The Fast-fourier Transform

Disclaimer: *These abbreviated notes DO NOT substitute the textbook for this class. They should be used IN CONJUNCTION with the textbook and the material presented in class. If there is a discrepancy between these notes and the textbook, ALWAYS consider the textbook to be correct. Report such a discrepancy to the instructor so that he resolves it. These notes are only distributed to the students taking this class with A. Gerbessiotis in Spring 2012 ; distribution outside this group of students is NOT allowed.*

(c) Copyright A. Gerbessiotis. CS 667-101 : Spring 2012 . All rights reserved.

From Subject 3 (page 14) one way to multiply two polynomials $A(x), B(x)$ is standard polynomial multiplication. An alternative approach is the point-value based multiplication that uses interpolation. The question that arises is how one can evaluate $A(x) = a_0 + a_1x + a_2x^2 + \ldots + a_{n-1}x^{n-1}$ at the x_i points efficiently. One can formulate the evaluation problem as a matrix-vector multiplication problem.

$$\begin{aligned} A_0 &= a_0 + a_1x_0^1 + a_2x_0^2 + \ldots + a_{n-1}x_0^{n-1} \\ &\ldots \\ A_k &= a_0 + a_1x_k^1 + a_2x_k^2 + \ldots + a_{n-1}x_k^{n-1} \\ &\ldots \\ A_{n-1} &= a_0 + a_1x_{n-1}^1 + a_2x_{n-1}^2 + \ldots + a_{n-1}x_{n-1}^{n-1} \end{aligned}$$

Then $A_k = A(x_k)$, $0 \leq k \leq n-1$. The vector of $\vec{A} = (A_k)$ can be written in matrix form as follows in term of the vector $\vec{a} = (a_k)$.

$$\begin{bmatrix} A_0 \\ A_1 \\ \ldots \\ A_{n-1} \end{bmatrix} = \begin{bmatrix} 1 & x_0 & x_0^2 & \ldots & x_0^{n-1} \\ 1 & x_1 & x_1^2 & \ldots & x_1^{n-1} \\ \ldots & \ldots & \ldots & \ldots & \ldots \\ 1 & x_{n-1} & x_{n-1}^2 & \ldots & x_{n-1}^{n-1} \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \ldots \\ a_{n-1} \end{bmatrix} \Rightarrow \vec{A} = X\vec{a}.$$

Problem 1. Evaluation: Given input vector $\vec{a}$ compute output vector $\vec{A}$.
Problem 2. Interpolation: Given input vector $\vec{A}$ compute output vector $\vec{a}$.

(c) Copyright A. Gerbessiotis. All rights reserved.

In order to compute either $\vec{A}$ or $\vec{a}$ from the other we require the formation of matrix X or its inverse $inv(X) = X^{-1}$.

The entry of the i-th row and j-th column of X is $x_{ij} = x_i^j$ (column and row indices starting from zero) or $x_{ij} = x_{i-1}^{j-1}$ (column and row indices starting from one).

For X^{-1} we follow the discussion of Chapter 30, page 826. Matrix X is a Vandermonde matrix and its inversion is equivalent to solving Lagrange's formulae given also in Subject 3.

It has been shown in a homework problem that Lagrange's method can invert X (i.e. compute $\vec{a}$ from $\vec{A}$) in $O(n^2)$ time.

In the remainder we show that if the x_i's have some specific form this interpolation requires only $O(n \lg n)$ time, and the constant hidden in the big-Oh notation is well-defined and small (around 5).

For this to happen we choose the x_i's to be the n-th roots of unity.

(c) Copyright A. Gerbessiotis. CS 667-101 : Spring 2012 . All rights reserved.

A complex number w is an n-th root of unity if $w^n = 1$. It is a primitive (or principal) n-th root of unity if, in addition $w^m \neq 1$ for all $m = 1, \ldots, n-1$. From now on we are going to assume that n is a power of two.

Let w_n be the primitive (or principal) n-th root of unity,

$$w_n = cos(2\pi/n) + i\ sin(2\pi/n) = e^{2\pi i/n}.$$

A primitive n-th root of unity can generate the remaining roots by forming the products w_n^j. All n roots such formed are distinct. Some interesting properties of the roots of unity follow below.

Property 1. All the n roots of unity are distinct, i.e. $w_n^k \neq w_n^j$, $0 \leq k \neq j \leq n-1$.

Property 2 (Cancellation). For any n, k, d, we have that $w_{dn}^{dk} = w_n^k$.

Property 3. For any $n > 0$ $w_n^{n/2} = w_2 = -1$.

Property 4. If $n > 0$ is even, then the collection of w_n^{2j} is the set of the $n/2$-nd roots of unity (two n-th roots get mapped to one $n/2$-nd root).

Proof Sketch of Property 4. $(w_n^k)^2 = w_{n/2}^k$ by Property 2. Also consider $(w_n^{k+n/2})^2 = w_n^{2k+n} = w_n^n\ w_n^{2k} = w_n^{2k} = (w_n^k)^2$, i.e. two n-th roots whose indices are $n/2$ apart have the same square.

Property 5 (Summation) $\sum_{j=0}^{n-1}(w_n^k)^j = 0$.

Proof Sketch of Property 5.

$$\sum_{j=0}^{n-1}(w_n^k)^j \quad = \quad ((w_n^k)^n - 1)/(w_n^k - 1) = ((w_n^n)^k - 1)/(w_n^k - 1) = ((1)^k - 1)/(w_n^k - 1) = 0.$$

Note however that the sum $\sum_{j=0}^{n-1}(w_n^n)^j$ equals to 1. In general if $k \equiv 0 \pmod n$ the sum adds up to 1, otherwise if $k \not\equiv 0 \pmod n$ the sum adds up to 0.

(c) Copyright A. Gerbessiotis. CS 667-101 : Spring 2012 . All rights reserved.

