of the numbers in question, and not their *sizes*. For the algorithm to be efficient, we must reduce the sizes. For example, if n is very large (say 1000 digits) and $m = 24$, we will need to subtract 24 from n approximately $n/24$ times. This computation will take $O(n)$ steps, which is exponential in the size of n.

Let's look at this algorithm again. We subtract m from n and apply the same algorithm to $n-m$ and m. If $n-m$ is still larger than m, we subtract m again. In other words, we keep subtracting m from n until the result becomes less than m. But this is exactly the same as dividing n by m and looking at the remainder. Division can be done quickly. This leads directly to Euclid's algorithm, which is presented in Fig. 9.3.

Complexity We claim that Euclid's algorithm has linear running time in the size of $n+m$; specifically, its running time (counting each operation as one step independent of the size of the operands) is $O(\log(n+m))$. To prove this claim, it is sufficient to show that the value of a is reduced by half in a constant number of iterations. Let's look at two consecutive iterations of algorithm *GCD*. In the first iteration, a and b $(a > b)$ are changed into b and $a \bmod b$. Then, in the next iteration, they are changed into $a \bmod b$ and $b \bmod (a \bmod b)$. So, in two iterations, the first number a is changed to $a \bmod b$. But, since $a > b$, we have $a \bmod b < a/2$, which establishes the claim.

9.4 Polynomial Multiplication

Let $P = \sum_{i=0}^{n-1} p_i x^i$, and $Q = \sum_{i=0}^{n-1} q_i x^i$, be two polynomials of degree $n-1$. A polynomial is represented by its ordered list of coefficients.

Algorithm GCD *(m, n)*
Input: *m* and *n* (two positive integers).
Output: *gcd* (the gcd of *m* and *n*).

```
begin
   a := max( n, m ) ;
   b := min( n, m ) ;
   r := 1 ;
   while r > 0 do  { r is the remainder }
      r := a mod b ;
      a := b ;
      b := r ;
   gcd := a
end
```

Figure 9.3 Algorithm *GCD*.

The Problem Compute the product of two given polynomials of degree $n-1$.

$$PQ = \left(p_{n-1}x^{n-1} + \cdots + p_0\right)\left(q_{n-1}x^{n-1} + \cdots + q_0\right) = \tag{9.2}$$

$$p_{n-1}q_{n-1}x^{2n-2} + \cdots + \left(p_{n-1}q_{i+1} + p_{n-2}q_{i+2} + \cdots + p_{i+1}q_{n-1}\right)x^{n+i} + \cdots + p_0q_0.$$

We can compute the coefficients of PQ directly from (9.2). It is easy to see that, if we follow (9.2), then the number of multiplications and additions will be $O(n^2)$. Can we do better? We have seen by now so many improvements of straightforward quadratic algorithms that it is not surprising that the answer is positive. A complicated $O(n \log n)$ algorithm will be discussed in Section 9.6. But first, we describe a simple divide-and-conquer algorithm.

For simplicity, we assume that n is a power of 2. We divide each polynomial into two equal-sized parts. Let $P = P_1 + x^{n/2}P_2$, and $Q = Q_1 + x^{n/2}Q_2$, where

$$P_1 = p_0 + p_1x + \cdots + p_{n/2-1}x^{n/2-1}, \quad P_2 = p_{n/2} + p_{n/2+1}x + \cdots + p_{n-1}x^{n/2-1},$$

and

$$Q_1 = q_0 + q_1x + \cdots + q_{n/2-1}x^{n/2-1}, \quad Q_2 = q_{n/2} + q_{n/2+1}x + \cdots + q_{n-1}x^{n/2-1}.$$

We now have

$$PQ = (P_1 + P_2x^{n/2})(Q_1 + Q_2x^{n/2}) = P_1Q_1 + (P_1Q_2 + P_2Q_1)x^{n/2} + P_2Q_2x^n.$$

The expression for PQ now involves products of polynomials of degree $n/2$. We can compute the product of the smaller polynomials (e.g., P_1Q_1) by induction, then add the results to complete the solution. Can we use induction directly? The only constraints are that the smaller problems be exactly the same as the original problem, and that we know how to multiply polynomials of degree 1. Both conditions are clearly satisfied. The total number of operations $T(n)$ required for this algorithm is given by the following recurrence relation:

$$T(n) = 4T(n/2) + O(n), \quad T(1) = 1.$$

The factor 4 comes from the 4 products of the smaller polynomials, and the $O(n)$ comes from adding the smaller polynomials. The solution of this recurrence relation is $O(n^2)$ (see Section 3.5.2), which means that we have not achieved any improvement (see Exercise 9.4).

To get an improvement to the quadratic algorithm we need to solve the problem by solving less than four subproblems. Consider the following multiplication table (the reason we use such an elaborate table for this simple notation will become apparent in the next section).

$\times$	P_1	P_2
Q_1	A	B
Q_2	C	D

We want to compute $A+(B+C)x^{n/2}+Dx^n$. The important observation is that we do not have to compute B and C separately; we need only to know their sum! If we compute the product $E=(P_1+P_2)(Q_1+Q_2)$, then $B+C=E-A-D$. Hence, we need to compute only three products of smaller polynomials: A, D, and E. All the rest can be computed by additions and subtractions, which contribute only $O(n)$ to the recurrence relation anyway. The new recurrence relation is

$$T(n)=3T(n/2)+O(n),$$

which implies $T(n)=O(n^{\log_2 3})=O(n^{1.59})$.

Notice that the polynomials P_1+P_2 and Q_1+Q_2 are related to the original polynomials in a strange way. They are formed by adding coefficients whose indices differ by $n/2$. This is quite a nonintuitive way to multiply polynomials, yet this algorithm reduces the number of operations significantly for large n.

□ Example 9.1

Let $P=1-x+2x^2-x^3$, and $Q=2+x-x^2+2x^3$. We compute their product using the divide-and-conquer algorithm. We carry the recursion only one step.

$$A=(1-x)\cdot(2+x)=2-x-x^2,$$

$$D=(2-x)\cdot(-1+2x)=-2+5x-2x^2,$$

and

$$E=(3-2x)\cdot(1+3x)=3+7x-6x^2.$$

From E, A, and D, we can easily compute $B+C=E-A-B$:

$$B+C=3+3x-3x^2.$$

Now, $P\cdot Q=A+(B+C)x^{n/2}+Dx^n$, and we have

$$P\cdot Q=2-x-x^2+3x^2+3x^3-3x^4-2x^4+5x^5-2x^6$$

$$=2-x+2x^2+3x^3-5x^4+5x^5-2x^6.$$

Notice that we used 12 multiplications compared to 16 in the straightforward algorithm, and 12 additions and subtractions instead of 9. (We could have reduced the number of multiplications to 9 if we had carried the recursion one more step.) The savings are, of course, much larger when n is large. (The number of additions and subtractions remains within a constant factor of that in the straightforward algorithm, whereas the number of multiplications is reduced by about $n^{0.4}$.) □