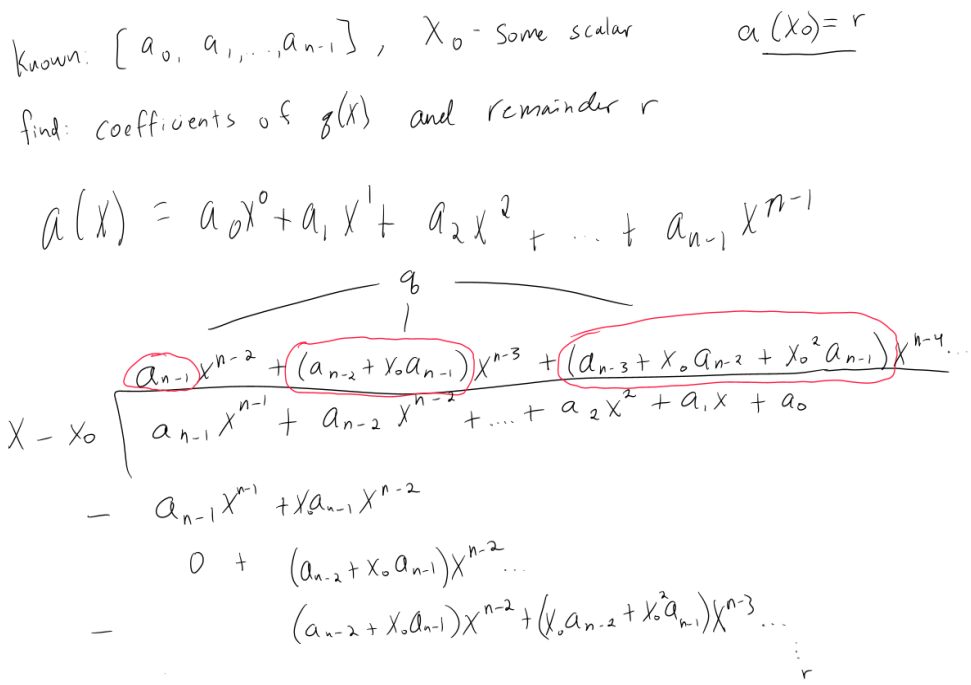
Kelsey Neis -neis@umn.edu

CSCI 5421 - Advanced Algorithms - HW1

30.1 - 2



Since the divisor $(x - x_0)$ does not have coefficients, we can assume q(x) will follow a certain pattern. It appears that the first coefficient q_0 (not shown above, because I reversed A(x)) will be the sum of all of A above it, multiplied by x_0*. I think the simplest way would be to go backwards to get the coefficients. Starting at A(n-1) for the value of the q(n-1) coefficient, then for subsequent values, multiply the previous coefficient by x_0 and then add A(i). Then the remainder r should equal $q_0 * X_0 + a_0$.

This should take θ(n) time, as the coefficient calculated for q(n-1) is reused to get the next one q(n-2), so it only requires iterating once through A.

3

$$A(x) = a_0x^0 + a_1x^1 + a_2x^2 + \dots + a_{n-1}x^{n-1}$$

$$A^{rev}(x) = a_{n-1}x^0 + a_{n-2}x^1 + a_{n-3}x^2 + \dots + a_0x^{n-1}$$

point-value of $A(x)$: $(x_0, y_0), (x_1, y_1), (x_2, y_2), \dots (x_{n-1}, y_{n-1})$

where $y_0 = a_0x_0^0 + a_1x_0^1 + a_2x_0^2 + \dots + a_{n-1}x_0^{n-1}$

$y_0^{rev} = a_0x_0^{n-1} + a_1x_0^{n-2} + a_2x_0^{n-3} + \dots + a_{n-1}x_0^0$

x_0^{n-1} x_0^{n-3} x_0^{n-5} $\dots$ $x_0^{-(n-1)}$

i : 0 1 2 $\dots$ $n-1$

$n-1-2i$

$n-1-2\cdot0$ $n-1-2\cdot1$ $n-1-2\cdot2$

$$a_i^{rev}(x) = \left(x_i, x^{n-1} \cdot a_i(x^{-2})\right)$$

4

Theorem: for any set of distinct n point-value pairs there is a unique polynomial $A(x)$ such that $y_k = A(x_k)$ for $k = 0, \dots, n$ (deg $< n$)

Vandermonde with $n-1$ points: $m \times n$

$$V \begin{pmatrix} 1 & x_0 & x_0^2 & \dots & x_0^{n-1} \\ 1 & x_1 & x_1^2 & \dots & x_1^{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n-2} & x_{n-2}^2 & \dots & x_{n-2}^{n-1} \end{pmatrix} \times \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_{n-1} \end{pmatrix} = \begin{pmatrix} y_0 \\ y_1 \\ \vdots \\ y_{n-2} \end{pmatrix}$$

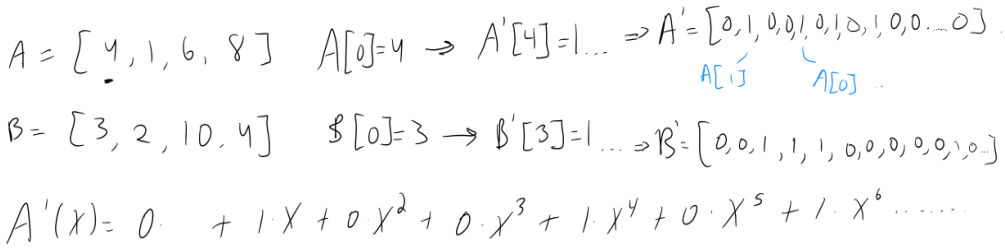
The above evaluation works correctly to get the values for (n-1) points. However, **V** is not square, and is therefore not invertible. To find the unique polynomial, we need all n coefficients, but we cannot get those with only n-1 point value pairs.

