Recitation Program #1 - Run Time Explanation

There were two significantly different solutions submitted to this problem. One had a nested loop structure and one did not. For the solution without a nested loop that goes through the numbers once, the explanation is clear for the run time - each number is examined once, and a constant amount of work is done with it, so the run time is a constant times the size of the array, which will be much faster than the 10^8 simple steps a computer can do in a second or so. (This solution is extremely tricky to write, which is why I didn't recommend it. I recommended a solution that was more straight-forward and more likely to produce a bug free implementation, which had the same run time.)

Here is the explanation for the more traditional solution sketched out in the recitation that introduced the problem:

The loop structure is as follows:

```
for (int i=0; i<n; i++) {

    if (array[i] == m) {

        for (int j=i-1; j>=0; j--) {
            // break if array[i] <= m
        }

        for (int j=i+1; j<n; j++) {
            // break if array[i] <= m
        }
    }
}
```

Let's count the total number of times over all iterations, that the two inner loops run. The key is to note that the loop going backwards stops at the first number less than or equal to m, and that the loop going forwards stops at the first number less than or equal to m. Consider any stretch of the array between two values less than or equal to m, where all the numbers are greater than m:

(m1) x x x x x x x x (m2)

Let (m1) and (m2) denote numbers less than or equal to m and x denote numbers greater than m. Each number within this range will get explored precisely twice: once by the forward loop from (m1) and once by the backward loop from (m2). It's impossible for a third inner loop to explore any of these numbers, since a third such loop would have had to originate from a value to the left of (m1) or the right of (m2). By the design of the code, either of these loops would have stoped at (m1) or (m2), respectively.

Thus, no array element gets explored more than twice and the total run time is 2 times a constant times the size of the array, which is still within our required bounds.