I-1. **[Horner]** Polynomial evaluation by use of *Horner's method* proceeds as follows. Let $f = a_n x^n + a_{n-1}x^{n-1} + \cdots + a_1 x + a_0$ be the polynomial in $R[x]$ that we wish to evaluate at $z \in R$; then

$$f(z) = (\cdots((a_n z + a_{n-1})z + a_{n-2})z + \cdots + a_1)z + a_0.$$

How many multiplications and additions are needed to compute $f(z)$ this way? How does this compare to the naïve algorithm, whereby one computes the consecutive powers $z^0, z^1, z^2, \ldots, z^n$ and adds up their products with the coefficients a_i?

I-2. **[Exponentiation]** Exponentiation (or 'powering') can be implemented by repeated multiplication and squaring. Suppose, for example, that you are given a positive integer a and a sequence of binary digits D representing the integer n (most significant bit first); decribe two algorithms for computing a^n:

(i) reading the bits in D *right-to-left*, one at a time, so obtaining the least significant bit first;

(ii) reading the bits in D *left-to-right*, so obtaining the most significant bit first.

I-3. **[Karatsuba]** The algorithm of *Karatsuba* performs integer multiplication recursively. Suppose that you are given integers x and y of at most n bits each; we will suppose (for ease of presentation) that $n = 2^k$ for some positive integer k. Now write

$$x = x_0 + x_1 2^{n/2}, \quad y = y_0 + y_1 \cdot 2^{n/2},$$

with x_0, x_1, y_0, y_1 integers of (at most) $\frac{n}{2} = 2^{k-1}$ bits.

(i) Show that by using only *three* multiplications (and 6 subtractions/additions) of integers of 2^{k-1} bits you can compute the 'digits' a, b, c for $x \cdot y = a + b2^{n/2} + c2^n$. [Hint: use $x_1 - x_0$ and $y_0 - y_1$.]

(ii) Conclude that recursive use of this trick leads to an $\mathcal{O}((^2\log n)^\ell)$ algorithm for computing $x \cdot y$, where $\ell = {}^2\log 3$.

I-4. **[Pollard-ρ]** Implement Pollard's ρ algorithm, including the optimisation indicated in the slides, for integer factorisation. As an indication of its performance, if m is a product of two primes of k and $2k$ decimal digits, how large can k be for your implementation to find the factorization in no more than 5 (run-time) minutes?