Exercise on Fibonacci numbers

$$Tr_n = \begin{cases} Tr_{n-1} + Tr_{n-2} + Tr_{n-3}, & \text{if } n \geq 3 \\ 1, & \text{if } n = 1, 2 \\ 0, & \text{if } n = 0 \end{cases}$$

algorithm $\texttt{tribonacci}(\textit{integer } n) \rightarrow \textit{integer}$
  if $n = 0$ **then return** 0
  else if $n \leq 2$ **then return** 1
  else return $\texttt{tribonacci}(\text{n-1}) + \texttt{tribonacci}(\text{n-2}) + \texttt{tribonacci}(\text{n-3})$
  end if

Let's examine the Tribonacci sequence, considering the sequence number up to n=7:
$Tr_3 = Tr_2 + Tr_1 + Tr_0 = 1 + 1 + 0 = 2$
$Tr_4 = Tr_3 + Tr_2 + Tr_1 = 2 + 1 + 1 = 4$
$Tr_5 = Tr_4 + Tr_3 + Tr_2 = 4 + 2 + 1 = 7$
$Tr_6 = Tr_5 + Tr_4 + Tr_3 = 7 + 4 + 2 = 13$
$Tr_7 = Tr_6 + Tr_5 + Tr_4 = 13 + 7 + 4 = 24$

1) Let's determine the running time for tribonacci algorithm, without proving it.

Let's consider the Fibonacci sequence:

$$F_n = \begin{cases} F_{n-1} + F_{n-2}, & \text{if } n \geq 3 \\ 1, & \text{if } n = 1, 2 \end{cases}$$

We know that fibonacci2 algorithm (see slide 16), which has a structure quite similar to tribonacci algorithm, has a running time $T(n) \approx F_n \approx \Phi^n$, where $\Phi \approx 1.618 \approx 2$ (see slide 20). That is, fibonacci2 has a running time $T(n) = O(2^n)$, i.e., an exponential running time, which is quite bad.

Therefore, for tribonacci algorithm it follows that $T(n) = O(2^n)$ that is, tribonacci algorithm has an exponential running time, a bad news because the algorithm is extremely slow!

2) Let's define tribonacci2 algorithm.

As we did for fibonacci3 algorithm, we can exploit the memoization technique to achieve a faster algorithm. Let's store the numbers of the Tribonacci sequence into the array called Trib.

Trib: array of n+1 elements Trib[0], Trib[1], …, Trib[n]
Note that the array index starts from 0 rather than from 1, because in this case it is more convenient for writing the algorithm (we also consider n=0, see the definition of Tr_n provided above).

Let's consider as example n=7; from the definition of Tr_n provided above, we obtain:

Trib[0] = 0
Trib[1] = 1
Trib[2] = 1
Trib[3] = 2
Trib[4] = 4
Trib[5] = 7
Trib[6] = 13
Trib[7] = 24

	0	1	2	3	4	5	6	7
Trib	0	1	1	2	4	7	13	24

The pseudocode for tribonacci2 is:

```
algorithm tribonacci2(integer n) -> integer
1. Let Trib be an array of n+1 integers
2. Trib[0] <- 0
3. Trib[1] <- Trib[2] <- 1
4. for i=3 to n do
5.   Trib[i] <- Trib[i-1] + Trib[i-2] + Trib[i-3]
    end for
6. return Trib[n]
```

3) Let's determine the running time of tribonacci2

Lines 1, 2, 3, and 6 are executed only once.
Lines 4 and 5 are executed <= n times. Let's observe that they are executed exactly n-2 times, since the for loop is executed n-3+1 = n-2 times. Therefore, the running time is equal to 4*1 + 2*(n-2). Anyway, we do not care of the constant and multiplicative factors because we are interested in the big O notation. Therefore, the running time of tribonacci2 is $T(n)=O(n)$, meaning that tribonacci2 has a linear running time, much better than the running time of tribonacci!