Given a degree-bound n polynomial $A(x)$ we will evaluate it at the n-th roots of unity. Then $A_k = A(w_n^k)$, $0 \leq k \leq n-1$. Vector $\vec{A}$ can be written in matrix form as follows.

$$DFT_n(\vec{a}) = \vec{A} = \begin{bmatrix} A_0 \\ A_1 \\ \dots \\ A_{n-1} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & w_n & w_n^2 & \dots & w_n^{n-1} \\ \dots & \dots & \dots & \dots & \dots \\ 1 & w_n^{n-1} & w_n^{(n-1)\cdot 2} & \dots & w_n^{(n-1)(n-1)} \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \dots \\ a_{n-1} \end{bmatrix} = F_n(w_n) \times \begin{bmatrix} a_0 \\ a_1 \\ \dots \\ a_{n-1} \end{bmatrix} = F_n(w_n) \times \vec{a}.$$

Vector $\vec{A} = (A_k)$ is called the **Discrete Fourier Transform** of the coefficient vector $\vec{a} = (a_k)$. We sometimes write $\vec{A} = DTF(\vec{a}) = F_n(w_n) \times \vec{a}$, where $\times$ signifies matrix-vector multiplication. More frequently the $\times$ is omitted altogether and is implied.

Thus the relationship between vectors $\vec{A}$ and $\vec{a}$ is that through the $n \times n$ matrix $F_n(w_n)$, whose (i,j)-th entry is w_n^{ij} if i, j start from 0 (or $w_n^{(i-1)(j-1)}$ if i, j start from 1).

The inverse $DFT_n^{-1}(\vec{A})$ can be computed as follows

$$\vec{a} = DFT_n^{-1}(\vec{A}) = \frac{1}{n} F_n(w_n^{-1}) \times \vec{A}.$$

It is obvious that as a result of matrix-vector multiplication both the DFT and its inverse can be computed in $O(n^2)$ time. However DFT (and its inverse) can be computed efficiently through a method that is called FFT (Fast Fourier Transform) in $O(n \lg n)$ time.

(c) Copyright A. Gerbessiotis. All rights reserved.

The following theorem tells us how fast we can compute $\vec{A}$ from $\vec{a}$ using $F_n(w_n)$.

Theorem Tukey-Cooley (1965). *FFT, i.e. the computation of vector $\vec{A}$ such that $\vec{A} = F_n \times \vec{a}$ can be computed by $5n \lg n$ REAL operations provided that n is a power of two (i.e. $n = 2^m$).*

Note 1. If n is not a power of n we can always make n to be a power of two by padding (i.e. evaluating $f(x)$ in additional points for a larger n). Such an approach may double in the worst case the value of n, i.e. worsening the FFT computation by a multiplicative factor of approximately 2.

Proof of Tukey-Cooley theorem. FFT is a divide-and-conquer method. We split the polynomial A (whose coefficients form $\vec{a}$ into the odd-indexed and even-indexed terms. Let

$$\begin{aligned} A(x) &= a_0 + a_1 x + \ldots + a_{n-1}x^{n-1} = (a_0 + a_2 x^2 + \ldots + a_{n-2}x^{n-2}) + x(a_1 + x_3 x^2 + \ldots + a_{n-1}x^{n-2}) \\ &= L(x^2) + xR(x^2) \end{aligned}$$

where $L(x) = a_0 + a_2 x + \ldots + a_{n-2}x^{n/2-1}$, is the $(n/2-1)$-degree polynomial of the even coefficients, and $R(x) = a_1 + a_3 x + \ldots + a_{n-1}x^{n/2-1}$ is the $(n/2-1)$-degree polynonial of the odd coefficients of $A(x)$.

Note 2. $A(x)$ has n coefficients and $R(x), L(x)$ have $n/2$ coefficients each.

We will show that in order to compute $A_k = A(w_n^k)$ for $k = 0, \ldots, n-1$, it suffices to compute $L_k = L(w_{n/2}^k)$ and $R_k = R(w_{n/2}^k)$ for $k = 0, \ldots, n/2-1$ and then combine the L_k and R_k values to obtain the A_k and $A_{k+n/2}$ values.

If $T(n)$ is the time to determine all n A_k values, then $T(n/2)$ is the time to determine L_k or R_k. We will show that we can combine L_k, R_k into A_k and $A_{k+n/2}$ in additional $O(n)$ time.

Therefore the recurrence for the running time is $T(n) = 2T(n/2) + O(n)$ which leads to the $T(n) = O(n \lg n)$ time bound.

We first determine the "top-half" values $A_k = A(w_n^k)$, $0 \leq k \leq n/2 - 1$.

$$\begin{aligned} A_k = A(w_n^k) &= L((w_n^k)^2) + w_n^k R((w_n^k)^2) \\ &= L(w_n^{2k}) + w_n^k R(w_n^{2k}) \\ &= L(w_{n/2}^k) + w_n^k R(w_{n/2}^k) \\ &= L_k + w_n^k R_k \end{aligned} \tag{1}$$

Similarly, we then determine the "bottom-half" values $A_{k+n/2}$, $0 \leq k \leq n/2 - 1$, noting that $w_n^{n/2} = -1$ and $w_n^n = 1$.

$$\begin{aligned} A_{k+n/2} = A(w_n^{k+n/2}) &= L((w_n^{k+n/2})^2) + w_n^{k+n/2} R((w_n^{k+n/2})^2) \\ &= L(w_n^{2k+n}) + w_n^k w_n^{n/2} R(w_n^{2k+n}) \\ &= L(w_n^{2k}) - w_n^k R(w_n^{2k}) \\ &= L(w_{n/2}^k) - w_n^k R(w_{n/2}^k) \\ &= L_k - w_n^k R_k \end{aligned} \tag{2}$$

Note 3. A_k and $A_{k+n/2}$ that are $n/2$ indices apart are derived from the same pair L_k and R_k.

Note 4. Vector $\vec{A} = (A_k)$ can be computed from vectors $\vec{L} = (L_k)$ and $\vec{R} = (R_k)$ which are half its size.

Note 5. The computation of $\vec{L}$ and $\vec{R}$ involes an DFT computation involving $w_{n/2}^k$ the $n/2$-nd roots of unity.

In vector form we have therefore established the following.

(c) Copyright A. Gerbessiotis. All rights reserved.

