Chapter 30

Michelle Bodnar, Andrew Lohr

April 20, 2016

Exercise 30.1-1

$$(56)x^6 - 8x^5 + (8-42)x^4 + (-80+6+21)x^3 + (-3-6)x^2 + (60+3)x - 30$$

which is

$$56x^6 - 8x^5 - 34x^4 - 53x^3 - 9x^2 + 63x - 30$$

Exercise 30.1-2

Let A be the matrix with 1's on the diagonal, $-x_0$'s on the super diagonal, and 0's everywhere else. Let q be the vector $(r, q_0, q_1, \ldots, x_{n-2})$. If $a = (a_0, a_1, \ldots, a_{n-1})$ then we need to solve the matrix equation $Aq = a$ to compute the remainder and coefficients. Since A is tridiagonal, Problem 28-1 (e) tells us how to solve this equation in linear time.

Exercise 30.1-3

For each pair of points, $(p, A(p))$, we can compute the pair $(\frac{1}{p}, A^{rev}(\frac{1}{p}))$. To do this, we note that $A^{rev}(\frac{1}{p}) = \sum_{j=0}^{n-1} a_{n-1-j} \left(\frac{1}{p}\right)^j = \sum_{j=0}^{n-1} a_j \left(\frac{1}{p}\right)^{n-1-j} = p^{1-n} \sum_{j=0}^{n-1} a_j p^j = p^{1-n} A(p)$ since we know what $A(p)$ is, we can compute $A^{rev}(\frac{1}{p})$ of course, we are using the fact that $p \neq 0$ because we are dividing by it. Also, we know that each of these points are distinct, because $\frac{1}{p} = \frac{1}{p'}$ implies that $p = p'$ by cross multiplication. So, since all the x values were distinct in the point value representation of A, they will be distinct in this point value representation of A^{rev} that we have made.

Exercise 30.1-4

Suppose that just $n-1$ point-value pairs uniquely determine a polynomial P which satisfies them. Append the point value pair (x_{n-1}, y_{n-1}) to them, and let P' be the unique polynomial which agrees with the n pairs, given by Theorem 30.1. Now append instead (x_{n-1}, y'_{n-1}) where $y_{n-1} \neq y'_{n-1}$, and let P'' be the

polynomial obtained from these points via Theorem 30.1. Since polynomials coming from n pairs are unique, $P' \neq P''$. However, P' and P'' agree on the original $n-1$ point-value pairs, contradicting the fact that P was determined uniquely.

Exercise 30.1-5

First, we show that we can compute the coefficient representation of $\prod_j (x - x_j)$ in time $\Theta(n^2)$. We will do it by recursion, showing that multiplying $\prod_{j<k}(x - x_j)$ by $(x - x_k)$ only takes time $O(n)$, since this only needs to be done n times, this gets is total runtime of $O(n)$. Suppose that $\sum_{i=0}^{k-1} k_i x^i$ is a coefficient representation of $\prod_{j<k}(x - x_j)$. To multiply this by $(x - x_k)$, we just set $(k+1)_i = k_{i-1} - x_k k_i$ for $i = 1, \ldots k$ and $(k+1)_0 = -x_k \cdot k_0$. Each of these coefficients can be computed in constant time, since there are only linearly many coefficients, then, the time to compute the next partial product is just $O(n)$.

Now that we have a coefficient representation of $\prod_j (x - x_j)$, we need to compute, for each k $\prod_{j-k}(x - x_j)$, each of which can be computed in time $\theta(n)$ by problem 30.1-2. Since the polynomial is defined as a product of things containing the thing we are dividing by, we have that the remainder in each case is equal to 0. Lets call these polynomials f_k. Then, we need only compute the sum $\sum_k y_k \frac{f_k(x)}{f_k(x_k)}$. That is, we compute $f(x_k)$ each in time $\Theta(n)$, so all told, only $\Theta(n^2)$ time is spent computing all the $f(x_k)$ values. For each of the terms in the sum, dividing the polynomial $f_k(x)$ by the number $f_k(x_k)$ and multiplying by y_k only takes time $\Theta(n)$, so total it takes time $\Theta(n^2)$. Lastly, we are adding up n polynomials, each of degree bound $n-1$, so the total time taken there is $\Theta(n^2)$.

