Written Questions

1. a) $2^{100}$ , $2^{2013}$ , 4log(n) , log($n^{10}$) , 70n , nlog(n) , 2nlog($n^2$) , $n^{1.03}$ , $5n^6$ , $8n^6+5n^2$ , $2n^{6.5}$ , $2.2^n$ , $2.5^n$

----------------------------------------------------

b) nlog(n) is the big-theta of 2nlog($n^2$)

=========================================================

2. O() = C + C + Cn*( C + Cn*( C ) + C) + C
   > removing constant terms
   O() = n*( n )
   > simplifying the expression
   Big-O = O($n^2$)

----------------------------------------------------

Ω() = C + C + C*( C + C*( C ) + C) + C
   > simplifying the expression
   Big-Omega = Ω(1)

=========================================================

3. a) i) f(n) is in θ(g(n))
   ii) f(n) is in O(g(n))
   iii) f(n) is in O(g(n))
   iv) f(n) is in O(g(n))
   v) f(n) is in O(g(n))
   vi) f(n) is in Ω(g(n))
   vii) f(n) is in Ω(g(n))
   viii) f(n) is in O(g(n))
   ix) f(n) is in O(g(n))
   x) f(n) is in Ω(g(n))
   xi) f(n) is in Ω(g(n))

----------------------------------------------------

b) i) time quadruples when n doubles, which hints towards a Big-O that is O($n^2$)
   ii) the formula executionTime^(1/n) gives a value very close to 1 for all n that are shown,
   I therefore assume that the Big-O is O($C^n$) where C is an undefined constant.

/////////////////////////////////////////////////////////

Programming Questions

My output/times :

```
1
     algone   : 1 0ms
     algtwo   : 1 0ms
     algthree : 1 0ms
10
```

```
     algone   : 55 0ms
     algtwo   : 55 0ms
     algthree : 55 0ms
100
     algone   : 5050 0ms
     algtwo   : 5050 0ms
     algthree : 5050 0ms
1000
     algone   : 500500 0ms
     algtwo   : 500500 4ms
     algthree : 500500 0ms
10000
     algone   : 50005000 1ms
     algtwo   : 50005000 130ms
     algthree : 50005000 0ms
100000
     algone   : 5000050000 0ms
     algtwo   : 5000050000 11432ms
     algthree : 5000050000 0ms
1000000
     algone   : 500000500000 0ms
     algtwo   : 500000500000 1273096ms
     algthree : 500000500000 0ms

---------------------------------------------------------

Other computer :

1
     algone   : 1 0ms
     algtwo   : 1 0ms
     algthree : 1 0ms
10
     algone   : 55 0ms
     algtwo   : 55 0ms
     algthree : 55 0ms
100
     algone   : 5050 0ms
     algtwo   : 5050 0ms
     algthree : 5050 0ms
1000
     algone   : 500500 0ms
     algtwo   : 500500 4ms
     algthree : 500500 0ms
10000
     algone   : 50005000 1ms
     algtwo   : 50005000 22ms
     algthree : 50005000 0ms
100000
     algone   : 5000050000 1ms
     algtwo   : 5000050000 2153ms
     algthree : 5000050000 0ms
```

```
1000000
     algone   : 500000500000 3ms
     algtwo   : 500000500000 213577ms
     algthree : 500000500000 0ms
```

---------------------------------------------------------

Order of magnitude :

```
     algone   : O() = C + n*C = O(n)
     algtwo   : O() = C + n*(n*C) = O(n^2)
     algthree : O() = C = O(1)
```

---------------------------------------------------------

Comment on findings :

Although the numbers are much bigger on my computer, there is a clear and very large increase in execution time when using algorithm 2, and a smaller one on algorithm 1.

=========================================================

Observations :

The execution time generally increased at a pace similar to the jumps in number of multiplications, but since the execution time is so small there were a lot of very large jumps (in micro seconds) in execution time.

Recurrence equations :

```
     M(n) = M(n-3)+3
     M(3) = 0
     M(2) = 0
     M(1) = 0
     M(0) = 0
```

---------------------------------------------------------

Asymptotic characterization :

```
     M(n)   = M(n-3)+3
     M(n-3) = M(n-6)+3+3
     M(n-6) = M(n-9)+3+3+3
     > M(k) = 3*k/3
     Big-O = O(n)
```

---------------------------------------------------------

Tail recursion :

No, the method Multiply(A,n) is not tail recursive since the return value is then multiplied by a number before being returned again.

Here is an example of the same method in tail-recursive form :

```
public static double multiply(int[] A, int n, int result) {
    if (n < 4) {
        return 0;
    } else {
        return multiply(A, n-3, A[n-1]*result);
    {
}
```