Lab 2 continued

(1) Prove by induction that for all $n > 7, F_n > \left(\sqrt{2}\right)^n$.

First notice that

$$\left(\sqrt{2}\right)^n = 2^{\frac{1}{2}\cdot n} = 2^{\frac{n}{2}}.$$

Next we prove that for all $n \geq 8$, $F_n > 2^{n/2}$. It will follow that F_n is $\Omega\left(2^{n/2}\right)$.

For the base case, note that $F_8 = 21$ and $2^{8/2} = 16$. For the induction step, for all k, $8 \leq k < n$, assume $F_k > 2^{k/2}$. Then

$$\begin{aligned} F_n &= F_{n-1} + F_{n-2} \\ &> 2^{(n-1)/2} + 2^{(n-2)/2} \\ &> 2 \cdot 2^{(n-2)/2} \\ &= 2^{\frac{n-2}{2}+\frac{2}{2}} \\ &= 2^{n/2} \end{aligned}$$

(2) (a) True.

$$\lim_{n\to\infty} \frac{2^n}{2^{n-1}} = 2.$$

(b) True: It is enough to prove that $\lim_{n\to\infty} \frac{\log n}{\log_3 n}$ is finite but nonzero:

$$\lim_{n\to\infty} \frac{\log n}{\log_3 n} = \lim_{n\to\infty} \frac{\log n}{\frac{\log_n}{\log 3}} = \log 3,$$

as required.

(3) Below, pseudo-code is given for the recursive factorial algorithm `recursiveFactorial`. Use the Guessing Method to determine the worst-case asymptotic running time of this algorithm. Then verify correctness of your formula.

```
Algorithm recursiveFactorial(n)
  Input: A non-negative integer n
  Output: n!
    if (n = 0 || n = 1) then
      return 1
    return n * recursiveFactorial(n-1)
```

Step 1: Formulate the recurrence relation. We aren't concerned with the running time when $n = 0$, so we consider positive n only.

$$T(n) = \begin{cases} 4 & \text{if } n = 1 \\ T(n-1) + 4 & \text{if } n > 1 \end{cases}$$

Step 2: Guess a closed form solution.

$$\begin{aligned} T(1) &= 4 \\ T(2) &= T(1) + 4 = 4 + 4 \\ T(3) &= T(2) + 4 = 4 + 4 + 4 \\ T(n) &= T(n-1) + 4 = 4n. \end{aligned}$$

Step 3: Prove the formula from Step 2 is a solution.

Let $f(n) = 4n$. We show $f(1) = 4$ and $f(n) = f(n-1) + 4$. The first part is obviously true. We also have

$$f(n) = 4n = 4(n-1) + 4 = f(n-1) + 4,$$

as required.

Step 4: Prove correctness.

(a) *Verify valid recursion.* The recursion is valid because "$n == 0 \,||\, n == 1$" is the base case and repeated self-calls in the algorithm lead to the base case since each self-call reduces the input size by 1.

(b) *Verify base case outputs are correct.* This follows since $0! = 1$ and $1! = 1$.

(c) *Verify inductively that outputs are correct for all n.* Assume `recursiveFactorial(j)` outputs `j!` for all `j < n`, where `n > 1`. Then `recursiveFactorial` on input `n` returns `recursiveFactorial(n-1) * n`, which, by inductive hypothesis, is `(n-1)! * n` = `n!`.

(4) Devise an iterative algorithm for computing the Fibonacci numbers and compute its running time. Prove your algorithm is correct.

Below is an iterative Java method that computes F_n on input n. It executes a single loop that depends on n, so running time is $\mathrm{O}(n)$. It also uses $\mathrm{O}(n)$ space, using the `store` array.

```
int[] store;
//precondition: n is non-negative integer
public int fib(int n) {
  store = new int[n+1];
  store[0] = 0;
  store[1] = 1;

  //Loop Invariant: I(i): store[i] = F_i
```

```
    for(int i = 2; i <= n; i++) {
      store[i] = store[i-1] + store[i-2];
    }
    //postcondition: store[n] = F_n

    return store[n];
  }
  //postcondition: F_n  is returned
```

We verify correctness. We first establish a loop invariant I and show that $I(k)$ holds at the end of the $i = k$ pass, for $2 \leq k \leq n$. Our loop invariant is:

$$I(i) : \text{store}[i] = F_i$$

For the Base Case, we establish $I(2)$ at the end of the $i = 2$ pass. But this is clear since $I[0] = F_0$ and $I[1] = F_1$ and $\text{store}[2] = \text{store}[0] + \text{store}[1]$.

For the Induction Step, assume $I(j)$ holds at the end of the $i = j$ pass for each $j < k$, where $2 \leq k \leq n$; we show $I(k)$ holds at the end of the $i = k$ pass. By the Induction Hypothesis, $\text{store}[k-1] = F_{k-1}$ and $\text{store}[k-2] = F_{k-2}$. By inspection of the algorithm, it is clear therefore that

$$\text{store}[k] = \text{store}[k-1] + \text{store}[k-2] = F_{k-1} + F_{k-2} = F_k.$$

This completes the induction and shows that the loop invariant holds for $2 \leq k \leq n$. Therefore, at the end of the $i = n$ pass, we have that $\text{store}[n]$ stores F_n, and this is the value returned by the algorithm. The algorithm is therefore correct.

(5) We use the Master Formula to solve this recurrence relation:

$$T(n) = \begin{cases} 1 \text{ if } n = 1 \\ T(n/2) + n \text{ otherwise} \end{cases}$$

Here, $a = 1, b = 2, c = 1, d = 1, k = 1$. Note $a < b^k$. The Master Formula tells us therefore that $T(n) \in \Theta(n)$.

(6) See solution in ZeroesAndOnes.java