# Finding the Difference of two linked list

Author : Clifford Imhomoh

## Problem

Suppose you are given two sorted lists of integers. The difference of the first list with respect to the second list is the set of all eleements that are on the first list but not the second,in the same order as the original list. For example :

L1 : 4→7→12→18→25→30→42
L2 : 4→13→18→24→30
RESULT: 7→12→25→42

Implement a method that takes in two linked list of integers and returns their difference (Keep in mind that each list does not have duplicate values so no need to account for that)

## Algorithm

The main thing to keep in mind when designing this algorithm is that we are only adding elements from the first list to our resulting list since we only care about the elements that appear in the first list and not the second list. Second thing to think about is how do we know when to add an element from the first list. More specifically how do we know for sure that the current element in the first list does not appear in the second list. Well to answer this question, lets look at our lists assuming we are using two pointers.

Note that first list is L1 and second list is L2

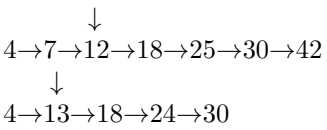
↓
4→7→12→18→25→30→42
↓
4→13→18→24→30

So what do we do in the case when both elements are equal ? Well in the case when both elements are equal, we should increment both pointers since we don't care about elements that are common between the lists. Hence we have case 1

↓
4→7→12→18→25→30→42
↓
4→13→18→24→30

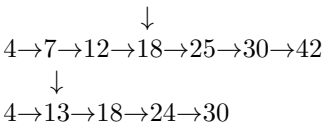
Result: 7→/

What do we do in the case when the element in the first list is smaller than the element in the second list ? In this case we add the element in the first list to the resulting list. Why? Well since we know both lists are sorted, we know that it is impossible to find 7 at a later point in the second list. So we can safely add 7 to our resulting list. We increment the pointer for the first list and leave the pointer for the second list. Hence we have case 2



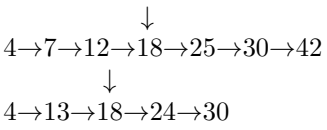

Result: 7→12→/

This condition falls into case 2 (:



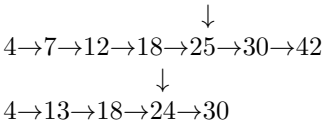

Result: 7→12→/

So now we come upon a new case. What do we do when the current element in the second list is less than the current element in the first list. Well in this case we just increment the pointer for the second list. Why? remember we are looking for the elements in the first list that do not appear in the second list. This case just means that 13 appears in the second list and not the first. We do not care about that. Hence we have case 3 (The final case!)



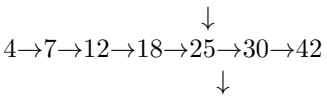

Result: 7→12→/

This case falls into case 1



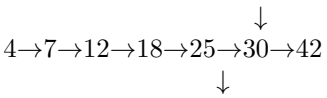
Result: 7→12→/

This falls into case 3



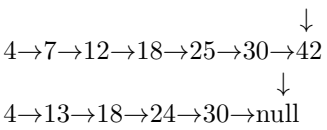

Result: 7→12→25→/

This falls into case 2




Result: 7→12→25→/

This falls into case 1




Result: 7→12→25→/

Now we get have a special (edge) case when the second list is shorter than the first (Meaning that the first ptr is not null but the second ptr is now null) What do we do in this case ? In this case we add the rest of the elements on the first list to our result list since we know that they won't appear on the second list. So our resulting list becomes //

Result: 7→12→25→42→/

## Method signature and Node class

```
public class Node {
    int data;
    Node next;
    public Node (int data, Node next) {
      this.data = data;
      this.next = next;
```

```
 }

}

public static Node difference (Node l1, Node l2) {

if ( l2 == null ) {
    return l1; If l2 is null then the difference is l1
   }

Node front;    These pointers are for our resulting list
Node last;    Last pointer: Insert at rear to maintain sorted order

Node ptr1 = l1;    These pointers are for going through our lists
Node ptr2 = l2;

while ( ptr1 != null && ptr2 != null ) {

      if ( ptr1.data == ptr2.data ) { Case 1: Both equals
         ptr1 = ptr1.next;
         ptr2 = ptr2.next;
       }

      else if ( ptr1.data > ptr2.data ) { Case 3: l1 data > l2 data
         ptr2 = ptr2.next;
        }

      else { Case 2: l1.data < l2.data; We add l1.data to list

            Node tmp = new Node(ptr1.data,null); Create the node

            if ( front == null ) {This is the first node we are adding
               front = tmp;
               last = tmp;
              }

            else { Not the first node we are adding, just insert at rear
               last.next = tmp;
               last = tmp;
             }
            ptr1 = ptr1.next We only increment ptr1 in this case
        }
  } End of main while loop

Check to see if there are more elements in l1 and add them
while (ptr1 != null) {
```

```
            Node tmp2 = new Node(ptr.data,null)
            if ( front == null ) {This is the first node we are adding
              front = tmp2;
              last = tmp2;
             }

            else { Not the first node we are adding, just insert at rear
              last.next = tmp2;
              last = tmp2;
            }
            ptr1 = ptr1.next;
          } End of second while loop
return front; return front of the new list, and tada we are done!
}
```