Exercise 30.1-6

If we wish to compute P/Q but Q takes on the value zero at some of these points, then we can't carry out the "obvious" method. However, as long as all point value pairs (x_i, y_i) we choose for Q are such that $y_i \neq 0$, then the approach comes out exactly as we would like.

Exercise 30.1-7

For the set A, we define the polynomial f_A to have a coefficient representation that has a_i equal zero if $i \notin A$ and equal to 1 if $i \in A$. Similarly define f_B. Then, we claim that looking at $f_C := f_A \cdot f_B$ in coefficient form, we have that the ith coefficient, c_i is exactly equal to the number of times that i is realized as a sum of elements from A and B. Since we can perform the polynomial multiplication in time $O(n \lg(n))$ by the methods of this chapter, we can get the final answer in time $O(n \lg(n))$. To see that f_C has the nice property described, we'll look at the ways that we could end up having a term of x^i appear. Each contribution to that coefficient must come from there being some k so that $a_k \neq 0$ and $b_{i-k} \neq 0$, because the powers of x attached to each are additive when we

multiply. Since each of these contributions is only ever 1, the final coefficient is counting the total number of such contributions, therefore counting the number of $k \in A$ such that $i - k \in B$, which is exactly what we claimed f_C was counting.

Exercise 30.2-1

$$\omega_n^{n/2} = \left(e^{2\pi i/n}\right)^{n/2} = e^{\pi i} = -1 = \omega_2$$

Exercise 30.2-2

The DFT is $(6, -2i - 2, -2, 2i - 2)$.

Exercise 30.2-3

We want to evaluate both of the functions at the fourth roots of unity, that is, $\pm 1, \pm i$. We have an initial call of $RECURSIVE-FFT((-10, 1, -1, 7, 0, 0, 0, 0))$. This causes a call of $RECURSIVE-FFT((-10, -1, 0, 0))$, which evaluates to $(-11, -10-i, -9, -10+i)$. It also causes a call of $RECURSIVE-FFT((1, 7, 0, 0))$ which returns $(8, 1+7i, -6, 1-7i)$. Now, in evaluating the original function call, we have $y_0 = -11+8 = -3$, $y_4 = -19$. Then, we change ω to $\omega_8 = \frac{1+i}{\sqrt{2}}$, and have $y_1 = -10-i+\frac{1+i}{\sqrt{2}}(1+7i) = -10-i-3\sqrt{2}+4\sqrt{2}i$ and $y_5 = -10-i+3\sqrt{2}-4\sqrt{2}$. At the next value of k, we get $y_2 = -9-6i$ and $y_6 = -9+6i$. Lastly, we compute $y_3 = -10 + i + 3\sqrt{2} + 4\sqrt{2}i$ and $y_7 = -10 + i - 3\sqrt{2} - 4\sqrt{2}i$. So, the vector returned is $(-3, -10 - i - 3\sqrt{2} + 4\sqrt{2}i, -9 - 6i, -10 + i + 3\sqrt{2} + 4\sqrt{2}i, -19, -10 - i + 3\sqrt{2} - 4\sqrt{2}i, -9 + 6i, -10 + i - 3\sqrt{2} - 4\sqrt{2}i)$. Similarly, if we wanted to compute the FFT of the other polynomial, we'd get the FFT of B is given by $(5, 3-7\sqrt{2}+\sqrt{2}i, 3-14i, 3+7\sqrt{2}+\sqrt{2}i, 1, 3+7\sqrt{2}-\sqrt{2}i, 3+14i, 3-7\sqrt{2}-\sqrt{2}i)$. Then, we just multiply together these point value representations to get that the product of A and B has the point value representation of

$$\begin{aligned}
&(-15,\\
&4 + 62\sqrt{2} + (-65 + 9\sqrt{2})i,\\
&-111 + 108i,\\
&4 - 62\sqrt{2} + (65 + 9\sqrt{2})i,\\
&-19,\\
&4 - 62\sqrt{2} + (-65 + -9\sqrt{2})i,\\
&-111 - 108i,\\
&4 + 62\sqrt{2} + (65 - 9\sqrt{2})i)
\end{aligned}$$

