i. According to the data, the Recursive implementation worked the best in every tested value. This could be because of the relative simplicity of implementing the function recursively, as there is at most 3 comparisons that have to be made per recursive call. In my stack implementation, the m value must be popped each time a comparison is made, and, depending on the result of the comparison, 0 - 2 objects are pushed to the stack. This means that the stack object is constantly updating, which is, in this case, more expensive.

ii. This matched my hypothesis, as I felt like the stack object would create a lot of time overhead. As objects must be created and deleted every time something is pushed or popped from the stack, it made sense to me that recursion would be the simpler, and more elegant solution.

iii. I learned that there is no easy way to convert a recursive solution into a stack. It took much trial and error to begin to think of how I could utilize the stack, what was independent of the stack. Once the solution was found, it became clear to me that it would have been difficult to use a formulaic approach to write the code. Much of my implementation was created while looking closely at this specific function's behavior.

The function itself showed me that there can be increasingly chaotic and runaway effects with recursive functions such as this one. It is deceivingly simple, but the result is something that can be computationally tough in the higher values of m. Thus, the issue of algorithmic efficiency is extremely important at this level.