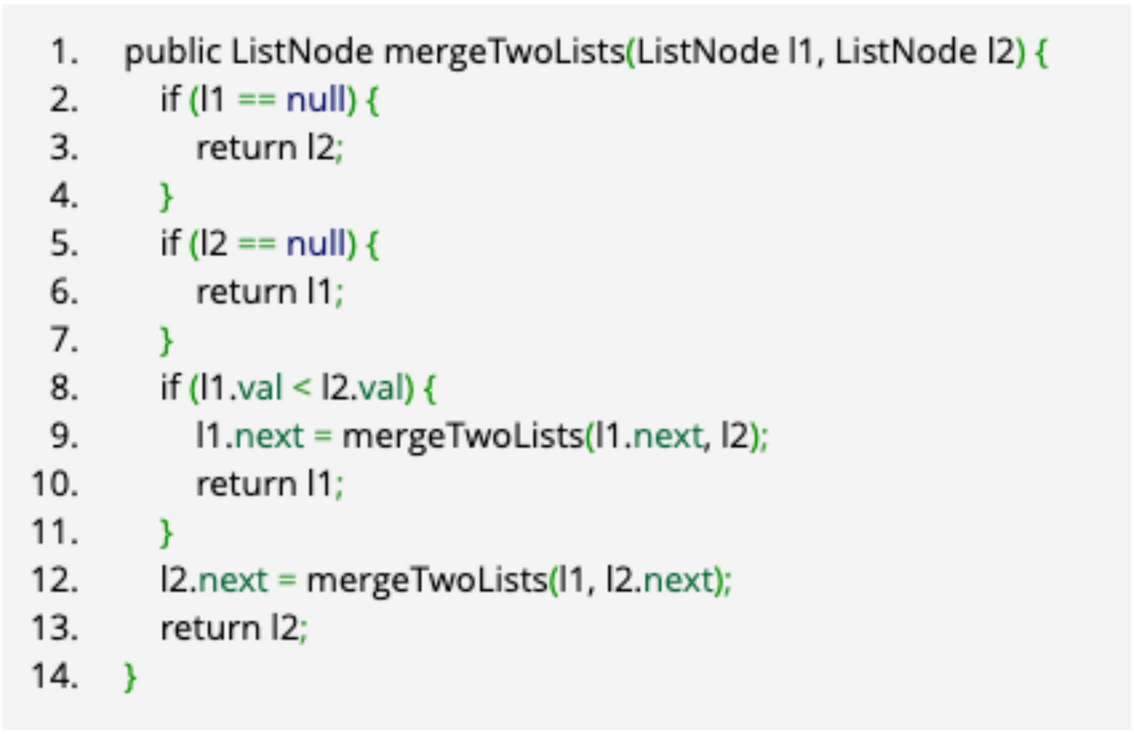
The following method is given the heads of two *sorted* linked lists l1 and l2 and merges them into one sorted list, returning the head of the merged linked list.



```
 1.   public ListNode mergeTwoLists(ListNode l1, ListNode l2) {
 2.       if (l1 == null) {
 3.           return l2;
 4.       }
 5.       if (l2 == null) {
 6.           return l1;
 7.       }
 8.       if (l1.val < l2.val) {
 9.           l1.next = mergeTwoLists(l1.next, l2);
10.           return l1;
11.       }
12.       l2.next = mergeTwoLists(l1, l2.next);
13.       return l2;
14.   }
```

Time analysis:
We will perform a worst-case analysis of this algorithm using the asymptotic notation. Here the operations executed by the algorithm depend on the length of the two input linked lists. We will assume each list is of length N.

Each call to this method involves constant time operations in lines 2-8, 10, 13. Executing some or all of these lines once takes O(1).

If one of the lists is empty (base case of the recursion) the method will return.
However if both lists are non-empty, the method will perform a single recursive call to the same method, either in line 9, or line 12.

In order to calculate the runtime of the method we need to calculate the number of recursive calls **in the worst case** before a base case is reached.Let this number be X. The WC running time of the function is then T(N) = O(1)*O(X).

We thus need to calculate the worst case for X.

[Possible alternative justifications follow; just one of these two justifications (or similar) would suffice]

- In a worst case input, a recursive call from line 9 is followed by a recursive call from line 12, and vice-versa, until a base case is reached. Each call decreases the length of one list by one, thus there will be (in the worst case) O(2*N)=O(N) recursive calls, which gives T(N)=O(N).

(note to students: worst case input example: lists contain [1,3,5,7,9,11,…2N+1] and [2,4,6,8,10,12,…,2N]).

- In another worst case input, all recursive calls are made from line 9. In this case we will have O(N) recursive calls, also leading to T(N)=O(N). The same will be true if all recursive calls are from line 12. Also the same will be true if there is an arbitrary interleaving of calls from lines 9 and 12, as at most there can only be 2*O(N) recursive calls.

Space analysis:

As there are O(N) recursive calls before all return (in the worst, but also in all cases), the call stack will require O(N) memory. Since the code does not allocate any new objects, the heap requirement is O(1). Thus the total memory requirement is O(N)+O(1) = O(N).