1. Problem 0.1 on page 8 of the textbook. In each of the following situations, write $O$, $\Omega$, $\Theta$ in the blank.

(a) $n - 100 = \Theta(n - 200)$
(b) $n^{1/2} = O(n^{2/3})$
(c) $100n + \log n = \Theta(n + \log^2 n)$
(d) $n \log n = \Theta(10n \log(10n))$
   $\quad n \log n = \Omega(10n + \log(10n))$
(e) $\log(2n) = \Theta(\log(3m))$
(f) $10 \log n = \Theta(\log(n^2))$
(g) $n^{1.01} = \Omega(n \log^2 n)$
(h) $n^{2/\log n} = \Omega(n \log^2 n)$
(i) $n^{0.1} = \Omega(\log^2 n)$
(j) $(\log n)^{\log n} = \Omega(n / \log n)$
(k) $\sqrt{n} = \Omega(\log^3 n)$
(l) $n^{1/2} = O(5^{\log_2 n})$
(m) $n 2^n = O(3^n)$
(n) $2^n = \Theta(2^{n+1})$
(o) $n! = \Omega(2^n)$
(p) $\log n \log n = O(2^{(\log_2 n)^2})$ [hard]
(q) $\sum_{i=1}^{n} i^k = \Theta(n^{k+1})$
2. Work problem 0.3(c) on page 9 of the textbook.

$F_n = F_{n-1} + F_{n-2}$ We start by assuming $F_n = 2^{nC}$ for some $C$. This is false, but it’s almost true, that is $\lim_{n \to \infty} \frac{F_n}{2^{nC}} = K = \Theta(1)$ for the correct value of $C$ and some positive number $K$. Making that assumption:

\[
\begin{align*}
F_{n+2} &= F_{n+1} + F_n \\
2^{C(n+2)} \cdot K &= 2^{C(n+1)} \cdot K + 2^{Cn} \cdot K
\end{align*}
\]

Divide both sides by $2^{Cn} \cdot K$:

\[
\begin{align*}
2^{2C} &= 2^{C} + 2^{0} \\
\text{Substitute } x = 2^{C} : \\
x^2 &= x + 1
\end{align*}
\]

The quadratic formula gives us two solutions.

But $x = 2^C$ cannot be negative. Thus:

\[
\begin{align*}
2^C &= \frac{1 + \sqrt{5}}{2} \text{ the golden ratio!} \\
C &= \log_2 \left( \frac{1 + \sqrt{5}}{2} \right)
\end{align*}
\]
3. Consider the following C++ program.

```cpp
void process(int n)
{
    cout << n << endl;
    if(n > 1) process(n/2);
    cout << n%2;
}

int main()
{
    int n;
    cout << "Enter a positive integer: ";
    cin >> n;
    assert(n > 0);
    process(n);
    cout << endl;
    return 1;
}
```

The last line of the output of `process(n)` is the binary numeral for `n`. 

4. The recursive algorithm implemented below as a C++ function is used as a subroutine during the calculation of the level payment of an amortized loan. What does it compute?

```cpp
float square(float x)
{
    return x*x;
}

float mystery(float x, int k)
{
    if (k == 0) return 1.0;
    else if(x == 0.0) return 0.0;
    else if (k < 0) return 1/mystery(x,-k);
    else if (k%2) return x*mystery(x,k-1);
    else return mystery(square(x),k/2);
}

mystery(x,k) returns \( x^k \).