Interpolating this polynomial using equation (30.11), we get

$$
\begin{aligned}
a_0 =& \frac{1}{8}\sum_{k=0}^{7} y_k = -30 \\
a_1 =& \frac{1}{8}\sum_{k=0}^{7} y_k \omega_8^{-k} = 63 \\
a_2 =& \frac{1}{8}\sum_{k=0}^{7} y_k \omega_8^{-2k} = -9 \\
a_3 =& \frac{1}{8}\sum_{k=0}^{7} y_k \omega_8^{-3k} = -53 \\
a_4 =& \frac{1}{8}\sum_{k=0}^{7} y_k \omega_8^{-4k} = -34 \\
a_5 =& \frac{1}{8}\sum_{k=0}^{7} y_k \omega_8^{-5k} = -8 \\
a_6 =& \frac{1}{8}\sum_{k=0}^{7} y_k \omega_8^{-6k} = 56 \\
a_7 =& \frac{1}{8}\sum_{k=0}^{7} y_k \omega_8^{-7k} = 0
\end{aligned}
$$

Giving us the polynomial

$$56x^6 - 8x^5 - 34x^4 - 53x^3 - 9x^2 + 63x - 30$$

The same as in problem 30.1-1.

Exercise 30.2-4

Exercise 30.2-5

To show that our algorithm for n being a power of 3 works, we will first prove an analogue of the halving lemma. In particular, for n a power of 3, that the cube of the n^{th} complex roots of unity are the $n/3$ complex $(n/3)$th roots of unity. First, we note that by the cancellation lemma, $(\omega_n^k)^3 = \omega_{n/3}^k$.

$$
\begin{aligned}
(\omega_n^{k+n/3})^3 =& \omega_n^{3k+n} \\
=& (\omega_n^k)^3
\end{aligned}
$$

Now, we write $A(x) = \sum_{j=0}^{n-1} a_j x^j$, and define $A^{[i]} = \sum_{j=0}^{n/3-1} a_{i+3j} x^j$ for $i =$

Algorithm 1 RECURSIVE-FFT-INV(y)

```
1: n = y.length
2: if n == 1 then
3:     return y/n
4: end if
5: w_n = e^{2πi/n}
6: w = 1/n
7: y^[0] = (y_0, y_2, ..., y_{n-2})
8: y^[1] = (y_1, y_3, ..., y_{n-1})
9: a^[0] = RECURSIVE-FFT-INV(y^[0])
10: a^[1] = RECURSIVE-FFT-INV(y^[1])
11: for k = 0 to n/2 − 1 do
12:     a_k = a_k^[0] + w a_k^[1]
13:     a_{k+(n/2)} = a_k^[0] − w a_k^[1]
14:     w = w w_n
15: end for
16: return a
```

$1, 2, 3$. Then, we can see that

$$A(x) = A^{[0]}(x^3) + xA^{[1]}(x^3) + x^2A^{[2]}(x^3)$$

The recurrence we get is

$$\begin{aligned} T(n) &= 3T(n/3) + \Theta(n) \\ &= \Theta(n \lg(n)) \end{aligned}$$

Exercise 30.2-6

It will suffice to show that each power of omega is distinct and that our new definition satisfies all the relevant properties of omega that made DFT and inverse DFT work in the first place. To start, note that $2^t, 2^{2t}, \ldots, 2^{nt/2}$ are all distinct and even since they are each less than m. Further, for $1 \leq k < n/2$ we have that $2^{kt}2^{nt/2}$ is equivalent to -2^{kt}, which is equivalent to $2^{nt/2} + 1 - 2^{kt}$ which is odd, so we conclude that all powers of ω are distinct, $\omega^n = 2^n t = (-1)^2 = 1$, $\omega^{k+n/2} = -\omega^k$. Finally, $\sum_{i=0}^{n-1} \omega^i = \sum_{i=0}^{n-1} 2^{ti} = (1 - 2^{tn})/(1 - 2^t) = 0$. Thus, our new definition of ω satisfies all the requisite properties to make the DFT work, and since the values are distinct it is well-defined and behaves identically.

