Solution to Problem 17 (32.1)

Graded by: Pavel Laskov
Adopted from a solution by:

Problem Statement

a. Show how to multiply two linear polynomials $ax + b$ and $cx + d$ using only three multiplications. (*Hint:* One of multiplications is $(a + b)(c + d)$.)

b. Give two divide-and-conquer algorithms for multiplying two polynomials of degree-bound n that run in time $\Theta(n^{\lg 3})$. The first algorithm should divide the input polynomial coefficients into a high half and a low half, and the second algorithm should divide them according to whether their index is even or odd.

c. Show that two n-bit integers can be multiplied in $O(n^{\lg 3})$ steps, where each step operates on at most a constant number of 1-bit values.

Part a.

Conventional polynomial multiplication uses 4 coefficient multiplications:

$$(ax + b)(cx + d) = acx^2 + (ad + bc)x + bd$$

However, notice the following relation:

$$(a + b)(c + d) = ad + bc + ac + bd$$

The first two components are exactly the middle coefficient for product of two polynomials. Therefore, the product can be computed as:

$$(ax + b)(cx + d) = acx^2 + ((a + b)(c + d) - ac - bd)x + bd$$

The latter expression has only three multiplications.

Part b. High/Low Algorithm

Let p denote the vector of coefficients of the first polynomial P, q denote the vector of coefficients of the second polynomial Q. Assume both of these vectors are of length $n = \max\{\text{length}(p_1), \text{length}(p_1)\}$ (whichever is smaller is padded with leading zeros). Let $m = \lceil \frac{n}{2} \rceil$. It can be easily seen that

$$\begin{aligned} P = p_0 + p_1 x + \ldots + p_{n-1}x^{n-1} &= p_0 + p_1 x + \ldots + p_{m-1}x^{m-1} \\ &\quad + x^m(p_m + p_{m+1}x + \ldots + p_{n-1}x^{n-1-m}) \\ &= Ax^m + B \end{aligned}$$

where

$$\begin{aligned} A &= p_m + p_{m+1}x + \ldots + p_{n-1}x^{n-1-m} \\ B &= p_0 + p_1 x + \ldots + p_{m-1}x^{m-1} \end{aligned}$$

Likewise,

$$\begin{aligned} Q = q_0 + q_1 x + \ldots + q_{n-1}x^{n-1} &= q_0 + q_1 x + \ldots + q_{m-1}x^{m-1} \\ &\quad + x^m(q_m + q_{m+1}x + \ldots + q_{n-1}x^{n-1-m}) \\ &= Cx^m + D \end{aligned}$$

where

$$C = q_m + q_{m+1}x + \ldots + q_{n-1}x^{n-1-m}$$
$$D = q_0 + q_1 x + \ldots + q_{m-1}x^{m-1}$$

Using the result of Part a. we can write the following expression for the product of P and Q:

$$(Ax^m + B)(Cx^m + D) = ACx^{2m} + ((A+B)(C+D) - AC - BD)x^m + BD \tag{1}$$

Based on equation (1) we can define a divide-and-conquer algorithm for polynomial multiplication:

- Split polynomials P and Q of degree-bound n into polynomials A, B, C, D of degree-bound m.
- Calculate the expression (1) for $(Ax^m + B)(Cx^m + D)$ using recursive calls for polynomial multiplication.

The resulting algorithm is summarized in Algorithm 1:

Algorithm 1 High/Low algorithm

```
proc RMul(p, q)
  begin
    n ← p.size()
    m ← ⌈n/2⌉
    if p.size() = 1
      then return pq                          // Size of q is also 1. See Lemma 1.
       else
            a ← p[m, n − 1]                    // Split p and q in halfs
            b ← p[0, m − 1]
            c ← q[m, n − 1]
            d ← q[0, m − 1]
            tmp1 ← RMul(a + b, c + d)    // Do recursive multiplications
            tmp2 ← RMul(a, c)
            tmp3 ← RMul(b, d)
            return tmp2 ≪ n + (tmp1 − tmp2 − tmp3) ≪ m + tmp3
  end
```

The operation $p \ll k$ denotes "shift p to the left by k digits". This is necessary to produce correct powers of x.

Correctness of the algorithm follows from the fact that it starightforwardly implements equation (1). The shifitng operation accounts for appropriate powers of x in the resulting polynomial. The only part which requires special attention is the termination condition. The following lemma justifies the condition used in the algorithm.

