30.1-2 Since $A(x) = (x - x_0)q(x) + r$ and

$$A(x) = \sum_{i=0}^{n} a_i x^i, \qquad q(x) = \sum_{j=0}^{n-1} q_j x^j$$

then

$$\begin{aligned}(x - x_0)q(x) + r &= (x - x_0)\sum_{i=0}^{n-1} q_j x^j + r \\ &= q_{n-1}x^n + (q_{n-2} + x_0 q_{n-1})x^{n-1} + (q_{n-3} + x_0 q_{n-2})x^{n-2} + \ldots + (q_0 + x_0 q_1)x + q_0 x_0 + r\end{aligned}$$

Equating coefficients on both sides of $A(x) = (x - x_0)q(x) + r$, one obtains

$$\begin{aligned}q_{n-1} &= a_n \\ q_{n-2} &= a_{n-1} - x_0 q_{n-1} \\ &\vdots \\ q_{n-k} &= a_{n-k+1} - x_0 q_{n-k+1} \\ &\vdots \\ q_0 &= a_1 - x_0 q_1 \\ r &= a_0 - x_0 q_0\end{aligned}$$

Therefore, setting $q_{n-1} := a_n$ and applying formulas $q_{n-k} := a_{n-k+1} - x_0 q_{n-k+1}$ for $k = 1, 2, \ldots, n$ gives an $\Theta(n)$ algorithm for computing $q(x)$ and r by $r := a_0 - x_0 q_0$

30.1-3 Assume that (x_i, a_i) $(i = 0, 1, \ldots, n-1)$ is given point-value representation of $A(x)$, i.e. $a_i = A(x_i)$. Since $A^{\text{rev}}(x) = x^{n-1}A(x^{-1})$ then $A^{\text{rev}}(x_i^{-1}) = x_i^{-(n-1)}A(x_i) = x_i^{-(n-1)}a_i$ for every $i = 0, 1, \ldots, n-1$. In other words $(x_i, x_i^{-(n-1)}a_i)$ $(i = 0, 1, \ldots, n-1)$ is required point-value representation of $A^{\text{rev}}(x)$.

30.1-4 Assume that (x_i, p_i) $(i = 0, 1, \ldots, k)$ are given point-value pairs where $k < n - 1$. Choose arbitrary distinct $x_{k+1}, x_{k+2}, \ldots, x_{n-1} \in \mathbb{R} \setminus \{x_0, x_1, \ldots, x_k\}$ and arbitrary values $p_{k+1}, p_{k+2}, \ldots, p_{n-1}$. Then, according to Theorem 30.1 exists a unique polynomial $p(x)$ of degree-bound n satisfying $p(x_i) = p_i$ for all $i = 0, 1, \ldots, n-1$. Now choosing different $p'_{k+1}, p'_{k+2}, \ldots, p'_{n-1}$, in the same way, one gets a new polynomial $p'(x)$ such that $p(x_i) = p_i$ for all $i = 0, 1, \ldots, k$ and $p(x_i) = p'_i$ for all $i = k+1, k+2, \ldots, n-1$. Now $p'(x_i) = p'_i \neq p_i = p(x_i)$ $(i = k+1, k+2, \ldots, n-1)$ implies that p and p' are distinct polynomials passing through point-value pairs (x_i, p_i) $(i = 0, 1, \ldots, k)$.

30.1-7 Let us introduce polynomials

$$P_A(x) = \sum_{i \in A} x^i, \qquad P_B(x) = \sum_{j \in B} x^j.$$

Then

$$Q(x) = P_A(x)P_B(x) = \left[\sum_{i \in A} x^i\right]\left[\sum_{j \in B} x^j\right] = \sum_{\substack{i \in A \\ j \in B}} x^{i+j} = \sum_{k=0}^{20n} c_k x^k$$

where c_k is the number of appearances of the term x^k in the previous sum, for $k = 0, 1, \ldots, 20n$. Since each k is obtained as the sum $k = i + j$, $i \in A$ and $j \in B$, we conclude that $C = \{k \mid c_k > 0,\ k = 0, 1, \ldots, 2n\}$ and that c_k is required number of times.

Coefficients c_k of $Q(x)$ can be computed in $\mathcal{O}(n \lg n)$ time using FFT polynomial multiplication.