Exercise 30.2-7

We just do a bunch of multiplications. More seriously, let $P_{i,0}(x) = (x - z_{i-1})$ for $i = 1, \ldots, n$. Then, we compute the following products, $P_{i,k} = P_{i,k-1} \cdot P_{2i,k-1}$, for any $i \leq n/(2^k)$. If we ever index outside of the already defined $P_{i,k}$ values, we pretend that the value we get is 1. Then, our final answer will

Algorithm 2 POW3FFT(a)

n = a.length
if n==1 **then**
  return a
end if
$\omega_n = e^{2\pi i/n}$
$\omega = 1$
$a^{[0]} = (a_0, a_3, a_6 \ldots, a_{n-3})$
$a^{[1]} = (a_1, a_4, a_7 \ldots, a_{n-2})$
$a^{[2]} = (a_2, a_5, a_8 \ldots, a_{n-1})$
$y^{[0]} = POW3FFT(a^{[0]})$
$y^{[1]} = POW3FFT(a^{[1]})$
$y^{[2]} = POW3FFT(a^{[2]})$
for k=0,1...n/2-1 **do**
  $y_k = y_k[0] + \omega y_k^{[1]} + \omega^2 y_k^{[2]}$
  $y_{k+n/3} = y_k[0] + \omega_3 \omega y_k^{[1]} + \omega_3^2 \omega^2 y_k^{[2]}$
  $y_{k+2n/3} = y_k[0] + \omega_3^2 \omega y_k^{[1]} + \omega_3 \omega^2 y_k^{[2]}$
  $\omega = \omega \omega_n$
end for
return y

be $P_{1,\lfloor \lg(n) \rfloor + 1}$. We have that obtaining a polynomial in this way that has the recurrence where we n represents the time required to do it for a polynomial of degree bound n that has zeroes at n given points.

$$T(n) = 2T(n/2) + \Theta(n \lg(n))$$

Which, we know by exercise 4.6-2, has a solution of $T \in \Theta(n \lg^2(n))$.

Exercise 30.2-8

Define a polynomial $P(x)$ of degree bound $2n$ by $P(x) = \sum_{j=0}^{2n-1} b_j x^j$ where $b_j = a_j z^{j^2/2}$ if $j \leq n-1$ and 0 for $j \geq n$. Define $Q(x) = \sum_{j=0}^{2n-1} c_j x^j$ where $c_j = z^{-j^2}/2$. We can compute their product in time $O(2n \lg 2n) = O(n \lg n)$. If we let d_k be the coefficient on x^k in their product, for $k \geq n$ we have

$$d_k = \sum_{j=0}^{k} b_j c_{k-j} = \sum_{j=0}^{n-1} \left(a_j z^{j^2/2} \right) \left(z^{-(k-j)^2}/2 \right).$$

By setting $y_k = z^{k^2/2} d_k$ in linear time, we can compute the chirp transform in $O(n \lg n)$.

Exercise 30.3-1

By calling BIT-REVERSE-COPY, we get that $A = (0, 4, 3, 7, 2, 5, -1, 9)$. after the first pass of the outermost, loop, when $s = 1$, we have that the array is $A = (4, -4, 10, -4, 7, -3, 8, -10)$. The value at the end of the next iteration of the outermost loop is $(14, -3 - 3i, -8, -3 - 3i, 15, 3 + 4i, -13, 3 - 4i)$. Then, on the last iteration, we get our final answer of

$$\begin{aligned} A = (19,& \\ & -4 - 4i + \frac{7}{\sqrt{2}} - \frac{13}{\sqrt{2}}i, \\ & -6 + i, \\ & -4 + 4i - \frac{7}{\sqrt{2}} - \frac{13}{\sqrt{2}}i, \\ & -1, \\ & -4 - 4i - \frac{7}{\sqrt{2}} + \frac{13}{\sqrt{2}}i, \\ & -6 - i, \\ & -4 + 4i + \frac{7}{\sqrt{2}} + \frac{13}{\sqrt{2}}i) \end{aligned}$$