By assumption we have that

$$\begin{bmatrix} A_0 \\ A_1 \\ \dots \\ A_{n-1} \end{bmatrix} = F_n(w_n) \begin{bmatrix} a_0 \\ a_1 \\ \dots \\ a_{n-1} \end{bmatrix}, where \begin{bmatrix} L_0 \\ L_1 \\ \dots \\ L_{\frac{n}{2}} \end{bmatrix} = F_{\frac{n}{2}}(w_{\frac{n}{2}}) \begin{bmatrix} a_0 \\ a_2 \\ \dots \\ a_{n-2} \end{bmatrix}, and \begin{bmatrix} R_0 \\ R_1 \\ \dots \\ R_{\frac{n}{2}} \end{bmatrix} = F_{\frac{n}{2}}(w_{\frac{n}{2}}) \begin{bmatrix} a_1 \\ a_3 \\ \dots \\ a_{n-1} \end{bmatrix}.$$

Then Eq. 1 and Eq. 2 can be rewritten in vector form as follows.

$$\begin{bmatrix} A_0 \\ A_1 \\ \dots \\ A_{\frac{n}{2}-1} \\ A_{\frac{n}{2}} \\ A_{\frac{n}{2}+1} \\ \dots \\ A_{n-1} \end{bmatrix} = \begin{bmatrix} L_0 \\ L_1 \\ \dots \\ L_{\frac{n}{2}-1} \\ L_0 \\ L_1 \\ \dots \\ L_{\frac{n}{2}-1} \end{bmatrix} + \begin{bmatrix} w_n^0 R_0 \\ w_n^1 R_1 \\ \dots \\ w_n^{\frac{n}{2}-1} R_{\frac{n}{2}-1} \\ -w_n^0 R_0 \\ -w_n^1 R_1 \\ \dots \\ -w_n^{\frac{n}{2}-1} R_{\frac{n}{2}-1} \end{bmatrix} \tag{3}$$

Note 6. If $\vec{L}$ and $\vec{R}$ are known Eq. 3 derives $\vec{A}$ with $n/2$ additional COMPLEX multiplications and n COMPLEX additions/subtractions.

Note 7. A COMPLEX Addition (CA) requires 2 Real additions (RA).

Note 8. A COMPLEX Multiplication (CM) requires 2 RA's and 4 RM (Real Multiplications).

Running time - Operation Count

Let $M_c(n)$ and $A_c(n)$ be the number of complex multiplications and additions required to find the FFT of n numbers/points.

Number of Complex Multiplications in FFT. We have from Eq.3 that

$$M_c(n) = 2M_c(n/2) + n/2 \Rightarrow M_c(n) = n \lg n/2.$$

Number of Complex Addition/Subtractions in FFT. We have from Eq.3 that

$$A_c(n) = 2A_c(n/2) + n \Rightarrow A_c(n) = n \lg n.$$

Real Additions/Subtractions and Multiplications in FFT.

In order to derive the number of real additions and multiplications $M_r(n)$ and $A_r(n)$ respectively we take into consideration Notes 7 and 8 of the previous page.

$$M_r(n) = 4M_c(n) \Rightarrow M_r(n) = 2n \lg n.$$

$$A_r(n) = 2A_c(n) + 2M_c(n) \Rightarrow A_r(n) = 3n \lg n.$$

Real Operations in FFT.

Therefore the total number of operations (real additions, subtractions, and multiplications performed) is the total of

$$T(n) = M_r(n) + A_r(n) = 5n \lg n,$$

and Theorem Tukey-Cooley of page 6 has been proven. The pseudocode of the algorithm is shown on the next page.

(c) Copyright A. Gerbessiotis. All rights reserved.

Pseudocode

```
   FFT (a[0..n-1],n,w_n) // a is a  vector a=[a_0..a_n-1] of n points
1. if (n==1)
2.   return (a);
3. w   = 1;
4. l = [ a_0 , a_2 , ... , a_n-2 ]; // L is a      vector
5. r = [ a_1 , a_3 , ... , a_n-1 ]; // R is also a vector
6. L = FFT(l,n/2,w_n*w_n);  //w_n*w_n is w_(n/2)
7. R = FFT(r,n/2,w_n*w_n);
8. for(k=0;k<n/2;k++) {
9.  A[k]     = L[k] + w R[k];
10. A[k+n/2] = L[k] - w R[k];
11. w = w * w_n;
12 }
13.return(A);
```

(c) Copyright A. Gerbessiotis. All rights reserved.

With reference to page 5, we have that the inverse DFT can be computed through an inverse FFT, if after the computation the vector values are divided by n as well.

$$DFT_n^{-1}(\vec{A}) = \frac{1}{n}\left(F_n(w_n^{-1}) \times \vec{a}\right).$$

leads to

$$F_n^{-1}(w_n) = \frac{1}{n}F_n(w_n^{-1}).$$

The next page has the proof in detail why this is so!

Note 9. We can find $\vec{a}$ from $\vec{A}$ by an FFT computation where we use w_n^{-1} instead of w_n and the final result vector is divided by n. I.e. interpolation is also efficient.

Note 10. Another way to perform interpolation is by noting that in F_n, the i-th row and j-th column entry is w_n^{ij}, whereas in F_n^{-1}, the i-th row and j-th column entry is $w_n^{-ij} = w_n^{(n-i)j}$, i.e. it is the $(n-i)$-th row and j-th column entry of F_n.

(c) Copyright A. Gerbessiotis. All rights reserved.

Inverse FFT : A proof (can be skipped)

We now prove the claim that $F_n(w_n)F_n^{-1}(w_n) = I_n$, the identity $n \times n$ matrix.

$$F_n(w_n)F_n^{-1}(w_n) = F_n(w_n)\left(\frac{1}{n}F_n(w_n^{-1})\right) \quad = \quad \sum_{k=0}^{n-1} w_n^{ik} w_n^{-kj}/n = \sum_{k=0}^{n-1} w_n^{(i-j)k}/n$$

We distinguish two cases.

Case a. $i = j$ (case of a diagonal element of the product. Then

$$F_n(w_n)F_n^{-1}(w_n^{-1}) \quad = \quad \sum_{k=0}^{n-1} w_n^{(i-j)k}/n = \sum_{k=0}^{n-1} 1/n = n/n = 1.$$

This implies that the diagonal elements of the product are 1.

Case b. $i \neq j$. Then

$$\begin{aligned} F_n(w_n)F_n^{-1}(w_n^{-1}) &= \sum_{k=0}^{n-1} w_n^{(i-j)k}/n = \frac{1}{n}\sum_{k=0}^{n-1} \frac{w_n^{(i-j)n} - 1}{w_n^{(i-j)} - 1} = \frac{1}{n}\sum_{k=0}^{n-1} \frac{(w_n^n)^{i-j} - 1}{w_n^{(i-j)} - 1} \\ &= \frac{1}{n}\sum_{k=0}^{n-1} \frac{1-1}{w_n^{(i-j)} - 1} = 0. \end{aligned}$$

Therefore all non diagonal elements that have $i \neq j$ are zero. This proves the claim that the product of the two matrices is I_n.

