Jon Snow and Soldiers

Solution Approach:

In this problem the main crux was based on the concept of merging two sorted linked lists into one single sorted linked list. The 2 lines of soldiers were to be stored in the form of a linked list separately and then they were to be merged in a sorted manner

A modification to the naïve approach was that there were following requirements:

T(n): O(n)

S(n): O(1) [in place merging of the linked lists]

Approach:

- Start traversing the lists from the start
- Check whether the 1st node of the 2nd list lies between the first two nodes of the 1st list if so then insert it in the middle and make the 2nd node of the 2nd list as the head.
- Continue the above step until you reach the end of both list
- In case 1st list is exhausted before the 2nd list then just join the next node of the 2nd list to the end node of the 1st list .i.e. just join the remnant nodes of the list

Solution:

Java

```
import java.util.*;


public class Merge_LL
{
    static class Node
    {
        int data;
        Node next;
        Node(int d) {
            data = d;
            next = null;
        }
    }
    Node head;

    public static void main(String args[])
    {

         Scanner sc = new Scanner(System.in);
         int t=sc.nextInt();

         while(t>0)
         {
```

```
            Node head1 = new Node(sc.nextInt());
            Node tail1 = head1;
            while(sc.hasNextInt())
            {
                int input = sc.nextInt();
                if(input==-1){
                    tail1.next=null;
                    break;
                }
                tail1.next = new Node(input);
                tail1 = tail1.next;
            }
            if(head1.data==-1){
                head1=null;
            }

            Node head2 = new Node(sc.nextInt());
            Node tail2 = head2;
            while(sc.hasNextInt())
            {
                int input = sc.nextInt();
                if(input==-1){
                    tail2.next=null;
                    break;
                }
                tail2.next = new Node(input);
                tail2 = tail2.next;
            }
            if(head2.data==-1){
                head2=null;
            }

            Node head = merge(head1, head2);

            printList(head);

            t--;

         }
    }

   public static void printList(Node head)
    {

        while (head!= null)
        {
           System.out.print(head.data+" ");
           head = head.next;
        }
        System.out.print("-1");
        System.out.println();
    }

    static Node mergeUtil(Node h1, Node h2)
    {

        if (h1.next == null) {
            h1.next = h2;
```

```
            return h1;
        }

        Node curr1 = h1, next1 = h1.next;
        Node curr2 = h2, next2 = h2.next;

        while (next1 != null && curr2 != null) {

            if ((curr2.data) >= (curr1.data) && (curr2.data) <=
(next1.data)) {
                next2 = curr2.next;
                curr1.next = curr2;
                curr2.next = next1;

                curr1 = curr2;
                curr2 = next2;
            }
            else {

                if (next1.next != null) {
                    next1 = next1.next;
                    curr1 = curr1.next;
                }

                else {
                    next1.next = curr2;
                    return h1;
                }
            }
        }
        return h1;
    }

    static Node merge(Node h1, Node h2)
    {
        if (h1 == null)
            return h2;
        if (h2 == null)
            return h1;

        if (h1.data < h2.data)
            return mergeUtil(h1, h2);
        else
            return mergeUtil(h2, h1);
    }

}
```

CPP

```
#include <bits/stdc++.h>

using namespace std;


struct Node {
```

```
	int data;
	struct Node* next;
};
struct Node* newNode(int key)
{
	struct Node* temp = new Node;
	temp->data = key;
	temp->next = NULL;
	return temp;
}
void printList(struct Node* node)
{
	while (node != NULL) {
		printf("%d ", node->data);
		node = node->next;
	}
}
struct Node* mergeUtil(struct Node* h1,
				struct Node* h2)
{

	if (!h1->next) {
		h1->next = h2;
		return h1;
	}
	struct Node *curr1 = h1, *next1 = h1->next;
	struct Node *curr2 = h2, *next2 = h2->next;

	while (next1 && curr2) {

		if ((curr2->data) >= (curr1->data) && (curr2->data) <=
(next1->data)) {
			next2 = curr2->next;
			curr1->next = curr2;
```

```
                curr2->next = next1;
                curr1 = curr2;
                curr2 = next2;
            }
            else {
                if (next1->next) {
                    next1 = next1->next;
                    curr1 = curr1->next;
                }
                else {
                    next1->next = curr2;
                    return h1;
                }
            }
        }
        return h1;
}
struct Node* merge(struct Node* h1,
                        struct Node* h2)
{
        if (!h1)
            return h2;
        if (!h2)
            return h1;
        if (h1->data < h2->data)
            return mergeUtil(h1, h2);
        else
            return mergeUtil(h2, h1);
}
```