Exercise 30.3-2

We can consider ITERATIVE-FFT as working in two steps, the copy step (line 1) and the iteration step (lines 4 through 14). To implement in inverse iterative FFT algorithm, we would need to first reverse the iteration step, then reverse the copy step. We can do this by implementing the INVERSE-INTERATIVE-FFT algorithm exactly as we did with the FFT algorithm, but this time creating the iterative version of RECURSIVE-FFT-INV as given in exercise 30.2-4.

Exercise 30.3-3

It computes a twiddle factor for each iteration of the innermost for loop. Since there are n/m iterations of the loop on line 6 and, for each $m/2$ iterations of the innermost loop, there are a total of $n/(2^s) \cdot 2^{s-1} = n/2$ twiddle factors. If we, before line 6 compute all of the powers $< m/2$ of ω_m, we won't have to do any computation of them later on. These are the only twiddle factors that will show up, and so, we only compute $m/2 = s^2/2 = 2^{s-1}$ of them.

Exercise 30.3-4

First observe that if the input to a butterfly operation is all zeros, then so is the output. Our initial input will be $(a_0, a_1, \ldots, a_{n-1})$ where $a_i = 0$ if $0 \leq i \leq n/2 - 1$, and i otherwise. By examining the diagram on page 919, we

see that the zeros will propogate as half the input to each butterfly operation in the final stage. Thus, if any of the output values are 0 we know that one of the y_k's is zero, where $k \geq n/2 - 1$. If this is the case, then the faulty butterfly added occurs along the wire connected to y_k. If none of the output values are zero, the faulty butterfly adder must occur as one which only deals with the first $n/2 - 1$ wires. In this case, we rerun the test with input such that $a_i = i$ if $0 \leq i \leq n/2 - 1$ and 0 otherwise. This time, we will know for sure which of the first $n/2 - 1$ wires touches the faulty butterfly adder. Since only $\lg n$ adders touch a given wire, we have reduced the problem of size n to a problem of size $\lg n$. Thus, at most $2 \lg^\star n = O(\lg^\star n)$ tests are required.

Problem 30-1

a. Similar to problem 4.2-7,

$$(a + b)(c + d) = ac + cb + ad + cd$$

So, we compute that product, we also compute ac and cd. This gets us the the product of the two polynomials is

$$acx^2 + ((a + b)(c + d) - ac - cd)x + cd$$

b. Assume that n is a power of two, if it isn't, then just bump it up to the nearest power of two, since the degree bound can be higher than the degree of the polynomials. Suppose that we want to multiply the polynomials $A_1(x) = \sum_{j=1}^{n-1} a_{j,1}x^j$ and $A_2(x) = \sum_{j=1}^{n-1} a_{j,2}x^j$.

In the first method, we'll set $H_i(x) = \sum_{j=\frac{n}{2}}^{n-1} a_{j,i}x^j$ and $L_i(x) = \sum_{j=0}^{n/2-1} a_{j,i}x^j$ for $i = 1, 2$. Then, we have that $A_i(x) = H_i(x)x^{n/2} + L_i(x)$ for $i = 1, 2$. Then, by the method of the first part of this problem, we have that

$$\begin{aligned} A_1(x) \cdot A_2(x) = {} & (H_1(x) \cdot H_2(x))x^n \\ & + ((H_1(x) + L_1(x)) \cdot (H_2(x) + L_2(x)) - H_1(x) \cdot H_2(x) - L_1(x) \cdot L_2(x))x^{n/2} \\ & + L_1(x) \cdot L_2(x) \end{aligned}$$

So, the runtime of this procedure for degree bound n is, by the master theorem:

$$\begin{aligned} HL(n) &= 3HL(n/2) + \Theta(n) \\ &= \Theta(n^{\lg(3)}) \end{aligned}$$