Interpolation at roots of unity: Pseudocode

If we utilize Note 9, the interpolation code is

```
  Interpolate (A,w,n)
1. a= FFT(A,n, 1/w_n);
2. for(i=0;i<n;i++)
3.  a[i] = a[i]/n;   // Use Note 9.
```

If we utilize Note 10, however, step 1 is standard FFT.

```
  Interpolate (A,w,n)
1. aux= FFT(A,n,w_n);
2. a[0]=aux[0]/n;
3. for(i=1;i<n;i++)
4.  a[i] = aux[n-i]/n;  // Use Note 10
```

(c) Copyright A. Gerbessiotis. CS 667-101 : Spring 2012 . All rights reserved.

Polynomial Multiplication

We return to the polynomial multiplication problem by slightly generalizing it i.e. we allow the two polynomials to have different degrees.

Consider two polynomials $A(x) = a_{n-1}x^{n-1} + \ldots a_1 x + a_0$, and $B(x) = b_{m-1}x^{m-1} + \ldots b_1 x + b_0$,

Polynomial multiplication is the operation that multiplies $A(x)$ and $B(x)$ i.e. $C(x) = A(x)B(x)$ and is a polynomial of degree $n + m - 2$.

Polynomial Multiplication. Given two vectors $\vec{a} = (a_0, \ldots, a_{n-1})$, and $\vec{b} = (b_0, \ldots, b_{m-1})$, corresponding to the coefficients of two polynomials $A(x)$, $B(x)$, the coefficients of $C(x) = A(x)B(x)$ is vector $\vec{c} = (c_0, \ldots, c_{n+m-2})$ given by Eq.4 and can be computed in $15N \lg N + O(N)$ where $N = deg(A) + deg(B) + 1 = n + m - 2$. Vector $\vec{c}$ is also called the **convolution vector**.

$$c_j = \sum_{k=0}^{j} a_k b_{j-k} \tag{4}$$

Polynomial multiplication can be solved as follows.

- Pad the two polynomials, by adding zero coefficient higher order terms until both become of artificial degree $n + m - 2$, i.e. of as "big" degree as their product. The following discussion assumes that $N = n + m - 1$ is a power of 2. If it is not, continue padding until a power of two is reached (this requires doubling of the degree of the padded (not the original) polynomial in the worst-case.
- Evaluate $A(x)$ and $B(x)$ in the $N = n + m - 1$ roots of unity using FFT i.e. find $A_k = A(w_N^k)$ and $B_k = B(w_N^k)$. This requires $2 \cdot 5N \lg N$ real operations.
- Multiply N pairs of values A_k and B_k to find $C(x)$ at the N roots of unity. Given that $C(x) = A(x)B(x)$ we also have that $C(w_N^k) = A(w_N^k)B(w_N^k)$ i.e. $C_k = A_k B_k$. This requires N complex multiplications (2 additions, 4 multiplications each) for a total of $6N$ operations.
- Interpolate on the C_k to find degree $N-1$ polynomial $C(x)$. This requires an FFT operation and $5N \lg N + O(N)$ operations.

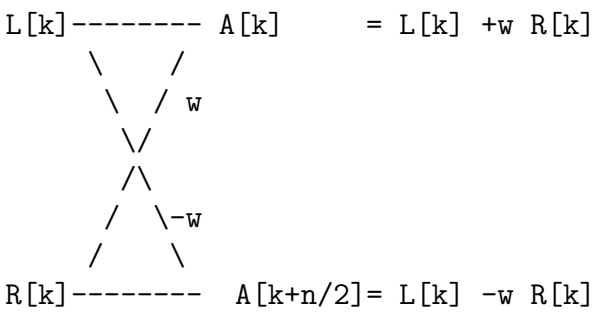
Polynomial Multiplication. Is an $O(n \lg n)$ operation with the hidden constant around 15.

Convolution. Convolution is asymptotically as difficult as polynomial evaluation.

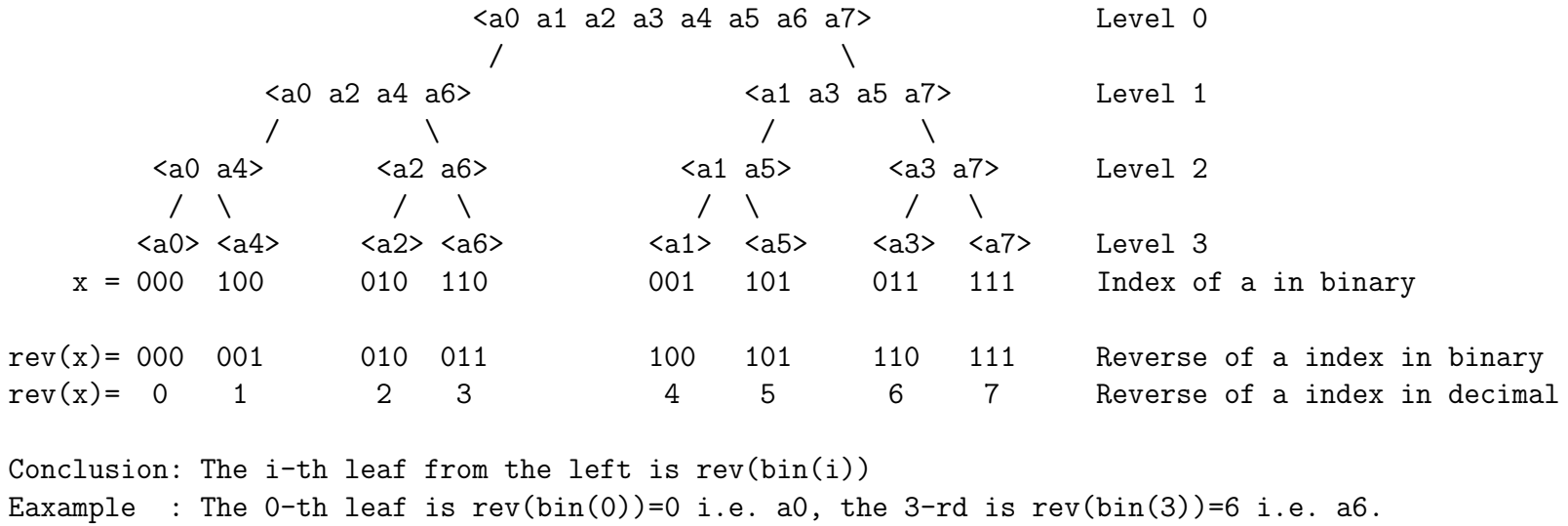
(c) Copyright A. Gerbessiotis. All rights reserved.

