1. Suppose that we have a C function,

```
void quadFunction(int n);
```

that has a quadratic, $\Theta(n^2)$, time complexity. Determine the time complexity for each of the following functions that use `quadFunction()`. For each function, give a brief explanation of your reasoning.

(a)
```
void function1(int n)
{
   int i;
   for (i = 0; i < n; i++)
   {
      quadFunction( n );
   }
}
```

(b)
```
void function2(int n)
{
   quadFunction( pow(n,2) );
}
```

(c)
```
void function3(int n)
{
   int i;
   for (i = 1; i < n; i*=2)
   {
      quadFunction( n );
   }
}
```

(d)
```
void function4(int n)
{
   int i;
   for (i = 0; i*i < n; i++)
   {
      quadFunction( n );
   }
}
```

(e)
```
void function5(int n, int k)
{
   int i;
   for (i = 0; i < k; i++)
   {
      quadFunction( n );
   }
}
```

(f)
```
void function6(int n)
{
   function5( n, pow(n,3) );
}
```

(g)
```
void function7(int n, int k)
{
   quadFunction( n );
   quadFunction( k );
}
```

(h)
```
void function8(int n)
{
   function7( n, pow(n,3) );
}
```

(i) What can you say about the time complexity of the following function? (This is kind of tricky, and it gets at the heart of the whole big O, Θ, Ω, idea.)

```
void function9(int n)
{
   if ( n%2 == 0 )
      quadFunction( n );
   else
      quadFunction( pow(n,2) );
}
```

2. This question is about the following recursive function.

```
void recursiveFunction(int n)
{
   if ( n == 1 )
      return;
   else
   {
      recursiveFunction( n/2 );
      recursiveFunction( n-1 );
      recursiveFunction( n/2 );
   }
}
```

(a) Write a recurrence relation for the time complexity, $T(n)$, of the function call `recursiveFunction(n)` (but don't try to solve it).

(b) Draw a "function call tree" of all the function calls that occur when you evaluate

```
recursiveFunction(8);
```

3. This question is about the following recursive function.

```
void recursiveFunction(int n, int k)
{
   if ( k == 0 )
      mysteryFunction1( n );
   else
   {
      recursiveFunction( n/2, k-1 );
      recursiveFunction( n/3, k-1 );
      mysteryFunction2( n );
   }
}
```

(a) Assume that `mysteryFunction2(n)` has time complexity $\Theta(\lg(n))$. Write a recurrence relation for the time complexity of `recursiveFunction(n,k)` (but don't try to solve it). Notice that the unknown function in the recurrence relation is a function of two variables, $T(n, k)$.

(b) Draw a "function call tree" of all the function calls that occur when you evaluate

```
recursiveFunction(216, 3);
```

(c) Assume that `mysteryFunction1(n)` has time complexity $\Theta(n)$. Use your tree from part (b) to estimate the running time $T(216, 3)$ of the function call

```
recursiveFunction(216, 3);
```

(d) Now use your recurrence relation from part (a) to derive the same answer you got in part (c) for the estimate of $T(216, 3)$.