Problem 30-1 is to be done in the spirit of "complete" problems, but with specific requirements. Here is a reworking of the problem.

Find Python starter code in `~tvandrun/cs445/c39p1p` (as always, you have the option of writing code from scratch instead). The interesting stuff is in the file `c30p1/polynomial.py`. This code has many similarities to the code examples we had in class, but one important difference: It doesn't have any classes to represent polynomials, but instead we just use lists of coefficients. Thus all operations are stand-alone functions, not methods of classes.

a. The book refers to two polynomials $ax + b$ and $cx + d$, but I suggest you think of them instead as $a_0 + a_1x$ and $b_0 + b_1x$, to use a notation that can be scaled up consistently in part b. For the equation

$$(a_0 + a_1x) \cdot (b_0 + b_1x) = c_0 + c_1x + c_2x^2$$

find forms for c_0, c_1, and c_2 that require only three distinct multiplications. (You can use the same multiplication more than once, that is, the result of a multiplication can be stored and reused.) The hint the book gives is that one of the multiplications is $(a_0 + a_1) \cdot (b_0 + b_1)$. (*Don't overthink this—it's not that hard, especially with the hint. This part sets you up for a slightly harder but analogous algebraic task in part b.*)

Then use this idea to implement the function `polyn_product_linear()`, which you can test using `test_ppl.py`. You are not required to add more testcases to that one.

b. The problem in the book refers to two algorithms, so we'll take it as if it were two parts, **30-1.b.i** and **30-1.b.ii**. In either case the task is multiplying two polynomials, $\sum_{i=0}^{n-1} a_ix^i \cdot \sum_{i=0}^{n-1} b_ix^i$. Assume n is a power of 2.

For part **b.i**, you are asked to think of the problem this way:

$$\begin{aligned}
\sum_{i=0}^{n-1} a_ix^i \cdot \sum_{i=0}^{n-1} b_ix^i &= \left(\sum_{i=0}^{\frac{n}{2}-1} a_ix^i + x^{\frac{n}{2}} \sum_{i=0}^{\frac{n}{2}-1} a_{i+\frac{n}{2}}x^i\right) \cdot \left(\sum_{i=0}^{\frac{n}{2}-1} b_ix^i + x^{\frac{n}{2}} \sum_{i=0}^{\frac{n}{2}-1} b_{i+\frac{n}{2}}x^i\right) \\
&= \underbrace{\left(\sum_{i=0}^{\frac{n}{2}-1} a_ix^i \cdot \sum_{i=0}^{\frac{n}{2}-1} b_ix^i\right)}_{\alpha} + x^n \underbrace{\left(\sum_{i=0}^{\frac{n}{2}-1} a_{i+\frac{n}{2}}x^i \cdot \sum_{i=0}^{\frac{n}{2}-1} b_{i+\frac{n}{2}}x^i\right)}_{\beta} \\
&\quad + x^{\frac{n}{2}} \underbrace{\left(\sum_{i=0}^{\frac{n}{2}-1} a_ix^i \cdot \sum_{i=0}^{\frac{n}{2}-1} b_{i+\frac{n}{2}}x^i + \sum_{i=0}^{\frac{n}{2}-1} a_{i+\frac{n}{2}}x^i \cdot \sum_{i=0}^{\frac{n}{2}-1} b_ix^i\right)}_{\delta}
\end{aligned}$$

Where α and $x^n\beta$ correspond to F and L in FOIL, and $x^{\frac{n}{2}}\delta$ corresponds to O + I. Moreover, α and β are two of the three (recursive) polynomial multiplications you are allowed, and you need to find a third polynomial multiplication (call its result γ) such that $\delta = \gamma - \alpha - \beta$.

Finding γ algebraically counts for the proof part of this problem. Implement this by finishing the function `polyn_product_dchlr()` (that's *polynomial product divide-and-conquer high-low recursive*). Test this using `test_ppdchl.py`. This inherits testcases from `test_polyn_prod.py`, which is set up so that it is easy for you to add testcases which will be used both here and in part **b.ii**.

Finally, show that this is $\Theta(n^{\lg 3})$ by finding and solving a recurrence. Don't worry about the time spent generating or manipulating lists; consider only arithmetic operations.

Part **b.ii** is similar and should be done similarly, but this time think of the problem as

$$\begin{aligned}\sum_{i=0}^{n-1} a_i x^{2i} \cdot \sum_{i=0}^{n-1} b_i x^{2i} &= \left(\sum_{i=0}^{\frac{n}{2}-1} a_{2i} x^{2i} + x \sum_{i=0}^{\frac{n}{2}-1} a_{2i+1} x^{2i}\right) \cdot \left(\sum_{i=0}^{\frac{n}{2}-1} b_{2i} x^{2i} + x \sum_{i=0}^{\frac{n}{2}-1} b_{2i+1} x^{2i}\right) \\ &= \underbrace{\left(\sum_{i=0}^{\frac{n}{2}-1} a_{2i} x^{2i} \cdot \sum_{i=0}^{\frac{n}{2}-1} b_{2i} x^{2i}\right)}_{\alpha} + x^2 \underbrace{\left(\sum_{i=0}^{\frac{n}{2}-1} a_{2i+1} x^{2i} \cdot \sum_{i=0}^{\frac{n}{2}-1} b_{2i+1} x^{2i}\right)}_{\beta} \\ &\quad + x \underbrace{\left(\sum_{i=0}^{\frac{n}{2}-1} a_{2i} x^{2i} \cdot \sum_{i=0}^{\frac{n}{2}-1} b_{2i+1} x^{2i} + \sum_{i=0}^{\frac{n}{2}-1} a_{2i+1} x^{2i} \cdot \sum_{i=0}^{\frac{n}{2}-1} b_{2i} x^{2i}\right)}_{\delta}\end{aligned}$$

As before, find a polynomial product (call its result γ) such that $\delta = \gamma - \alpha - \beta$.

Implement this by finishing the function `polyn_product_dceor()` (*polynomial product divide-and-conquer even-odd recursive*). You may find it useful to represent a polynomial as a function of x^2 instead of x. For example `[2,3,7]` interpreted as a function of x^2 means $2 + 3x^2 + 7x^4$. To convert this back to a function of x, add a zero after every entry, ie make the list `[2,0,3,0,7,0]`. The provided function `perforate()` does this. Test using `test_ppdceo.py`. You shouldn't need any new testcases, since the testcases you wrote in `test_polyn_prod.py` will be inherited. The analysis should be similar.

c. The last part of the problem (top of page 921) can be answered with a short verbal explanation.