Efficiency Issues

In lines 10-11 of the FFT pseudocode the operations performed can be described by an operation that is known as butterfly. In particular consider the schematic below. At A_k the input lines from L_k and R_k meet; the line from R_k feeds R_k multiplies by the value shown on that line w. Similarly the line from L_k to $A_{k+n/2}$ has $-w$ i.e. L_k should be multiplied by $-w$



We then show how the FFT recursive calls unfold, i.e. how the input is distributed in the various recursive calls.



Conclusion: The i-th leaf from the left is rev(bin(i))
Eaxample : The 0-th leaf is rev(bin(0))=0 i.e. a0, the 3-rd is rev(bin(3))=6 i.e. a6.

Efficiency Issues (continued)

We also observe the following about the bit representation of the indices $(0 \ldots 7)$. For x a binary string of zeroes and ones, let $rev(x)$ be the reverse string of x where x is written in binary, i.e. $rev(100) = 001$.

```
Bit positions are numbered right (least significant) to left (most signif.)
In level 1:  0,2,4,6, are  000, 010, 100, 110 i.e. all have 0 in bit 1
        and  1,3,5,7  are  001, 011, 101, 111 i.e. all have 1 in but 1

In level 2:  0 4 are 000 100  1,5 are 001 101 all have 0 in bit 2
             2 6 are 010 110  3,7 are 011 111 all have 1 in bit 2

In level 3:  0 2 1 3 are 000 010 001 011 all have 0 in bit 3
             4 6 5 7 are 100 110 101 111 all have 1 in bit 3.

Conclusion 1: A node at level i if it is left  child it  has 0 in bit i,
     whereas                  if it is right child it  has 1 in bit i.

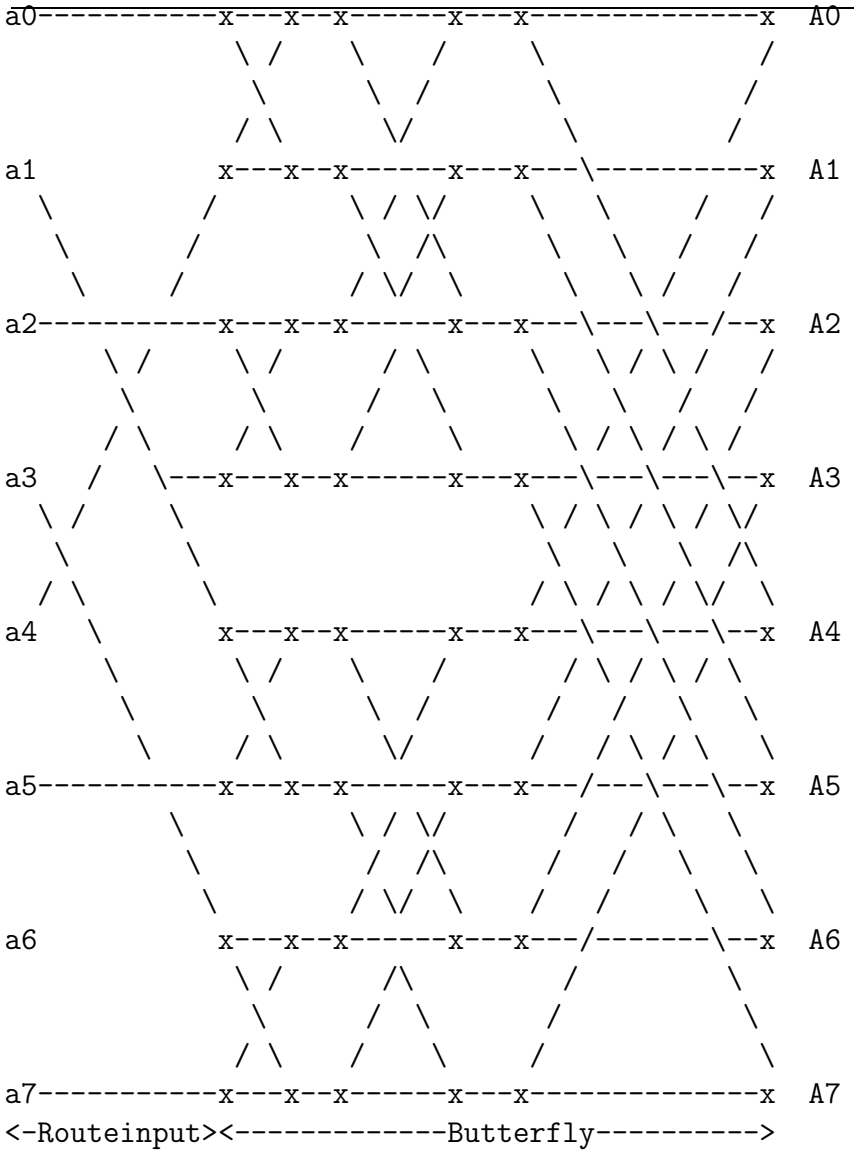
Conclusion 2: The j-th leaf from the left is rev(j).
```

The operations performed during an FFT computation, are sometimes referred to as the butterfly network.

The FFT network was introduced for the purpose of performing FFTs efficiently and in parallel. Each parallel step of an n point FFT is carried out on one level of a $\lg n$-dimensional butterfly. In level 3, the w_2 are used, in level 2, w_4 and in level 1, w_8.

(c) Copyright A. Gerbessiotis. All rights reserved.

FFT on the butterfly



(c) Copyright A. Gerbessiotis. All rights reserved.

Polynomial Division

We have shown how polynomial multiplication can be performed efficiently faster that $O(n^2)$, where n is the degree of the polynomials to be multiplied, by using FFT. In this section we will show how to perform polynomial division.

