COMP 352: Data Structures and Algorithms
Winter 2025 – Programming Assignment 1
Chilka Castro 40298884
Christian David 40268798
Professor Aiman Latif Hanna
10 February 2025

Linear Recursion Pseudocode

Algorithm LinearTetranacci(k):
Input: A nonnegative integer k
Output: A tuple representing the last four Tetranacci numbers up to Tk, with the last element being Tk

```
if k <= 2 then
    return (0, 0, 0, 0)                    // T(0), T(1), and T(2) are all 0
else if k == 3 then
    return (0, 0, 0, 1)                    // T(3) is 1
else
    (a, b, c, d) = LinearTetranacci(k - 1)  // recursively get the last four values
    return (b, c, d, a + b + c + d)         // shift previous values and compute the new Tetranacci number
```

- **Time Complexity:** O(n)
- **Space Complexity:** O(n)

Multiple Recursion Pseudocode

Algorithm MultipleTetranacci(k):
Input: A nonnegative integer k
Output: The k-th Tetranacci number Tk

```
if k == 0 or k == 1 or k == 2 then
    return 0
else if k == 3 then
    return 1
else
    return MultipleTetranacci(k-1) + MultipleTetranacci(k-2) + MultipleTetranacci(k-3) +
MultipleTetranacci(k-4)
```

- **Time complexity:** $O(2^n)$
- **Space Complexity:** O(n)

COMP 352: Data Structures and Algorithms
Winter 2025 – Programming Assignment 1
Chilka Castro 40298884
Christian David 40268798
Professor Aiman Latif Hanna
10 February 2025

a.) Briefly explain why the first algorithm is of exponential complexity and the second one is linear (more specifically, how the second algorithm resolves some specific bottleneck(s) of the first algorithm).

The Multiple Recursion algorithm is exponential because each call generates four more recursive calls (for T(k-1), T(k-2), T(k-3), and T(k-4)), and these calls recompute the same subproblems repeatedly. This redundancy causes the number of function calls to grow exponentially as k increases.

In contrast, the Linear Recursion algorithm avoids this inefficiency by maintaining a tuple of the last four computed values. Instead of making multiple recursive calls, it makes just one recursive call and then updates the tuple in constant time to compute the next number. This ensures that each Tetranacci number is computed only once, resulting in linear time complexity.

b.) Do any of the previous two algorithms use tail recursion? Why or why not? Explain your answer.
If your answer is "No" then:
Both of the two previous algorithms do **not** use tail recursion.

- For linear recursion, it isn't a tail-recursion because even though it calls itself just once, it needs to update some values in an array after it gets the result from that call. It can't just make the call and immediately finish. It needs to do some updates first.
- For multiple recursion, it is not a tail-recursion because it needs to finish all its recursive calls and then add their results together before it can give an answer. This means it can't just pass the job to the next function call and forget about it. It has to wait for the results and do more work.

Can a tail-recursion version of Tetranacci calculator be designed?
i. ii. If yes; write the corresponding pseudo code for that tail-recursion algorithm and implement it in Java, and repeat the same experiments as in part (a) above. If no, explain clearly why such tail-recursive algorithm is infeasible.

- Yes, the linear recursion can be made into tail recursion. Multiple recursion cannot be turned into tail recursion because it involves many recursive calls that need to be resolved before the function can return, preventing it from being the last action. The second version of linear recursion is tail recursion because the recursion is the last thing it does.
- The tail recursion algorithm is better than the linear recursion algorithm because it uses less memory. In linear recursion, each function call creates a new stack frame. This increases memory usage(O(n) space). Tail recursion avoids this. It passes values directly in function arguments. This keeps memory use constant. (O(1) space). This makes the algorithm more efficient.

COMP 352: Data Structures and Algorithms
Winter 2025 – Programming Assignment 1
Chilka Castro 40298884
Christian David 40268798
Professor Aiman Latif Hanna
10 February 2025

Tail Recursion Pseudocode

Algorithm TailTetranacci(k, a, b, c, d): // helper function

Input: k, the position of the number in the Tetranacci sequence wanted, and a, b, c, d, the last four Tetranacci numbers.

Output: The Tetranacci number at position k.

```
if k == 0 then
    return d
else
    return TailTetranacci(k-1, b, c, d,  a + b + c + d)      // use the last four numbers to find the next one.
```

Algorithm StartTetranacci(k):

Input: k, the position of the number in the Tetranacci sequence wanted.

Output: The Tetranacci number at position k.

```
if k < 3 then
    return 0
else if k == 3 then
    return 1
else
    return TailTetranacci(k - 3, 0, 0, 0, 1)          // start the helper function with initial values.
```

- **Time Complexity:** O(n)
- **Space Complexity:** O(1)