Now, for the second method, we write $O_i(x) = \sum_{j=0}^{n/2-1} a_{2j+1,i}x^j$ and $E_i(x) = \sum_{j=0}^{n/2-1} a_{2j,i}x^j$ for $i = 1, 2$. Then, we have that for both values of i, $A_i = xO_i(x^2) + E_i(x^2)$. So,

$$\begin{aligned} A_1(x) \cdot A_2(x) =& x^2(O_1(x^2) \cdot O_2(x^2)) \\ &+ x((O_1(x^2) + E_1(x^2))(O_2(x^2) + E_1(x^2)) - O_1(x^2) \cdot O_2(x^2) - E_1(x^2) \cdot E_2(x^2)) \\ &+ E_1(x^2) \cdot E_2(x^2) \end{aligned}$$

So, again, we only need to do three multiplies, each with a degree bound of half. So, the runtime for this, call it $OE(n)$ is

$$\begin{aligned} OE(n) &= 3OE(n/2) + \Theta(n) \\ &= \Theta(n^{\lg(n)}) \end{aligned}$$

c. Suppose that we want to multiply two integers $A_1 = \sum_{k=0}^{\lfloor \lg(A_1) \rfloor} a_{k,1}2^k$ and $A_2 = \sum_{k=0}^{\lfloor \lg(A_2) \rfloor} a_{k,2}2^k$. Then, we'll associate to these polynomials $f_i = \sum_{k=0}^{\lfloor \lg(A_i) \rfloor} a_{k,i}x^i$. Then, we exactly have that $f_i(2) = A_i$. So, to find $A_1 \cdot A_2$, all we need do is multiply the polynomials f_1 and f_2 and evaluate their product at 2. Since both of their degrees are bounded by n, we can multiply them in time $\Theta(n^{\lg(3)})$ by the previous part. evaluating them also only takes linear time, so doesn't change the total runtime.

Problem 30-2

a. The sum of two Toeplitz matrices is Toeplitz, but the product is not.

b. Let A be a Toeplitz matrix. We can use a vector of length $2n-1$ to represent it, given by: $(c_0, \ldots, c_{2n-2}) = (a_{n,1}, a_{n-1,1}, \ldots, a_{2,1}, a_{1,1}, a_{1,2}, \ldots, a_{a,n})$. To add two Toeplitz matrices, simply add their associated vectors.

c. We can interpret this as the multiplication of two polynomials. Specifically, let $P(x) = c_0 + c_1x + \ldots + c_{2n-1}x^{2n-2}$. Let $(b_0, b_1, \ldots, b_{n-1})$ be the vector of length n by which we wish to multiply, and let y_k denote the k^{th} entry of the vector which results from the multiplication. Let $Q(x) = b_0x^{n-1} + b_1x^{n-2} + \ldots + b_{n-1}$. Then the coefficient on $x^{n-k+n-1}$ in $P(x)Q(x)$ is given by $\sum_{i=0}^{n-1} c_{n-k+i}b_i = \sum_{i=0}^{n-1} a_{k,i}b_i = y_k$. Since we can multiply the polynomials in $O(n \lg n)$ and the needed results are just some of the coefficients, we can multiply a Toeplitz matrix by an n-vector in $O(n \lg n)$.

d. We can view matrix multiplication as simply multiplication by an n vector carried out n times. If b_j is the j^{th} column of the second matrix, then Ab_j is the j^{th} column of the resulting matrix. By part c, this can be done in $O(n^2 \lg n)$ time, which is asymptotically faster than even Strassen's algorithm.