Let us consider two polynomials $f(x)$ and $g(x)$, where $f(x)$ is of degree n and $deg(g) = m \leq n = deg(f)$ respectively. Then we know that there exist two polynomial $q(x)$, and $r(x)$, the quotient and remainder respectively such that

$$f(x) = g(x)q(x) + r(x) \tag{5}$$

such that $deg(r) < deg(g) = m$. By Eq. 5 it is also imperative that since $deg(f) = deg(q) + deg(g)$, then $deg(q) \leq deg(f)$. The quotient of the division of $f(x)$ and $g(x)$ is represented by $f(x)/g(x)$. This can be rewritten as the product $f(x) \cdot 1/g(x)$. Therefore polynomial division is equivalent to determining the inverse $1/g(x)$ of a polynomial, and running time for division is no more that running time for polynomial multiplication plus the running time for polynomial inverse finding.

The following theorem will be proved.

Theorem PolyDivision. Polynomials q and r can be computed in $O(n \lg n)$ operations given polynomials f and g.

Example 1. Let 13 and 3 be integers. Then, $13 = 3 \cdot 4 + 1$. We also have that $13 \bmod 3 = 13\%3 = 1$, and $13/3 = 13 div 3 = 4$.

Example 2. Let $g(x) = 1 - x$. Then $1/g(x) = 1 + x + x^2 + \ldots$.
Proof We multiply $g(x)$ and $1/g(x)$.

$$\begin{aligned} g(x)(1/g(x)) &= (1-x)(1+x+x^2+\ldots) \\ &= (1+x+x^2+\ldots) - x(1+x+x^2+\ldots) \\ &= 1+x+x^2+\ldots-x-x^2-\ldots \\ &= 1 \end{aligned} \tag{6}$$

Background (H. T. Kung)

Let $f(x), g(x)$ be two polynomials of degrees n, m respectively where $n \geq m$. We may assume $g_m = 1$ since otherwise we can scale the two polynomials by dividing all coefficients by g_m in $O(n+m)$ time. Then $f(x) = g(x)q(x) + r(x)$, gives the $q(x)$, the quotient and $r(x)$, the remainder, of the division of $f(x)$ by $g(x)$. Polynomial $q(x)$ has degree $n-m$ and $r(x)$ has degree less than m, i.e. the degree bound of $r(x)$ is m. Let the coefficients of the polynomials be f_i, g_i, q_i, r_i for the appropriate range of i.

Then, in the derivations below we first substitute $1/z$ for x, then we multiply both sides by z^n and reorder/rewrite the terms of the sum last to first rather than first to last.

$$\begin{aligned}
f(x) &= g(x)q(x) + r(x) \\
\sum_{i=0}^{n} f_i x^i &= \sum_{i=0}^{m} g_i x^i \cdot \sum_{i=0}^{n-m} q_i x^i + \sum_{i=0}^{m-1} r_i x^i \\
\sum_{i=0}^{n} f_i z^{-i} &= \sum_{i=0}^{m} g_i z^{-i} \cdot \sum_{i=0}^{n-m} q_i z^{-i} + \sum_{i=0}^{m-1} r_i z^{-i} \\
\sum_{i=0}^{n} f_{n-i} z^i &= \sum_{i=0}^{m} g_{m-i} z^i \cdot \sum_{i=0}^{n-m} q_{n-m-i} z^i + \sum_{i=0}^{m-1} r_{m-1-i} z^{n-m+1+i} \qquad (7)
\end{aligned}$$

In Eq. 7 we then take both sides $\bmod z^{n-m+1}$, which in effect truncates all terms higher that z^{n-m}. This has the effect of removing terms from f, possibly g, it does not affect q, and completely eliminates the r since the lowest term there is $z^{n-m+1+0}$. Therefore only the q terms are unknown and they can be found from the resulting formula. Then, $r(x)$ can be established from $r(x) = f(x) - q(x)g(x)$.

$$\begin{aligned}
\sum_{i=0}^{n} f_{n-i} z^i (\bmod z^{n-m+1}) &= \sum_{i=0}^{m} g_{m-i} z^i \cdot \sum_{i=0}^{n-m} q_{n-m-i} z^i + \sum_{i=0}^{m-1} r_{m-1-i} z^{n-m+1+i} (\bmod z^{n-m+1}) \\
\sum_{i=0}^{n-m} f_{n-i} z^i &= \sum_{i=0}^{n-m} g_{m-i} z^i \cdot \sum_{i=0}^{n-m} q_{n-m-i} z^i (\bmod z^{n-m+1}) \qquad (8)
\end{aligned}$$

(c) Copyright A. Gerbessiotis. All rights reserved.

Background (H. T. Kung) continued

In Eq. 8, we can find the inverse of $g(x)$ depicted as $G(x) = 1/g(x)$, easily provided that $g_m = 1$. This is based on the idea that since $1/1 - x = 1 + x + x^2 + \ldots + x^i + \ldots$, then $k(x) = 1 + k_1 x + \ldots = 1 - h(x)$, where $h(x) = -k_1 x - \ldots$, and thus $K(x) = 1/k(x) = 1/(1 - h(x)) = 1 + h(x) + h^2(x) + \ldots$. This serves as proof for the time being. More details are available in the next slide.

Therefore there exists polynomial $G(x)$ such that $\left(\sum_{i=0}^{n-m} g_{m-i} z^i\right) G(x) = 1$ and thus

$$\begin{aligned}
\sum_{i=0}^{n-m} f_{n-i} z^i &= \sum_{i=0}^{n-m} g_{m-i} z^i \cdot \sum_{i=0}^{n-m} q_{n-m-i} z^i (\mathrm{mod} z^{n-m+1}) \\
\left(\sum_{i=0}^{n-m} f_{n-i} z^i\right) G(x) &= \sum_{i=0}^{n-m} q_{n-m-i} z^i \\
\sum_{i=0}^{n-m} q_{n-m-i} z^i &= \left(\sum_{i=0}^{n-m} f_{n-i} z^i\right) \left(G(x) \bmod z^{n-m+1}\right) (\mathrm{mod} z^{n-m+1}) \qquad (9)
\end{aligned}$$

Therefore we can find $q(x)$ as long as we know how to find a finite number of terms of $G(x)$. The number of term of $G(x)$ needed is $n - m + 1$ i.e. those of coefficients up to and including z^{n-m}. We find those terms of $G(x)$ through Newton-Raphson (last two/three pages). We prove there thand finding $q(x)$ requires $O(n \lg n)$ time.

