COMP 352 Fall Semester

Assignment 1

Rui Zhao

40018813

Programming Questions:

a)

Pseudo code for Tetranacci (two versions):

Algorithm quaternaryTet(n):

```
Input: Nonnegative integer n
Output: The n-th Tetranacci number $T_n$
If n<=2 then
return 0
else if n=3 then
return 1
else
return quaternaryTet(n-1)+ quaternaryTet(n-2)+ quaternaryTet(n-3)+ quaternaryTet(n-4)
```

Algorithm linearTet(n):

```
Input: Nonnegative integer n
Output: array of Tetranacci numbers ($T_n$, $T_{n-1}$, $T_{n-2,}$ $T_{n-3}$)
If n<=2 then
return (0,0,0,0)
if n=3 then
```

```
return (1,0,0,0)
else
(i, j, k, m) = linearTet(n-1)
return (i+j+k+m, i, j, k)
```

From the records of runtime for each method, it is clear that when n goes bigger, the runtime of quaternaryTet method increases rapidly(exponentially). For example, calculate T(10) took 92448 nanoseconds, while T(15) took 2069421 nanoseconds, which increases more than 14 times. Actually, because of this rapid increment, I only tested until T(35). After that, the calculation becomes too slow that I did not want to wait for it anymore.

As for the linearTet method, when n gets bigger, the runtime increases in proportion to the n. While T(80) took 67399 nanoseconds, T(100) only took 39162 nanoseconds.

b)

For the quaternaryTet, let n_k be the number of recursive calls, then

$n_0 = n_1 = n_2 = n_3 = 1$

$n_4 = 1+1+1+1+1=5$

$n_5 = n_4 + n_3 + n_2 + n_2 + 1 = 5+1+1+1+1 = 9$

$n_6 = n_5 + n_4 + \ldots + 1 = 17$

$n_7 = 17+9+5+1+1=33$

$n_8 = 33+17+9+5+1=65$

same as the binaryFib(k), $n_k > 2^{k/2}$, which is exponential in terms of time complexity.

For the linearTet,

$n_0 = n_1 = n_2 = n_3 = 1$

$n_4 = n_3 + 1 = 2$

$n_5 = n_4 + 1 = 3$

$n_6 = n_5 + 1 = 4$

…

which is a linear complexity. The linear version is better because it significantly reduces the runtime (see observation in part a).

c) The quaternaryTet is obviously **not** a tail-recursion, because it is not linear recursion. While the linear version is **not** a tail-recursion either, because the method who calls itself will be waiting for the result array be returned before it can resume.

I don't think a tail-recursion version of Tetranacci calculator can be designed, because the characteristic of Tetranacci series is to add previous 4 elements and put the sum as the next one, the new element will always need the values of its previous element. Therefore, the recursive call cannot be the last one that being called.