Lemma 1 *length*(p) = *length*(q).

Proof. I will prove the claim by induction on valid recursion depth d (assuming depth d is reachable). For $d = 0$, that is, during the initial call to *RMul*, the claim is true from the assumption that two vectors are aligned. Suppose the claim is true for some depth d and recursive calls are further made to depth $d+1$. length$(a+b)$ =

$m = \text{length}(c+d)$. $\text{length}(b) = m = \text{length}(d)$. If n is even then $\text{length}(\text{a}) = m = \text{length}(c)$, otherwise $\text{length}(\text{a}) = m - 1 = \text{length}(c)$. It can be easily seen that sizes of both argumens in all three recursive calls are the same. q.e.d.

Part b. Even/Odd Algorithm

Let $n_e = 2\lfloor \frac{n}{2} \rfloor$. Let $n_o = 2\lceil \frac{n}{2} \rceil - 1$. Under the same assumptions as in Part a. another decomposition of P and Q can be derived:

$$\begin{aligned} P = p_0 + p_1 x + \ldots + p_{n-1}x^{n-1} &= p_0 + p_2 x^2 + \ldots + p_{n_e} x^{n_e} \\ &\quad + x(p_1 + p_3 + \ldots + p_{n_o} x^{n_o - 1}) \\ &= Ax + B \end{aligned}$$

where

$$\begin{aligned} A &= p_1 + p_3 + \ldots + p_{n_o} x^{n_o - 1} \\ B &= p_0 + p_2 x^2 + \ldots + p_{n_e} x^{n_e} \end{aligned}$$

Likewise,

$$\begin{aligned} Q = q_0 + q_1 x + \ldots + q_{n-1}x^{n-1} &= q_0 + q_2 x^2 + \ldots + q_{n_e} x^{n_e} \\ &\quad + x(q_1 + q_3 + \ldots + q_{n_o} x^{n_o - 1}) \\ &= Cx + D \end{aligned}$$

where

$$\begin{aligned} C &= q_1 + q_3 + \ldots + q_{n_o} x^{n_o - 1} \\ D &= q_0 + q_2 x^2 + \ldots + q_{n_e} x^{n_e} \end{aligned}$$

Using the result of Part a. we can write the following expression for the product of P and Q:

$$(Ax + B)(Cx + D) = ACx^2 + ((A+B)(C+D) - AC - BD)x + BD \tag{2}$$

The same divide-and-conquer scheme as in the high/low algorithm is applied with a slightly different "conquer" phase: instead of shifting the powers of x by n and m, they are shifted by 2 and 1 respectively. The even/odd algorithm is summarized in Algorithm 2.

Correctness of the even/odd algorithm follows from the fact that its recursive part is a straightforward implementation of equation (2). It can be also shown in a similar way that $\text{length}(p) = \text{length}(q)$ at every recursive call to *RMul*.

Complexity of all non-recursive operations in the high/low and even/odd algorithms is $O(n)$. Therefore the corresponding recurrence relation is

$$T(n) = 3T\left(\left\lceil \frac{n}{2} \right\rceil\right) + O(n)$$

a solution to which is $\Theta(n^{\lg 3})$.

Part c.

Observe that an n-bit integer (base 2) $d_{n-1} \ldots d_1 d_0$ is the evaluation of a degree-n polynomial at $x = 2$:

$$N = d_{n-1} 2^{n-1} + \ldots + d_1 2^1 + d_0 2^0$$

Thus any of the algorithms of Part b. can be applied to multiplication of two integers. The running time will be $O(n^{\lg 3})$ (not Θ!) because the lower bound actually depends on the highest non-zero digit in N.

Algorithm 2 Even/Odd algorithm

```
proc RMul(p, q)
  begin
    if p.size() = 1
      then return pq                                   // Size of q is also 1.
      else
           a ← p[odd]                                  // Split p and q in halfs
           b ← p[even]
           c ← q[odd]
           d ← q[even]
           tmp1 ← RMul(a + b, c + d)                   // Do recursive multiplications
           tmp2 ← RMul(a, c)
           tmp3 ← RMul(b, d)
           return tmp2 ≪ 2 + (tmp1 − tmp2 − tmp3) ≪ 1 + tmp3
  end
```

Grading Policy

Points:

2	Part a.
3+3	Part b.
2	Part c.