Having found $q(x)$ we can find $r(x)$ from $f(x) - q(x)g(x) = r(x)$ in also $O(n \lg n)$ time.

Theorem. Polynomial division requires $O(n \lg n)$ time through FFT.

(c) Copyright A. Gerbessiotis. All rights reserved.

Let a and b be two polynomials $a(x) = a_0 + a_1x + \ldots$ and $b(x) = b_0 + b_1x + \ldots$.
Then $a(x) + b(x) = (a_0 + b_0) + (a_1 + b_1)x + \ldots$.
Also, $a(x)b(x) = (a_0b_0) + (a_0b_1 + a_1b_0)x + \ldots + (a_0b_n + \ldots + a_nb_0)x^n + \ldots$.
Suppose that polynomial g is such that $g(x) = 1 + g_1x + \ldots$. Then g has an inverse $1/g$. This is because $1/g(x) = 1/(1 - (-g_1x - \ldots) = 1/(1 - h(x))$, where $h(x) = -g_1x - \ldots$, and thus $1/g(x) = 1 + h(x) + h^2(x) + \ldots$.
We have just shown the following.

Theorem Inverse Any polynomial g of the form $g(x) = 1 + g_1x + \ldots$ has an inverse $g^{-1} = 1/g$.

Def. 1. For polynomials f and h we write $f \equiv h \pmod{x^n}$ if $x^n/f - h$ i.e. $\exists g(x)$ such that $f(x) - h(x) = x^n g(x)$.
Lemma 1. If $f \equiv h \bmod x^n$, then $f_0 = h_0, f_1 = h_1, \ldots, f_{n-1} = h_{n-1}$ for the coefficients of the two polynomials.

Approximating $f(x)/g(x)$. Since $f(x) = g(x)q(x) + r(x)$ and $deg(g) \leq deg(f) = n$ and also $deg(q) \leq deg(f)$ it suffices if we can compute NOT ALL of $q(x)$ but only the first $n + 1$ coefficients of all terms of degree up to n, i.e. it suffices to find $q(x) \bmod x^{n+1}$.

Example 3. Let $f(x) = x^3 - 1$ with $deg(f) = 3$ and $g(x) = 1 - x$. Then $deg(q) \leq 3$. Therefore it suffices to find $q(x) \bmod x^4$.
We know from before that $1/g(x) = 1 + x + x^2 + x^3 + \ldots$. Then the first $3 + 1$ terms of $1/g(x)$ are $(1/g(x)) \bmod x^{3+1} = 1 + x + x^2 + x^3$.
If we multiply $f(x)$ and $1/g(x) \bmod x^4$ we obtain

$$(x^3 - 1)(1 + x + x^2 + x^3) = -1 - x - x^2 - x^3 + x^3 + x^4 + x^5 + x^6 == -1 - x - x^2 + x^4 + x^5 + x^6$$

Since $deg(q) \leq 3$ by taking modx^4 of the above result we obtain $q(x) = ((f(x)(1/g(x) \bmod x^4)) \bmod x^4) = -1 - x - x^2$, as required. It is then easy to verify that $f(x) = x^3 - 1 = (-1 - x - x^2)g(x) = (-1 - x - x^2)(1 - x)$.

(c) Copyright A. Gerbessiotis. All rights reserved.

Consider $f(a) = b$ for some polynomial f.

- If f, a are given and b is needed, the problem is called polynomial evaluation.
- If a, b are given and f is needed, it is polynomial interpolation.
- If f, b are given and a is needed, the problem is called root finding.

Suppose we have a polynomial f and we want to find the roots of $f(x) = 0$. One way to find a root of f is iterative. First establish two points a, b such that $f(a)f(b) < 0$. This would mean that a solution to $f(x) = 0$ is in the interval $[a, b]$. Then a binary search is performed by inspecting the intervals $[a, c]$ and $[c, b]$ with $c = (a+b)/2$ to further refine the interval containing a solution. The two new intervals are half the size of the original. After m iterations the interval containing a solution is $1/2^m$ the size of the original $[a, b]$ interval. For a predetermined precision ϵ the method stops when $2^{-m}(b - a) < \epsilon$.

A better and faster approximation method is Newton-Raphson approximation. Let r be a root and let the current approximation to the solution be x_0 such that $r = x_0 + \delta$, i.e. we are δ away from the root. Then $f(r) = f(x_0 + \delta) = 0$. By taking a Taylor expansion of $f(x_0 + \delta)$ we have that

$$0 = f(x_0 + \delta) = f(x_0) + \delta f'(x_0) + \delta^2 \Delta$$

where Δ is a tiny quantity. From this approximation we conclude that

$$0 \approx f(x_0) + \delta f'(x_0)$$

i.e. $\delta = -f(x_0)/f'(x_0)$.

In Newton-Raphson we start with an arbitrary approximation of the root r, say x_0. Then, the next approximation becomes $x_1 = x_0 - f(x_0)/f'(x_0)$, and then a refinement of $x_2 = x_1 - f(x_1)/f'(x_1)$ is used , and this is repeated until the obtained approximation x_i is satisfactorily close to r.

(c) Copyright A. Gerbessiotis. All rights reserved.

Let g be a real number. We would use N-R to find $y = 1/g$. Let us consider $f(y) = g - 1/y$. The solution of $f(y) = 0$ for y is $g - 1/y = 0$ therefore $y = 1/g$ is the inverse of g.

Observation. The inverse of g is the solution of $f(y) = 0$.

Using the Newton-Raphson method, we get that $f'(y) = 1/y^2$. If the first approximation to $1/g$ is y_0 (we will discuss how to choose such a y_0 later). Then $y_1 = y_0 - f(y_0)/f'(y_0) = y_0 - (g - 1/y_0)/(1/y_0^2)$ and then we obtain that $y_1 = 2y_0 - gy_0^2$. Therefore,

Lemma APPRX. If y_0 is an initial approximation to $1/g$, then the sequence of Newton-Raphson approximations is given by

$$y_i \;=\; 2y_{i-1} - gy_{i-1}^2.$$

Lemma Error. If y_0 is an initial approximation to $1/g$, then in the i-th Newton-Raphson iteration the error is if $g \leq 1$.

$$1/g - y_i \;\leq\; g(1/g - y_0)^{2^i}. \tag{10}$$

Proof of Lemma Error. Consider first the error of approximating $1/g$ by y_1. We obtain the following error bound.

$$\begin{aligned} 1/g - y_1 &= 1/g - 2y_0 + gy_0^2 \\ &= g(1/g^2 - 2y_0(1/g) + y_0^2) \\ &= g(1/g - y_0)^2 \end{aligned} \tag{11}$$

Induction completes the rest of the proof. □

If $g \leq 1$ and $\mid 1/g - y_0 \mid \leq 1/2$, then after i iterations the error of approximating $1/g$ by y_i is at most $1/2^{2^i}$. For $i = \lg n$, the error is less than $1/2^n$.

