**Pseudocode:**

sort\_algorithm(array)

ASSERT array is a list

SET array\_length <- length(array)

ASSERT array\_length > 0

FOR i <- (array\_length - 1, 0, -1)

SET largest\_element <- 0

FOR j <- (1, i + 1)

IF array[j] > array[largest\_element]

largest\_element <- j

ASSERT largest\_element < array\_length

ASSERT i < array\_length

SET array[largest\_element], array[i] <- array[i], array[largest\_element]

ASSERT array is sorted correctly

RETURN array

find\_missing\_number(sorted\_array)

ASSERT sorted\_array is a list

ASSERT array\_length > 1

ASSERT sorted\_array is sorted correctly

SET array\_length <- length(sorted\_array)

FOR i <- (0, array\_length - 1)

IF sorted\_array[i + 1] - sorted\_array[i] != 1

RETURN sorted\_array[i] + 1

RETURN None

main()

GET array

SET array <- sort\_algorithm(array)

SET missing\_number <- find\_missing\_number(array)

IF missing\_number == None

PUT “There is no missing number”

ELSE

PUT “The missing number is: {missing\_number}”

CALL main()

**Efficiency:**

We will start with the sort\_algorithm function. The algorithmic efficiency of this function is not great at all, but it gets the job done. The outer loop runs from array\_length - 1 to 0, which means it will run n times (n being the length of the array), giving it an efficiency of O(n). The inner loop runs from 1 to i + 1, meaning it will also run n times, which gives the inner loop a time efficiency of O(n). Inside the inner loop there is a swapping statement that is O(1) as it is simply setting some variables equal to values, but the overall efficiency is absolutely dominated by the inner and outer loops. This means that the overall efficiency of the function is O(n) x O(n), which results in a not great O(n^2).

The other part of the pseudocode that needs an efficiency assessment is the find\_missing\_number function. This function iterates through the array one time in order to check if there is a missing number, so the function has an algorithmic efficiency of O(n).

**Understandability:**

I would assess my code as being straightforward. Any programmer will be able to look at it and understand what it is doing after just a few moments because everything is pretty clear. Unfortunately, the complexity of the sort\_algorithm function makes it so that it isn’t immediately obvious to a programmer as to what is happening, even if the find\_missing\_number function is obvious in what it is doing because it is much more simple. All of the pieces are there and clearly stated and there isn’t much room for doubt and confusion, but I definitely wouldn’t say that there is no room for doubt or confusion. Thus, I determine this code to be straightforward.

**Malleability:**

I would say that my program is refactorable. There are no outside configuration files (getting the array from the user doesn’t count) and any changes to the logic or functionality of the program would require altering the logic of the functions in major ways.

**Trace:**

For this trace, let’s use the following 4-item array: [ 4, 1, 3, 5 ]

A table of numbers and symbols

Description automatically generated with medium confidence

The missing number was 2.