Lab Session Software Testing 2014, Week 6 With each deliverable, indicate the time spent.

```
module Lab6

where
import Data.List
import System.Random
import Week6
```

Question 1 *Implement a function*

```
exM :: Integer -> Integer -> Integer -> Integer
```

that does modular exponentiation of x^y in polynomial time, by repeatedly squaring modulo N.

E.g., $x^{33} \mod 5$ *can be computed by means of*

$$x^{33} \pmod 5 = x^{32} \pmod 5 \times x \pmod 5.$$

$x^{32} \pmod N$ *is computed in five steps by means of repeatedly squaring modulo N:*

$$x \pmod N \rightarrow x^2 \pmod N \rightarrow x^4 \pmod N \rightarrow \ldots \rightarrow x^{32} \pmod N.$$

If this explanation is too concise, look up relevant literature.

Question 2 *Check that your implementation is more efficient than* `expM` *by running a number of relevant tests and documenting the results.*

Question 3 *In order to test Fermat's Primality Check (as implemented in function* `primeF`*), the list of prime numbers generated by Eratosthenes' sieve is useless, for Fermat's Primality Check correctly classify the primes as primes. Where the check can go wrong is on classifying composite numbers; these can slip through the Fermat test.*

Write a function `composites :: [Integer]` *that generates the infinite list of composite natural numbers. Hint: use Eratosthenes' sieve.*

Question 4 *Use the list of composite numbers to test Fermat's primality check. What is the least composite number that you can find that fools the check, for* `testF k` *with* $k = 1, 2, 3$ *? What happens if you increase k?*

Question 5 *Use the list generated by the following function for a further test of Fermat's primality check.*

```
carmichael :: [Integer]
carmichael = [ (6*k+1)*(12*k+1)*(18*k+1) |
      k <- [2..],
      isPrime (6*k+1),
      isPrime (12*k+1),
      isPrime (18*k+1) ]
```

Read the entry on Carmichael numbers on Wikipedia to explain what you find. If necessary, consult other sources.

Question 6 *Use the list from the previous exercise to test the Miller-Rabin primality check. What do you find?*

Question 7 *You can use the Miller-Rabin primality check to discover some large Mersenne primes. The recipe: take a prime p, and use the Miller-Rabin algorithm to check whether $2^p - 1$ is also prime. Find information about Mersenne primes on internet and check whether the numbers that you found are genuine Mersenne primes. Report on your findings.*

Question 8 Bonus *For RSA public key cryptography, one needs pairs of large primes with the same bitlength. Such pairs of primes can be found by trial-and-error using the Miller-Rabin primality check. Write a function for this, and demonstrate how a pair p, q that you found can be used for public key encoding and decoding.*