(c) Copyright A. Gerbessiotis. All rights reserved.

We examine the quality of this Newton-Raphson approximation.

Towards this, we have a short excursion on error bounds.

We know that $1/3 = 0.33333\ldots$. If we approximate $1/3$ by 0.3 then the error is $0.0333\ldots$ or less than $4\ 10^{-2}$, if we approximate $1/3$ by 0.33 then the error is $0.00333\ldots$, or less than $4\ 10^{-3}$ and so on.

For binary numbers, the representation of integers as binary numbers is quite established. If fractional numbers are represented as binary, 0.11 represents in binary $2^{-1} + 2^{-2} = 3/4$ and $0.111 = 3/4 + 2^{-3} = 3/4 + 1/8 = 7/8$.

Therefore approximating 0.111 by 0.11 introduces an error of 2^{-3} (two to the power of the index of the eliminated bit, index 1 starting immediately to the right of the period .). As a conclusion, if the error is less than 2^{-m}, this means that the approximation is accurate to $m-1$ significant digits.

An error after $\lg n$ iterations of at most $1/2^n$ means that the approximation is correct to $n-1$ digits.

Let a be a number represented in binary as $a = 2^m \times 0.\epsilon_1\epsilon_2\ldots = 2^m \cdot 0.b$, with $.b = .\epsilon_1\epsilon_2\ldots$, and $\epsilon_1 = 1$. Note that if $\epsilon_1 \neq 1$, we can adjust (decrease) m to achieve this.

For example consider $a = 0.25$ which is $a = 0.01$ in binary. Since $\epsilon_1 \neq 1$, shifting a left one position we obtain $0.1 \times 2^{-1} = 2^{-1}.1$ which is still $(1/2)(2^{-1}) = 1/4 = 0.25$.

Because $\epsilon_1 = 1$, we have that $1/2 \leq b \leq 1$. Then $a^{-1} = 2^{-m}b^{-1}$, where $1 \leq b^{-1} \leq 2$.

Now we are ready to apply N-R to polynomial inverse and division.

A binary number B written as $b_{n-1} \ldots b_1 b_0$ denotes $b_{n-1}2^{n-1} + \ldots + b_1 2^1 + b_0$, in other words B is the evaluation of the polynomial $b(x) = b_{n-1}x^{n-1} + \ldots b_1 x^1 + b_0$ at point $x = 2$, i.e. $B = b(2)$. This establishes a relationship between polynomials and number representations, and can also be extended to any radix/base other than binary.

Lemma Poly-N-R. A Newton-Raphson approximation of the inverse of a polynomial g, where $g(x) = 1 + g_1 x + \ldots$, yields an approximation y_k in the k-th iteration such that $y_k = (1/g(x)) \bmod x^{2^k}$, with $y_0 = 1$. y_k is thus accurate for the coefficients of all powers up to (including) x^{2^k-1}.

Proof Poly-N-R. (by induction on k).

For the base case $k = 0$ we have that $x = x^{2^0}$ divides $y_0 - 1/g = 1 - 1 - g_1 x - \ldots = xh(x)$, and therefore the base case is true.

We approximate $y_{k+1} = 2y_k - gy_k^2$ and assume by the inductive hypothesis that $y_k = (1/g(x)) \bmod x^{2^k}$. Then similarly to the error bound derivation of the ordinary number N-R approximation we derive that

$$1/g - y_{k+1} = g(1/g - y_k)^2.$$

Since $y_k = (1/g(x)) \bmod x^{2^k}$ we have that $x^{2^k}/(y_k - 1/g(x))$ i.e. $(y_k - 1/g(x)) = x^{2^k}h(x)$ for some $h(x)$. Then,

$$1/g - y_{k+1} = g(1/g - y_k)^2 = g(x^{2^k}h(x))^2 = gx^{2^{k+1}}h^2(x)$$

which implies that $x^{2^{k+1}}$ divides $1/g - y_{k+1}$ which is equivalent to $1/g = y_{k+1} \bmod x^{2^{k+1}}$, as required. This concludes the induction. □

```
  Algorithm Inverse_Simple (g(x),n) //computes 1/g(x) mod x**(n+1)
1. y =1; // y is a polynomial; initialized to constant polynomial here.
2. for(i=1;i<ceil(lg(n+1)); i++) {
3.   y =  2 * y - g(x) * y * y;
   }
4. return(y mod x**(n+1)) // return coefficients of y(x) up to x**n term
```

A faster approximation

A better and faster approximation can be obtained.

In such an approximation $z_0 = 1$ as before, but

$$z_{i+1} \quad = \quad (2z_i - (g \bmod x^{2^{i+1}})z_i^2) \bmod x^{2^{i+1}}. \tag{12}$$

The difference between this and the prior approximation is that we do not fully utilize g (i.e. all its terms but only those needed). This is analogous to Example 3, where $f(x)/g(x)$ is computed; since $f(x)/g(x)$ is of degree at most 3 the terms $x^4, x^5, \ldots$ are redundant anyway and their calculation not necessary.

Analysis of Inversion. Equation 12 requires $O(2^i \lg 2^i) = O(2^i i)$ operations using FFT for polynomial mutliplication Then

$$\sum_{i=1}^{\lg n} O(2^i i) = O(\sum_{i=1}^{\lg n} 2^i i) = O(\lg n \sum_{i=1}^{\lg n} 2^i) = O(\lg n 2^{\lg n}) = O(n \lg n),$$

and inversion is no more complex than multiplication.

In order to compute the quotient $f(x)/g(x)$, given that we have computed $1/g(x)$ one extra polynomial multiplication is required.

Theorem Division. The first $n+1$ terms of degree up to n of the polynomial $f(x)/g(x)$ can be computed on $O(n \lg n)$ time.

(c) Copyright A. Gerbessiotis. All rights reserved.