$a = V(X_0, X_,, \dots, X_{n-2})^{-1}y$ This equation can't be solved if **V** is not nxn

Further, if we were to add another point value pair, **a** would change, and if we assumed from the beginning that **a** was the unique polynomial of degree bound n, this would lead to a contradiction.

7

We need to convert A and B to coefficient arrays of length 20n, so the first step is to iterate through each set and represent each value in their corresponding 20n-length arrays. If the arrays can be initialized with 20 zeros in constant time, then this step will take time 2n.



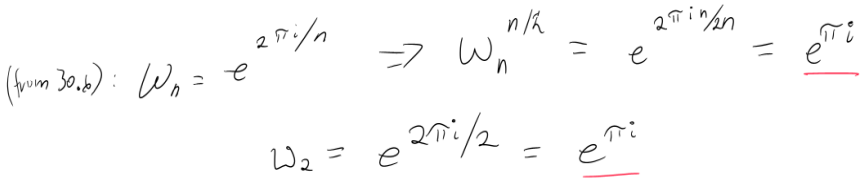
These will be the coefficient arrays, and using the fact that multiplying 2 powers of X adds together the powers, the power of the resulting C(X) will represent the sums we are trying to compute, and the coefficients will represent the number of times each sum appears in the Cartesian sum.

Now, we need to evaluate the A' and B' coefficients at the 2(th) root of unity by applying the FFT of order 2n to get a point-value representation. Evaluation takes nlogn time.

Then pointwise multiply the A'(x) by B'(X) and lastly, apply the FFT on the point value pairs of C(X) to get the coefficients. The resulting vector contains the number of times a given sum is computed from A and B at the index of that sum and from there it is straightforward to present the data.

This will take n + nlogn time, resulting in a total of θ(nlogn)

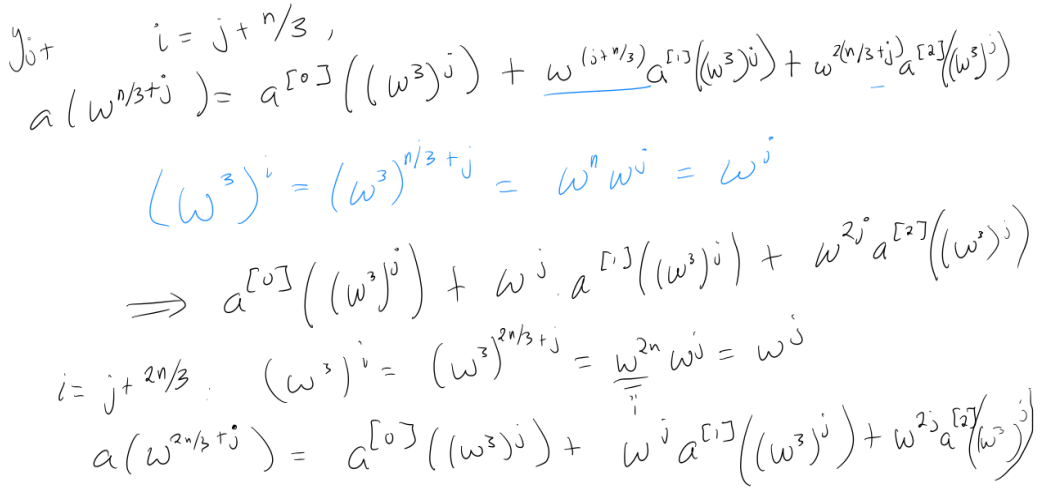
30.2 -1



5

If n is a power of 3, it is not as straightforward to divide the problem in to sub problems. We can't rely on the odd-even split, but could use the value of i % 3 to put each coefficient into a[0], a[1], or a[2]. If i % 3 = 0, it goes into a [0], if i % 3 = 1, it's part of a[1], otherwise, a [2]. Make 3 recursive calls to FFT with a[0],

a[1], and a[2]. Iterate i=0 to n /3 - 1, adding y[0] + wy[1] + w²y[2] to get the top third of the solution vector. To get the middle and bottom thirds, evaluate **a** at j+n/3 and j+2n/3:



Runtime:

$T(n) = 3T\left(\frac{n}{3}\right) + \Theta(n)$

using Master Theorem: a=3, b=3, f(n)=n

Matches case 2:

$f(n) = n^{\log_3 3}$

Therefore, the runtime is:

$T(n) = \Theta(n\log n)$

7

Given the polynomial P(X) must be a multiple of (x-zj), the remainder r of the following equation is 0:

$$P(X) = q(X)(X - z_j) + r$$
$$= q(X)(X - z_j)$$

We need to find the quotient that satisfies the above equation for all z.

If P(x) = 0 at only z_o...zn-1, then it must be composed only of multiples of X-Zj, meaning we can roughly represent it as follows:

$$P(X) = (X - z_0)(X - z_1)(X - z_2) \cdots (X - z_{n-1})$$

Because the Z_o term contains an x, the above polynomial is n + 1 bound. It only remains to get the coefficients now.

From there, we can easily get a point-value representation by choosing n integers not in Z_o...Zn and evaluating P(x) at those points.

To then get the coefficients of the polynomial, we need to interpolate, plugging in the point-value representation into the inverse FFT function to get the inverse DFT, which will be the vector of coefficients.