Final Exam: Find The Missing Number

The problem definition is as follows, quoted from the final exam instructions. (Some formatting may have been added)

> You are given an array of size n with the range of numbers from 1 to $n+1$. This array has no duplicates, one number is missing, and the array is *not* sorted. Find the missing number.

Finding The Solution

I originally had thought of a very efficient solution, but it unfortunately required the list to be sorted. Because of this, I thought it might be good to try sorting the list. I eventually got this down to a fairly efficient sort and was about to use that solution. However, I overheard (not sure if it's in the context of the final or not) someone mentioning subtraction.

Having heard this, I realized I can find the missing number by subtracting the sum of all the numbers from the sum of all numbers from `1` to `n + 1`. In addition to this, I remembered that in another class I had the task of writing a proof about the sum of all numbers from `1` to `n`. This led to me playing with numbers and eventually finding the equation $\frac{n}{2}(n+1)$, which conveniently has an algorithmic efficiency of `O(1)`. All that's left is to add all the numbers in the list which has an efficiency of `O(n)` and subtract which has the efficiency of `O(1)`.

This means that the program should have an overall algorithmic efficiency of `O(n)`.

Pseudocode

Using the algorithm I described as my solution above, I wrote the following pseudocode. This also includes comments at some locations for the trace table later on.

```
list = INPUT
n = LENGTH_OF(list) + 1
expected_sum = (n / 2) * (n + 1)
actual_sum = 0
# A

# Add all the numbers in the list
FOREACH num IN list
    actual_sum += num
    # B

missing_number = expected_sum - actual_sum
# C

RETURN missing_number
```

This program works with any length of list. (This includes a length of zero!) It's hard to assert efficiently in the program, but this works only with the list described in the problem definition. This makes the code extremely maleable, only needing to change what is given when asked for the input for `list` in order to find a missing number in a different list. I will also describe below when finding the efficiency how to do this without a `LENGTH_OF()` function for finding the length of the list.

Efficiency

As you can see, every line in the pseudocode has an efficiency of `O(1)`, with the exception of the loop which has an efficiency of `O(n)`. This means that **the overall efficiency is `O(n)`**. If the `LENGTH_OF()` function for some reason doesn't exist, you can simply add a counter to the loop and add one before calculating `expected_sum`. This won't change the algorithmic efficiency at all.

Trace Table

I will use a short example list that follows the rules described in the problem definition. While the exam is closed-book and no code editors are allowed, I am hoping generating the list using a python script can be accepted. The only things the actual code will be used for is generating the list, removing a number, shuffling the list, and reporting the number that was removed (used to verify my own work on the trace table gave the proper answer).

With that said, the list I will be using is:
`[9, 3, 8, 1, 2, 7, 6, 4]`
And the missing number is: `5`

Program Location	`expected_sum`	`actual_sum`	`missing_number`
A	45	0	
B	45	9	
B	45	12	
B	45	20	
B	45	21	
B	45	23	
B	45	30	
B	45	36	
B	45	40	
C	45	40	5

As expected, following this trace table and the pseudocode above found the correct missing number 5.

Final Thoughts

So, yeah. There's my solution. It's extremely effecient with the efficiency being `O(n)`. This is because it only needs to iterate through the entire list once, and that is the only part that is less efficient and `O(1)`. The program is short and insanely easy to write. It's also very readable with easy-to-understand variable names and comments, in addition to it being short and having the code separated to distinguish different functions. Not only that, but it takes advantage of a fun mathematical trick (the sum of all numbers from 1 to n)! The trace table is also easy to follow, since most of the time only one variable changes.

Bye

There ya go!