2^4

$c = 2$
$n = 4$

CS 312
HW 1

$g(n) = 1 + c + c^2 + \ldots + c^n$

1) 0.1)

a) $f(n) = \Theta(g(n))$
b) $f(n) = O(g(n))$
c) $f(n) = \Theta(g(n))$
d) $f(n) = \Theta(g(n))$
e) $f(n) = \Theta(g(n))$
f) $f(n) = \Theta(g(n))$

m) $f(n) = O(g(n))$

2) 0.2)

		proof
1)	$\Theta(1)$ if $c < 1$	assumed
2)	$\Theta(n)$ if $c = 1$	assumed
3)	$\Theta(c^n)$ if $c > 1$	assumed
4)	$c^n = 1$ if $c = 1$	
5)	$g(n) = n$	#4 & $g(n)$
6)	$\Theta(g(n)) = c^n$ if $c > 1$	n^a dominates n^b if $a > b$
7)	if $c < 1$, c must be > 0	c must be positive & real
8)	$\Theta(g(n)) = \Theta(1)$	#7

3) a)

function ~~fab~~ fabl(n)

```
if n = 0, 1, 2
    return 1
return fabl(n-1) + fabl(n-2) * fabl(n-3)
```

This gives an upper bound Big-O of $O(3^n)$. The time is exponential w/ a base of 3. This is b/c we have 3 recursive calls each time we call fabl(n), except when $n \leq 2$. Since each call calls 3 functions, those 3 functions multiply for each call, thus 3^n.

b)

function fab2(n)

```
if n = 0, 1, 2
    return 1
create array myArray[0 ... n]
f[0, 1, 2] = 1
for(i = 3 ... n)
    myArray[i] = myArray[i-1] + myArray[i-2] * myArray[i-3]
return myArray[n]
```

As a function of n, the exact number of adds & multiplies is $f(n) = 2n - 3$ → for each n, you have to perform 1 addition & 1 multiplication, except for $n = 0, 1, 2$