Flattening a Linked List

Given a linked list where every node represents a linked list and contains two pointers of its type:
(i) Pointer to next node in the main list (we call it 'right' pointer in below code)
(ii) Pointer to a linked list where this node is head (we call it 'down' pointer in below code).
All linked lists are sorted. See the following example

```
5 -> 10 -> 19 -> 28
|    |     |     |
V    V     V     V
7    20    22    35
|          |     |
V          V     V
8          50    40
|                |
V                V
30               45
```

Write a function flatten() to flatten the lists into a single linked list. The flattened linked list should also be sorted. For example, for the above input list, output list should be 5->7->8->10->19->20->22->28->30->35->40->45->50.

Recommended: Please solve it on "*PRACTICE*" first, before moving on to the solution.

The idea is to use Merge() process of merge sort for linked lists. We use merge() to merge lists one by one. We recursively merge() the current list with already flattened list.
The down pointer is used to link nodes of the flattened list.

Following are C and Java implementations.

C/C++ Java

```c
// C program for flattening a linked list
#include <stdio.h>
#include <stdlib.h>

// A Linked List Node
typedef struct Node
{
    int data;
    struct Node *right;
    struct Node *down;
} Node;

/* A utility function to insert a new node at the begining
   of linked list */
void push (Node** head_ref, int new_data)
{
    /* allocate node */
    Node* new_node = (Node *) malloc(sizeof(Node));
    new_node->right = NULL;

    /* put in the data  */
    new_node->data  = new_data;

    /* link the old list off the new node */
    new_node->down = (*head_ref);

    /* move the head to point to the new node */
    (*head_ref)    = new_node;
}

/* Function to print nodes in the flattened linked list */
void printList(Node *node)
{
    while (node != NULL)
    {
        printf("%d ", node->data);
        node = node->down;
    }
}
```

```
// A utility function to merge two sorted linked lists
Node* merge( Node* a, Node* b )
{
    // If first list is empty, the second list is result
    if (a == NULL)
        return b;

    // If second list is empty, the second list is result
    if (b == NULL)
        return a;

    // Compare the data members of head nodes of both lists
    // and put the smaller one in result
    Node* result;
    if (a->data < b->data)
    {
        result = a;
        result->down = merge( a->down, b );
    }
    else
    {
        result = b;
        result->down = merge( a, b->down );
    }

    return result;
}
```

```
// The main function that flattens a given linked list
Node* flatten (Node* root)
{
    // Base cases
    if (root == NULL || root->right == NULL)
        return root;

    // Merge this list with the list on right side
    return merge( root, flatten(root->right) );
}
```

```
// Driver program to test above functions
int main()
{
    Node* root = NULL;

    /* Let us create the following linked list
       5 -> 10 -> 19 -> 28
       |    |     |     |
       V    V     V     V
       7    20    22    35
       |          |     |
       V          V     V
       8          50    40
       |                |
       V                V
       30               45
    */
    push( &root, 30 );
    push( &root, 8 );
    push( &root, 7 );
    push( &root, 5 );

    push( &( root->right ), 20 );
    push( &( root->right ), 10 );

    push( &( root->right->right ), 50 );
    push( &( root->right->right ), 22 );
    push( &( root->right->right ), 19 );

    push( &( root->right->right->right ), 45 );
    push( &( root->right->right->right ), 40 );
    push( &( root->right->right->right ), 35 );
    push( &( root->right->right->right ), 20 );

    // Let us flatten the list
    root = flatten(root);

    // Let us print the flatened linked list
    printList(root);

    return 0;
}
```

Output:

```
5 7 8 10 19 20 20 22 30 35 40 45 50
```