1. **Algorithm tetTetranacci(n)**

Input: Nonnegative integer n

Output: nth Tetranacci number

if or or then

return

else if then

return

else

return

**Algorithm linTetranacci(n)**

Input: Positive integer n

Output: nth Tetranacci number

if or or then

return myArray

else if

return myArray

else

if then

return myArray

The linearTetranacci algorithm can compute all inputs in 0 ms. It is a linear recursive function and as such, the runtime increases linearly with the number of inputs. The quaternaryTetranacci algorithm is non-linear, and the runtime increased drastically as we increased the input. For example, for , it took 0 ms, but for , it took 15 ms, and for , it took 14768 ms.

1. The quaternaryTetranacci algorithm makes 4 calls in its recursive case, which then make 4 other calls each (ie. So for an input of n, its time complexity will be , or in Big-O notation, . The linearTetranacci algorithm makes only 1 call in its recursive case. Thus for an input of n, the time complexity will be , where c is a constant, and thus in Big-O notation, . The bottleneck we face in the first algorithm is caused by the 4 recursive calls, as they will have to wait for other function calls, some of which computing values already computing before (ie. quaternaryTetranacci(8) will call quaternaryTetranacci(7), quaternaryTetranacci(6), quaternaryTetranacci(5), quaternaryTetranacci(4) and quaternaryTetranacci(7) will call almost all the same functions.) The linearTetranacci algorithm computes values linearly, as in values will be stored and used to compute the next function resolving the bottleneck of computing the same value over and over.
2. No both functions do not use tail-recursion.

quaternaryTetranacci makes 4 recursive calls, rendering it non-linear and thus cannot use tail-recursion. linearTetranacci performs statements after the recursive call that change the result before it is returned, and thus it is not tail-recursive.