

### QUESTION 1

This definition of `sumlist/2` is not tail recursive:

```
sumlist([], 0).
sumlist([N|Ns], Sum) :-
    sumlist(Ns, Sum0),
    Sum is N + Sum0.
```

Rewrite it to be tail recursive.

### QUESTION 2

Given a binary tree defined to satisfy the predicate `tree/1`

```
tree(empty).
tree(node(Left,_,Right)) :-
    tree(Left),
    tree(Right).
```

write a predicate `tree_list(Tree, List)` that holds when `List` is a list of all the elements of `Tree`, in left-to-right order. This code need only work in the mode where the tree is input.

### QUESTION 3

Revise the definition from the previous question not to use `append/3` to construct the list. That is, ensure the cost of the operation is proportional to the number of elements in the tree.

Hint: look at the approach taken to write a tail recursive reverse predicate for inspiration.

### QUESTION 4

Write a predicate `list_tree(List, Tree)` that holds when `Tree` is a balanced tree whose elements are the elements of `List`, in the order they appear in `List`. This need only work in the mode where `List` is a proper list.

Hint: First divide the list into the first half, the middle element, and the last half, then recursively construct a tree from the first half and second half, and assemble these into the resulting tree.

Clarification: The main intended use of this predicate is to build a balanced tree from a list. It should, as far as reasonable, work in other modes. One complication is that it's possible for there to be different trees, balanced in different ways, with the same nodes in order. You don't need to handle this possibility here (though it would make an interesting challenge exercise). You just need to handle balanced trees that could have been produced by this predicate.