30.2-1 $\omega_n^{n/2} = \left[e^{2\pi i/n}\right]^{n/2} = e^{2\pi i/n \cdot n/2} = e^{\pi i} = -1 = e^{2\pi i/2} = \omega_2$

30.2-5 Consider the polynomial $A(x) = \sum_{k=0}^{n-1} a_k x^k$ with degree $n-1$ and write it in the following form:

$$A(x) = A^0(x^3) + xA^1(x^3) + x^2A^2(x^3)$$

where

$$A^j(x) = \sum_{k=0}^{n/3-1} a_{3k+j}x^k, \qquad j = 0, 1, 2.$$

The goal is to evaluate $A(x)$ at the points $\omega_n^0, \omega_n^1, \ldots, \omega_n^{n-1}$. Since

$$A(\omega_n^i) = A^0(\omega_n^{3i}) + \omega_n^i A^1(\omega_n^{3i}) + \omega_n^{2i} A^2(\omega_n^{3i})$$

and $\omega_n^{3i} = \omega_{n/3}^i$, it is enough to evaluate $A^j(x)$ in $\omega_{n/3}^i$ for $i = 0, 1, \ldots, n/3-1$ and $j = 0, 1, 2$ and then one can compute required values $A(\omega_n^i)$ $(i = 0, 1, \ldots, n-1)$ in $\mathcal{O}(n)$ time.
The whole procedure is given by the following pseudocode:

Algorithm 1 RecFFT3($\mathbf{a}$)

Require: $\mathbf{a} = (a_0, a_1, \ldots, a_{n-1})$, the coefficients of the polynomial $A(x)$.
if $n = 1$ **then**
  return a_0
end if
$\omega_n := e^{2\pi i/n}, \quad \omega := 1$
$\mathbf{a}^0 := (a_0, a_3, \ldots, a_{n-3}), \quad \mathbf{y}^0 := \text{RecFFT}(\mathbf{a}^0)$
$\mathbf{a}^1 := (a_1, a_4, \ldots, a_{n-2}), \quad \mathbf{y}^1 := \text{RecFFT}(\mathbf{a}^1)$
$\mathbf{a}^2 := (a_2, a_5, \ldots, a_{n-1}), \quad \mathbf{y}^2 := \text{RecFFT}(\mathbf{a}^2)$
for $k = 0$ to $n-1$ **do**
  $k' := k \bmod n/3$ (corresponds to $\omega_n^{3k} = \omega_{n/3}^k = \omega_{n/3}^{k \bmod n/3}$)
  $y_k := y_{k'}^0 + \omega \cdot y_{k'}^1 + \omega^2 \cdot y_{k'}^2$
  $\omega := \omega\omega_n$
end for
return $\mathbf{y} = (y_0, y_1, \ldots, y_{n-1})$

Denote by $T(n)$ the time complexity of the previous algorithm. Since there are 3 recursive calls and additional $\mathcal{O}(n)$ operations, the recurrence for $T(n)$ is given by:

$$T(n) = 3T(n/3) + \mathcal{O}(n)$$

which solution is given by $\mathcal{O}(n \log_3 n)$.

30.2-7 The idea is to divide the roots $(z_0, z_1, \ldots, z_{n-1})$ in two halves, compute the polynomial for both halves recursively, and then multiply obtained polynomials using FFT based multiplication.

Algorithm 2 PolyFromRoots($\mathbf{z}$, x)

Require: $\mathbf{z} = (z_0, z_1, \dots, z_{n-1})$ and symbolic variable x.

$k := \lfloor n/2 \rfloor$

$\mathbf{z}' := (z_0, z_1, \dots, z_{k-1})$

$\mathbf{z}'' := (z_k, z_{k+1}, \dots, z_{n-1})$

$P'(x) :=$ PolyFromRoots$(\mathbf{z}', x)$

$P''(x) :=$ PolyFromRoots$(\mathbf{z}'', x)$

return FFT-MUL$(P'(x), P''(x))$

Note that the time complexity $T(n)$ of the previous algorithm satisfies

$$T(n) = 2T(n/2) + \mathcal{O}(n \lg n)$$

where $n \lg n$ term comes from FFT based multiplication. For the sake of simplicity, assume that $n = 2^k$. Then

$$\begin{aligned} T(2^k) &= 2T(2^{k-1}) + C \cdot k \cdot 2^k \\ &= 4T(2^{k-2}) + C \cdot (k-1)2^k + k \cdot 2^k \\ &\vdots \\ &= 2^k T(1) + C \cdot (1 + 2 + \ldots + k) \cdot 2^k \\ &= 2^k T(1) + C \cdot 2^{k-1} k(k+1) \end{aligned}$$

implying $T(2^k) = \mathcal{O}(2^k k^2) = \mathcal{O}(n \lg^2 n)$.