Polynomial Multiplication.

Given two univariate polynomials

$$A(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_{n-1} x^{n-1},$$

$$B(x) = b_0 + b_1 x + b_2 x^2 + \cdots + b_{n-1} x^{n-1},$$

each of degree $n-1$,

Compute the product polynomial

$$C(x) = A(x) \times B(x)$$

$$= a_0 b_0 + (a_0 b_1 + a_1 b_0) x + (a_0 b_2 + a_1 b_1 + a_2 b_0) x^2$$

$$+ \cdots + \Big(\sum_{\substack{0 \le i \le n-1, \\ 0 \le j \le n-1: \\ i+j = K}} a_i b_j \Big) x^K + \cdots + a_{n-1} b_{n-1} x^{2n-2}$$

We will refer to a polynomial of degree $n-1$ as an n-polynomial. Although the product of 2 n-polynomials is a $(2n-1)$-polynomial, we think of it as a $(2n)$-polynomial with $C_{2n-1} = 0$.

```
Multiply(A, B, n)
Initialize array C of length 2n to 0's.

For k ← 0 to n-1 do
    For i ← 0 to k do
        C[k] ← C[k] + A[i] × B[k-i]
    end for
end for

For k ← n to 2n-2 do
    For i ← k-(n-1) to n-1 do
        C[k] ← C[k] + A[i] × B[k-i]
    end for
end for
Return C
```

The ~~Running~~ Worst case Running time of this algorithm is $\Theta(n^2)$. Can we do better?

The Key to this is a different ~~view~~ approach to multiplying polynomials — a recursive approach.

Can we ~~not~~ reduce the multiplication of two n-polynomials to several multiplications of two $\left(\frac{n}{2}\right)$-polynomials?

Henceforth, we will assume that n is a power of 2 for convenience. If $n > 1$, this will guarantee that $\frac{n}{2}$ is an integer and in fact a power of 2. We'll worry about the general case later.

Write

$$A(x) = a_0 + a_1 x + \cdots + a_{\frac{n}{2}-1} x^{\frac{n}{2}-1}$$

$$+ x^{n/2}\left(a_{n/2} + a_{n/2+1} x + \cdots + a_{n-1} x^{\frac{n}{2}-1}\right)$$

$$= A_{low}(x) + x^{n/2} A_{high}(x),$$

where A_{low} and A_{high} are $\frac{n}{2}$ - polynomials.

Similarly, write

$$B(x) = B_{low}(x) + x^{n/2} B_{high}(x).$$

Now, $C(x) = A(x) \times B(x)$

$$= A_{low} \times B_{low} + x^{n/2}\left(A_{low} \times B_{high} + A_{high} \times B_{low}\right)$$

$$+ x^{n} \times A_{high} \times B_{high}. \quad \text{-----} \quad (1)$$

So here is how we compute C based on formula (1).

Recursive-multiply (A, B, n)

Create array C of length $2n$.

If $n = 1$

$C[0] \leftarrow A[0] \times B[0]$

$C[1] \leftarrow 0$

Return C

else

Extract A_{low}, A_{high}, B_{low}, B_{high}

(arrays of length $\frac{n}{2}$).

$N_0 \leftarrow$ Recursive-multiply $(A_{low}, B_{low}, \frac{n}{2})$

$N_1 \leftarrow$ Recursive-multiply $(A_{low}, B_{high}, \frac{n}{2})$

$N_2 \leftarrow$ Recursive-multiply $(A_{high}, B_{low}, \frac{n}{2})$

$N_3 \leftarrow$ Recursive-multiply $(A_{high}, B_{high}, \frac{n}{2})$

Fill in the entries of C from

N_0, N_1, N_2, N_3 based on (1)

Return C

endif.

Denoting by $T(n)$ the worst-case running time of the algorithm as a function of n, we see that

$$T(n) \leq 4T\left(\frac{n}{2}\right) + c_1 n \quad \text{for } n > 1$$

$$T(1) \leq c_2$$

for constants $c_1, c_2 > 0$.

~~As we see~~

$T(n)$ can be bounded as $O(n^2)$ based on this. Details in class.

We can also see that $T(n)$ is $\Omega(n^2)$. This is because n^2 recursive calls are made with $n = 1$. Again, more detail in class.

This more sophisticated algorithm is not really better than the earlier one. But the analysis suggests the direction we should look at for a better algorithm.

Rather than use 4 recursive calls on $\frac{n}{2}$-polynomials, can we get away with 3 ?

Consider multiplying two 2-polynomials:

$$a_{low} + a_{high}\, y$$
$$b_{low} + b_{high}\, y .$$

The product is

$$a_{low} b_{low} + (a_{low} b_{high} + a_{high} b_{low})\, y + a_{high} b_{high}\, y^2 .$$

Here is a question to ponder:

Given a_{low}, a_{high}, b_{low}, b_{high},

can we obtain the 3 coefficients of the product polynomial using just 3, rather than 4, multiplications?

We are allowed to use any number of additions and subtractions.

~~Think of~~

After some thought, here is the solution:

Compute $T_0 \leftarrow a_{low} + a_{high}$

$T_1 \leftarrow b_{low} + b_{high}$

Compute $N_0 \leftarrow T_0 \times T_1$

$N_1 \leftarrow a_{low} \times b_{low}$

$N_2 \leftarrow a_{high} \times b_{high}$

Note that product polynomial

is $N_1 + (N_0 - N_1 - N_2)\, y$

$+ N_2 y^2$. .... (2)

We used 3 multiplications only.

This suggests the following recursive algorithm.

Faster-multiply (A, B, n).

Create array C of length $2n$.

If $n = 1$

$C[0] \leftarrow A[0] \times B[0]$

$C[1] \leftarrow 0$

Return C

else

Extract A_{low}, A_{high}, B_{low}, B_{high}

$T_0 \leftarrow A_{low} + A_{high}$

$T_1 \leftarrow B_{low} + B_{high}$

// Note: T_0 and T_1 are of length $\frac{n}{2}$

$N_0 \leftarrow$ Faster-multiply $(T_0, T_1, \frac{n}{2})$

$N_1 \leftarrow$ Faster-multiply $(A_{low}, B_{low}, \frac{n}{2})$

$N_2 \leftarrow$ Faster-multiply $(A_{high}, B_{high}, \frac{n}{2})$.

Fill in entries of C from N_0, N_1, N_2 based on formula:

$$C(x) = N_1(x) + x^{\frac{n}{2}}\left(N_0(x) - N_1(x) - N_2(x)\right) + x^n N_2(x)$$

Return C.

endif.

Worst-Case running time $T(n)$ of this algo satisfies:

$$T(n) \leq 3T\left(\frac{n}{2}\right) + c_1 n \quad \text{if } n > 1$$

$$T(1) \leq c_2 ,$$

where $c_1, c_2 > 0$ are constants.

We show that $T(n) = O(n^{\log_2 3})$.

Details in class. Note that $\log_2 3 < 2$.

We also show that $T(n)$ is $\Omega(n^{\log_2 3})$,

so $T(n) = \Theta(n^{\log_2 3})$.