A little summary on the notation for recursive algorithms

Step 1: Define the header

Your algorithm has a name and it has variables. In the specification, we don't care about the types of the variables, you just write the name. The convention is L for lists, T for arrays, and small letters for numbers. For lists, you usually want to denote the first element in a special way, you call it the head and write it h; if you want to refer to the first two nodes, then n1 and n2. Note that in a list, we use '.' To express a link: n1.n2.L means n1 followed by n2 followed by L (remainder of the list).

Algorithm show taking a list:	show(h.L)
Algorithm compare taking two lists:	compare(h1.L1, h2.L2)
Algorithm isSorted taking a list:	isSorted(n1.n2.L)
Algorithm sum taking an array:	sum(T)
Algorithm add taking two numbers:	add(a,b)

Step 2: Initializing extra variables in the header

When the user calls the algorithm, he just gives you what he needs. You may create extra variables, for example to keep track of something between the steps. So, your first algorithm is empty and all it does is to call the real one by initializing it.

Algorithm sum taking a List: it needs a variable for the sum.

sum(L) : sum(L, 0)	*I call the real algorithms and initialize total to 0…*
sum(h.L, total):	*Now I specify the algorithm, and by 'h' I refer to the head.*

Step 3: General case

Express the processing that you do in the general case. If you want the sum of a List, the general case means that there is a head, and the action is to go to the next element and add the data in the head to the total:

$$h \neq \varnothing \rightarrow \text{sum}(L, \text{total} + h)$$

If you want to compare two lists L1 and L2 and count the number of elements in which they differ, then:

$$h1 \neq \varnothing \wedge h2 \neq \varnothing \wedge h1 \neq h2 \rightarrow \text{diff}(L1, L2, \text{total}+1)$$
$$h1 \neq \varnothing \wedge h2 \neq \varnothing \wedge h1 = h2 \rightarrow \text{diff}(L1, L2, \text{total})$$

This means "if both lists have a head and the elements are different, then keep processing and say that one more element differs; if both lists have a head and the elements are the same, then keep processing". Use ^ for "and" (if you want several conditions altogether) and v for "or".

Step 4: When to stop

If you reach the end of your list, then the head is empty. You may have to return a result, but in any case you should stop. For example, to return the sum of the elements in the list:

$$h = \varnothing \rightarrow \text{total}$$

If you have two counters c1 and c2 and you want to return "true" when c1 appears more often than c2, and false otherwise, then:

$$h = \varnothing \wedge c1 > c2 \rightarrow \text{true}$$
$$h = \varnothing \wedge c1 \leq c2 \rightarrow \text{false}$$

Step 5: Check it

Firstly, check that it's written correctly: all the cases must be disjoint, so you cannot satisfy two at the same time. Secondly, check that it actually works: take a very small example, and process it.

Examples

Sum of the elements in a List. First version: the result is thrown outside.
sum(h.L):
$h = \varnothing \rightarrow 0$
$h \neq \varnothing \rightarrow h + sum(L)$

Sum of the elements in a List. Second version: the result is produced inside with a variable.
sum(L): sum(L, 0)
sum(h.L, total):
$h = \varnothing \rightarrow total$
$h \neq \varnothing \rightarrow sum(L, total+h)$

Returning the list itself. First version: the result is thrown outside.
identical(h.L):
$h = \varnothing \rightarrow \varnothing$
$h \neq \varnothing \rightarrow h.identical(L)$

Returning the list itself. Second version: the result is produced inside with a variable.
identical(L1): identital(L1, Ø)
identical(L1, L2):
$h = \varnothing \rightarrow L2$
$h \neq \varnothing \rightarrow identical(L1, L2.h)$

Concatenating two lists. First version: the result is thrown outside.
concat(h1.L1, h2.L2):
$h1 = \varnothing \wedge h2 \neq \varnothing \quad \rightarrow h2.concat(\varnothing, L2)$
$h1 = \varnothing \wedge h2 = \varnothing \quad \rightarrow \varnothing$
$h1 \neq \varnothing \quad \rightarrow h1.concat(L1, h2.L2)$

Concatenating two lists. Second version: the result is produced inside with a variable.
concat(L1, L2): concat(L1, L2, Ø)
concat(h1.L1, h2.L2, L):
$h1 = \varnothing \wedge h2 \neq \varnothing \quad \rightarrow concat(\varnothing, L2, L.h2)$
$h1 = \varnothing \wedge h2 = \varnothing \quad \rightarrow L$
$h1 \neq \varnothing \quad \rightarrow h1.concat(L1, h2.L2, L.h1)$

You can throw the result outside when it does not need further processing: the user will get what he asked for directly. If you need some processing, then it is better to keep the result going in variables and look at these variables when you reach the base case. For example, if you have a list of 0s and 1s and you want to know which appears more often, then keep track of what you've seen in a variable and analyze it in the base case to return true or false.