Sort linked list which is already sorted on absolute values

Given a linked list which is sorted based on absolute values. Sort the list based on actual values.

Examples:

```
Input :  1 -> -10
output: -10 -> 1

Input : 1 -> -2 -> -3 -> 4 -> -5
output: -5 -> -3 -> -2 -> 1 -> 4

Input : -5 -> -10
Output: -10 -> -5

Input : 5 -> 10
output: 5 -> 10
```

Source : Amazon Interview

A simple solution is to traverse the linked list from beginning to end. For every visited node, check if it is out of order. If it is, remove it from its current position and insert at correct position. This is implementation of insertion sort for linked list and time complexity of this solution is O(n*n).

A better solution is to sort the linked list using merge sort. Time complexity of this solution is O(n Log n).

An efficient solution can work in O(n) time. An important observation is, all negative elements are present in reverse order. So we traverse the list, whenever we find an element that is out of order, we move it to the front of linked list.

Below is C++ implementation of above idea.

C++

```
// C++ program to sort a linked list, already
// sorted by absolute values
#include <bits/stdc++.h>
using namespace std;

// Linked List Node
struct Node
{
    Node* next;
    int data;
};

// Utility function to insert a node at the
// beginning
void push(Node** head, int data)
{
    Node* newNode = new Node;
    newNode->next = (*head);
    newNode->data = data;
    (*head) = newNode;
}

// Utility function to print a linked list
void printList(Node* head)
{
    while (head != NULL)
    {
        cout << head->data;
        if (head->next != NULL)
            cout << " -> ";
```

```
        head = head->next;
    }
    cout<<endl;
}

// To sort a linked list by actual values.
// The list is assumed to be sorted by absolute
// values.
void sortList(Node** head)
{
    // Initialize previous and current nodes
    Node* prev = (*head);
    Node* curr = (*head)->next;

    // Traverse list
    while (curr != NULL)
    {
        // If curr is smaller than prev, then
        // it must be moved to head
        if (curr->data < prev->data)
        {
            // Detach curr from linked list
            prev->next = curr->next;

            // Move current node to beginning
            curr->next = (*head);
            (*head) = curr;

            // Update current
            curr = prev;
        }

        // Nothing to do if current element
        // is at right place
        else
            prev = curr;

        // Move current
        curr = curr->next;
    }
}

// Driver code
int main()
{
    Node* head = NULL;
    push(&head, -5);
    push(&head, 5);
    push(&head, 4);
    push(&head, 3);
    push(&head, -2);
    push(&head, 1);
    push(&head, 0);

    cout << "Original list :\n";
    printList(head);

    sortList(&head);

    cout << "\nSorted list :\n";
    printList(head);

    return 0;
}
```

Java

```
// Java program to sort a linked list, already
// sorted by absolute values
class SortList
```

```
{
    static Node head;  // head of list

    /* Linked list Node*/
    static class Node
    {
        int data;
        Node next;
        Node(int d) {data = d; next = null; }
    }

 // To sort a linked list by actual values.
        // The list is assumed to be sorted by absolute
        // values.
 Node sortedList(Node head)
 {
  // Initialize previous and current nodes
  Node prev = head;
  Node curr = head.next;

  // Traverse list
  while(curr != null)
  {
   // If curr is smaller than prev, then
                        // it must be moved to head
   if(curr.data < prev.data)
   {
    // Detach curr from linked list
    prev.next = curr.next;

    // Move current node to beginning
    curr.next = head;
    head = curr;

    // Update current
    curr = prev;
   }

   // Nothing to do if current element
                        // is at right place
   else
   prev = curr;

   // Move current
   curr = curr.next;
  }
  return head;
 }

    /* Inserts a new Node at front of the list. */
    public void push(int new_data)
    {
        /* 1 & 2: Allocate the Node &
                  Put in the data*/
        Node new_node = new Node(new_data);

        /* 3. Make next of new Node as head */
        new_node.next = head;

        /* 4. Move the head to point to new Node */
        head = new_node;
    }

    /* Function to print linked list */
    void printList(Node head)
    {
        Node temp = head;
        while (temp != null)
        {
           System.out.print(temp.data+" ");
           temp = temp.next;
        }
        System.out.println();
```

```
    }

    /* Driver program to test above functions */
    public static void main(String args[])
    {
        SortList llist = new SortList();

        /* Constructed Linked List is 1->2->3->4->5->6->
           7->8->8->9->null */
        llist.push(-5);
        llist.push(5);
        llist.push(4);
        llist.push(3);
        llist.push(-2);
        llist.push(1);
        llist.push(0);

        System.out.println("Original List :");
        llist.printList(llist.head);

        llist.head = llist.sortedList(head);

        System.out.println("Sorted list :");
        llist.printList(llist.head);
    }
}

// This code has been contributed by Amit Khandelwal(Amit Khandelwal 1).
```

Output:

```
Original list :
0 -> 1 -> -2 -> 3 -> 4 -> 5 -> -5

Sorted list :
-5 -> -2 -> 0 -> 1 -> 3 -> 4 -> 5
```