nullptr;

    ListNode* t1 = mergeSort(h1);
    ListNode* t2 = mergeSort(h2);
    ListNode* t3 = merge(t1, t2);

    return t3;
}

// Function to print the linked list
void printList(ListNode* head) {
    ListNode* temp = head;
    while (temp != nullptr) {
        cout << temp->data << " ";
        temp = temp->next;
    }
    cout << endl;
}

int main() {
    // Creating an example linked list: 4 -> 2 -> 1 -> 3
    ListNode* head = new ListNode(4);
    head->next = new ListNode(2);
    head->next->next = new ListNode(1);
    head->next->next->next = new ListNode(3);

    cout << "Original Linked List:" << endl;
    printList(head);

    head = mergeSort(head);

    cout << "Sorted Linked List:" << endl;
    printList(head);

    // Clean up allocated memory
    ListNode* current = head;
    while (current != nullptr) {
        ListNode* next = current->next;
        delete current;
        current = next;
    }

    return 0;
}
```

Output:-

```
0
```