Problem 30-3

a.

$$\begin{aligned} y_{k_1,\ldots,k_d} &= \sum_{j_1=0}^{n_1-1} \cdots \sum_{j_d=0}^{n_d-1} a_{j_1,\ldots j_d} \omega_{n_1}^{j_1 k_1} \cdots \omega_{n_d}^{j_d k_d} \\ &= \sum_{j_d=0}^{n_d-1} \cdots \sum_{j_1=0}^{n_1-1} a_{j_1,\ldots j_d} \omega_{n_1}^{j_1 k_1} \cdots \omega_{n_d}^{j_d k_d} \\ &= \sum_{j_d=0}^{n_d-1} \cdots \sum_{j_2=0}^{n_2-1} \left(\sum_{j_1=0}^{n_1-1} a_{j_1,\ldots j_d} \omega_{n_1}^{j_1 k_1} \right) \omega_{n_2}^{j_2 k_2} \cdots \omega_{n_d}^{j_d k_d} \end{aligned}$$

So, the thing inside the parentheses is a one dimensional Fourier transform that must be computed for every possible term of the outer sums, that is, must be computed $n_2 n_3 \cdots n_d = n/n_1$ times because in each term the a values might be different. Once that is computed we've decreased the number of sums by 1. This means that by keeping on applying this, we can keep decreasing the number of dimensions until the problem is solved. We are actually only needing to do $n/(\prod_{i \le k} n_i)$ of the DFT's along dimension k instead of the larger number stated in the problem of n/n_i.

b. We can exchange the order of summation however we please because none of the indices of summation ever appear in the bounds for a different summation sign.

c. The time to do each DFT along the kth dimension is $O(n_k \lg(n_k))$, since we only need to do it at most $n/(\prod_{j<k} n_j)$ times, the runtime of all of them in dimension k is at most $O(n/(\prod_{j<k} n_j) \lg(n_k))$. Also, note that we may assume that all of the n_k values are at least two, because otherwise doing that DFT would be trivial. So, the total time is on the order of

$$\begin{aligned} \sum_{k=1}^{d} n/\left(\prod_{j<k} n_j\right) \lg(n_k) &\le \lg(n) \sum_{k=1}^{d} n/\left(\prod_{j<k} n_j\right) \\ &\le \lg(n) \sum_{k=1}^{d} n/2^{k-1} \\ &< n \lg(n) \end{aligned}$$

Which is independent of d.

Problem 30-4

a. It is straightforward to check that $A^{(t)}(x_0) = t!b_t$. Since $A^{(t+1)}(x_0) = (t+1)A^{(t)}(x_0)b_{t+1}/b_t$, we can compute each next term in constant time from the previous, so computing $A^{(t)}(x_0)$ for $t = 0, 1, \ldots, n-1$ takes $O(n)$ time.

b. We just need to perform an inverse FFT procedure on the n point-value pairs $A(x_0 + w_n^k) = \sum_{j=0}^{n-1} b_j w_n^{kj}$. This takes $O(n \lg n)$.

c. Let $\chi(x) = 1$ if $x \geq 0$ and 0 otherwise. Using the binomial theorem we have

$$\begin{aligned}
A(x_0 + w_n^k) &= \sum_{j=0}^{n-1} a_j \sum_{r=0}^{j} \binom{j}{r} w_n^{kr} x_0^{j-r} \\
&= \sum_{j=0}^{n-1} a_j \sum_{r=0}^{n-1} \frac{j!}{r!(j-r)!} w_n^{kr} x_0^{j-r} \chi(j-r) \\
&= \sum_{r=0}^{n-1} \frac{w_n^{kr}}{r!} \sum_{j=0}^{n-1} a_j j! \frac{x_0^{j-r}\chi(j-r)}{(j-r)!} \\
&= \sum_{r=0}^{n-1} \frac{w_n^{kr}}{r!} \sum_{j=0}^{n-1} f(j)g(r-j).
\end{aligned}$$

d. Let $s(r) = \sum_{j=0}^{n-1} f(j)g(r-j)$ in $O(n)$ time. Letting $b_j = \frac{s(r)}{r!}$. Then we need only compute the DFT of the coefficient vector $(b_0, b_1, \ldots, b_{n-1})$, which can be done in $O(n \lg n)$. By part b, once we have these evaluations we can compute the derivatives in $O(n \lg n)$ time as well.

