Merge Sort for Linked List

Given Pointer/Reference to the head of the linked list, the task is to **Sort the given linked list using Merge Sort**.
Note: If the length of linked list is odd, then the extra node should go in the first list while splitting.

Example 1:

```
Input: N = 5
value[]  = {3,5,2,4,1}
Output: 1 2 3 4 5 Explanation: After sorting the given
linked list, the resultant matrix
will be 1->2->3->4->5.
```

Example 2:

```
Input: N = 3
value[]  = {9,15,0}
Output: 0 9 15 Explanation: After sorting the given
linked list , resultant will be
0->9->15.
```

Your Task:
For C++ and Python: The task is to complete the function **mergeSort**() which sort the linked list using merge sort function.
For Java: The task is to complete the function **mergeSort**() and return the node which can be used to print the sorted linked list.

Expected Time Complexity: O(N*Log(N))
Expected Auxiliary Space: O(N)

Constraints:
1 <= T <= 100
1 <= N <= 10^5

```python
class Solution:
    #Function to sort the given linked list using Merge Sort.
    def mergeSort(self, head):
        if head is None or head.next is None:
            return head
        mid = self.middleOfLL(head)
        nextHead = mid.next
        mid.next = None
```

```
        first = self.mergeSort(head)
        second = self.mergeSort(nextHead)
        return self.merge2SortedLists(first,second)




    def middleOfLL(self,head):
        slow = head
        fast = head

        while fast.next is not None and fast.next.next is not None:
            slow = slow.next
            fast = fast.next.next
        return slow


    def merge2SortedLists(self,head1,head2):
        if head1 is None or head2 is None:
            return head1 if head2 is None else head2
        dummyNode = Node(-1)
        l1 = head1
        l2 = head2
        prev = dummyNode

        while l1!=None and l2!=None:
            if l1.data>l2.data:
                prev.next = l2
                l2 = l2.next
            else:
                prev.next = l1
                l1 = l1.next
            prev = prev.next
        if l1 is  None:
            prev.next = l2
        else:
            prev.next = l1
        return dummyNode.next
```