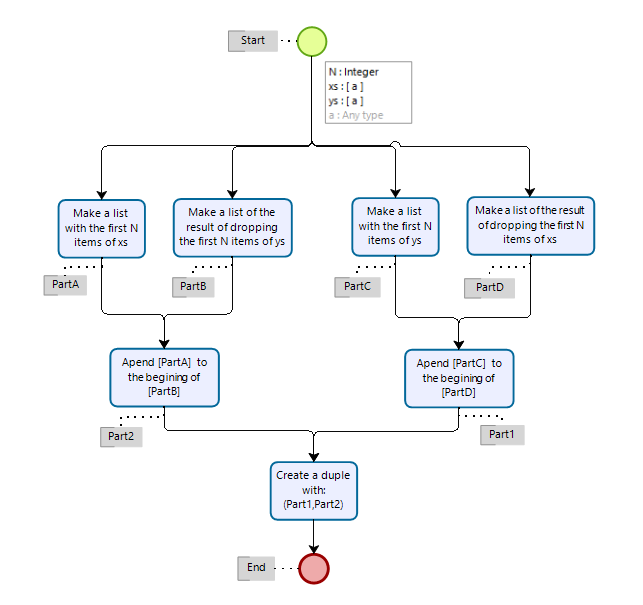
**ALGORITHMS ANALYSIS**

1. **Algorithm #1**
   1. **Flow diagram**



* 1. **Big O complexity**

Since this analysis is intended to be *apriori*, I will be using pseudo code to describe the algorithm for further interpretation.

Function1 ( number n, list1, list2 ) {

X1 = Concatenate ( get\_first\_n\_items ( n, list\_1) , get\_all\_but\_first\_n\_items ( n, list\_2) )

X2 = Concatenate ( get\_first\_n\_items ( n, list\_2) , get\_all\_but\_first\_n\_items ( n, list\_1) )

Return (X2,X1)

}

The functions in blue will have to go through the first n items of the lists to be able to perform their respective purposes (for each item they either push it to a new list or drop it to have the remaining items, depending of which of the two blue functions we are talking about), from this we can infer that these functions take a time of n to execute (f(n) = n).

The function concatenate could be done by going through one of the lists in other to concatenate the other one, but here we will assume that it doesn’t need to go through any of the lists and just takes a time of 1.

Assignations and return statement take times of 1 only so, we end up with something like this:

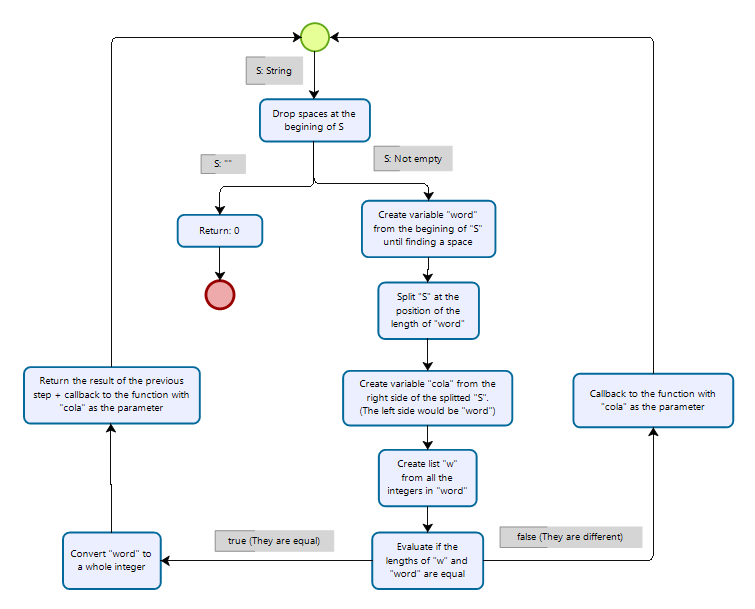
F(n) = 4n + c

4 being the times we call either of the functions in blue, c being the sum of the other operations that are negligible for the finding of the big O. The big O would be:

O(f(n)) = O(n)

1. **Algorithm #2**

**2.1 Flow Diagram**

****

**2.2 Big O complexity**

Since this analysis is intended to be *a priori*, I will be using pseudo code to describe the algorithm for further interpretation.

Function2 ( list ) {

Delete\_spaces\_at\_begining\_of\_list ( list )

If ( isEmpty( list ) )

then do

return ( 0 )

else do

word, tail = split\_in\_2lists\_at\_first\_space\_found( list )

w = take\_all\_digits ( word )

If ( are\_the\_same\_length ( word, w ) )

Then do

Return ( w + Function2 ( tail ) )

Else do

Return ( Function2 ( tail ) )

}

The functions in blue are the recursive part of the algorithm they behave in time as t(n-1). The action in green creates two lists from the parameter list, since generally when doing this kind of action the computer has to go through the whole original list, we will say that this action takes a time of n .We will assume the worst-case scenario, so the algorithm will enter the “Then Do” statement. The efficiency of the algorithm would look something like this:

F(n) = n + (n-1) + c

The bigger term kind of term would be n, so the big O should be:

O(f(n)) = n