Problem 30-5

a. First, note that because the degree of $x - z$ is one, $A(x) mod (x - z)$ will be a constant. By the definition of modular arithmetic for polynomials(or any Euclidean Domain), there is some polynomial $f(x)$ so that $A(x) = f(x)(x - z) + (A(x) \mod (x - z))$. So, if we evaluate this expression at z, the first term goes to zero, and we have that $A(z) = A(x) \mod (x - z)$.

b. $P_{kk}(x) = (x - x_k)$ so, by the previous part, $Q_{kk} = A(x_k)$. The degree of $P_{0,n-1}$ is equal to n which is higher than the degree of A. Therefore modding out by it doesn't change the value value of $A(x)$ at all. That is, if we were to write $A(x) = f(x)P_{0,n-1} + Q_{0,n-1}$, the only acceptable value of $f(x)$ is zero, otherwise there would be a too high degree term on the right.

c. Suppose we write $A(x) = f_1(x)P_{ij}(x) + Q_{ij}(x)$. Then, we take $Q_ij = f_2(x)P_{ik} + (Q_{ij}(x) \mod P_{ik})$. Since we have that P_{ik} is a product over a smaller set of irreducible factors than P_{ij}, we can write $A(x) = (f_1(x) \prod_{\ell=k+1}^{j}(x - x_\ell) + f_2)P_{ik} + (Q_{ij}(x) \mod P_{ik})$. Since we can write it as a remainder of A

after dividing by P_{ik}, we have that $A \mod P_{ik} = Q_{ij}(x) \mod P_{ik}$, which is to say, $Q_{ik} = Q_{ij}(x) \mod P_{ik}$.

We basically do the same thing to show the other equality. Suppose that $Q_{ij} = f_3 P_{kj} + (Q_{ij}(x) \mod P_{kj})$, then $A(x) = (f_1 \prod_{\ell=i}^{\ell=k-1}(x - x_\ell) + f_3)P_{kj} + (Q_{ij}(x) \mod P_{kj})$ and so, $Q_{kj} = A \mod P_{kj} = (Q_{ij}(x) \mod P_{kj})$.

d. Initially, we know what the value of $Q_{0,n-1}$ is, since it is just $A(x)$. Suppose that n is a power of 2, since it makes the analysis easier to do, if it is not a power of two, then bump up the degree bound to the nearest value of 2, since we have that $(2n) + \lg^2(2n) \in O(n \lg^2(n))$, doing this increase of the degree bound will not change the asymptotics of the algorithm. Since we have that the number of points we are evaluating at is equal to the degree bound, just pick arbitrary points to pad the original set of points with and then disregard their values once they are computed. The idea is to cut in half $0, \ldots, n-1$, by computing $Q_{0,n/2-1}$ and $Q_{n/2,n-1}$ using the rule from the previous part until you arrive at having to compute Q_{ii} for some i, which by part b is equal to $A(x_i)$. Since the computing of each of the Q values before the end is only a matter of computing two polynomial remainders, the time to do this is given by the recurrence

$$T(n) = 2T(n/2) + \Theta(n \lg(n))$$

Even though the master theorem doesn't apply to this recurrence, exercise 4.6-2 tells us that $T \in \Theta(n \lg^2(n))$.

Problem 30-6

a. By the prime number theorem, the number of primes between 1 and N is approximately $\frac{1}{\ln(N)}$. This means that between 1 and $n \lg(n)$, we can expect that a random number will be prime with probability $\frac{1}{\lg(n \lg(n))} = \frac{1}{\lg(n)+\lg(\lg(n))} \approx \frac{1}{\lg(n)}$. Also, we are considering $\lg(n)$ numbers between 1 and $n \lg(n) + 1$ that are one more than a multiple of n, so, the expected number of prime numbers of that form with $k \leq \lg(n)$ is one, so, by the poisson paradigm, we would think that with probability at least about $\frac{1}{e}$, there is a prime number of that form with $k \leq \lg(n)$.

Sine the value of p is expected to be about $n \lg(n)$, it has a length of about $\lg(n \lg(n)) = \lg(n) + \lg(\lg(n))$.

b.

c.

d.