The Fast Fourier Transform (FFT)

Disclaimer: These notes DO NOT substitute the textbook for this class. The notes should be used IN CONJUNCTION with the textbook and the material presented in class. If a statement in these notes seems to be incorrect, report it to the instructor so that it be fixed immediately. These notes are only distributed to the students taking this class with A. Gerbessiotis in Fall 2004; distribution outside this group of students is NOT allowed.

Degrees and Degree bounds

A polynomial $A(x) = a_0 + \ldots + a_{n-1}x^{n-1}$ is of degree $n-1$ if $a_{n-1} \neq 0$. Then, any $k > n-1$ is the degree bound of $A(x)$. Thus a polynomial of degree bound n may have degree 0, 1, ... up to (inclusive) $n-1$. If two polynomial $A(x)$ and $B(x)$ of degree bounds a (i.e. of actual degree at most $a-1$) and b (i.e. of actual degree at most $b-1$) respectively are multiplied, the product $C(x) = A(x)B(x)$ is of actual degree $a+b-2$ i.e. its degree bound is $a+b-1$ which is at most $a+b$.

Although we will avoid the use of the term degree bound in the remainder, in any subsequent discussion a reference to a polynomial of degree n may also imply that its degree is less than n.

Coefficient Representation of a polynomial.

Given a polynomial $A(x)$ of degree bound n we represent it by the ordered tuple of its coefficients $(a_0, \ldots, a_{n-1})$.

Polynomial Evaluation

The operation that finds the value of a polynomial like $A(x)$ at a point $x = c$ is called polynomial evaluation and the simplest way to evaluate $A(c)$ is by using Horner's rule that requires n additions and n multiplications.

Therefore the problem that given $A(x)$ in the form $(a_0, \ldots, a_{n-1})$ and a number c and finding $A(c)$, is called polynomial evaluation.

Point-value Representation of a polynomial.

A polynomial $A(x)$ of degree $n-1$ can also be uniquely represented by n different point-value pairs, i.e. the tuple

$$((x_0, A(x_0)), \ldots, (x_{n-1}A(x_{n-1})) \text{ where } x_i \neq x_j \text{ for all } i, j$$

Given these n pairs the coefficients of $A(x)$ can easily be found by an operation that is known as interpolation using Lagrange's formula in $O(n^2)$ time.

Lagrange's Formula

Lagrange's formula Consider n distinct point-value pairs $((x_1, b_1), \ldots, (x_n, b_n))$. We want to find the $n-1$ degree polynomial $A(x) = a_{n-1}x^{n-1} + \ldots + a_1 x + a_0$ such that $f(x_i) = b_i$, $1 \leq i \leq n$.

$$g_i(x) = \frac{(x - x_1)(x - x_2)\ldots(x - x_{i-1})(x - x_{i+1})\ldots(x - x_n)}{(x_i - x_1)(x_i - x_2)\ldots(x_i - x_{i-1})(x_i - x_{i+1})\ldots(x_i - x_n)}$$

It is easy to verify that $g_i(x_i) = 1$ as the numerator and denominator cancel out, and $g_i(x_j) = 0$ for $j \neq i$ as $(x_j - x_j)$ appears in the numerator and the corresponding term in the denominator is $x_i - x_j$.

If we consider the polynomial derived from all $g_i(x)$ such that

$$g(x) = b_1 g_1(x) + \ldots + b_n g_n(x)$$

we observe that $g(x_i) = b_i$ for all $i = 1, \ldots, n$.

Therefore

$$A(x) = g(x).$$

Point-value Representation of the product of two polynomials.

Given $A(x)$, $B(x)$ the product $C(x) = A(x)B(x)$ is of degree $2n-2$. Therefore its point-value representation requires $2n-1$ pairs to be known in $2n-1$ distinct values. The point-value representation of $A(x)$ and $B(x)$ are available, however, in only n points as they are both of degree $n-1$. Therefore we need to represent $A(x)$ and $B(x)$ in an extended form of $2n-1$ point-value pairs for this representation to be useful in establishing $C(x)$'s point-value pair representation.

Then let $((x_0, a_0), (x_1, a_1), \ldots, (x_{2n-1}, a_{2n-1})$ be the point-value representation of $A(x)$ in extended form. Let similarly $((x_0, b_0), (x_1, b_1), \ldots, (x_{2n-1}, b_{2n-1})$ be the point-value representation of $B(x)$ in extended form. Then since $C(x) = A(x)B(x)$ we know that $C(x_i) = A(x_i)B(x_i)$.

Therefore the point-value representation of $C(x)$ is $((x_0, a_0b_0), (x_1, a_1b_1), \ldots, (x_{2n-1}, a_{2n-1}b_{2n-1})$, and can be obtained from those of $A(x)$, $B(x)$ by performing $2n-1$ multiplications only i.e. it's very efficient compared to performing a polynomial multiplication of $A(x)$ and $B(x)$ to obtain $C(x)$ and then evaluating $C(x)$ at $2n-1$ or $2n$ points for a total cost of $\Theta(n^2)$.

Given this representation, $C(x)$ can be derived very easily by an additional interpolation operation.

Therefore there are two ways to find $C(x) = A(x)B(x)$ given $A(x), B(x)$.

```
   NaiveMultiply(A(x),B(x))
1. C(x)=A(x)* B(x);
2. return(C(x));
```

```
      IntelligentMultiply(A(x),B(x))
*1.  Evaluate A(x) at 2n points x_i.
*2.  Evaluate B(x) at 2n points x_i.
 3.  Pairwise multiply A(x_i)B(x_i)
 4.  Since C(x_i) = A(x_i) B(x_i)
     Interpolate to find C(x)
 5.  return(C(x));
```

```
* : We only need evaluate the polynomials at 2n-1 points. The redundancy
    (one extra evaluation is to simplify exposition).
```

In the naive method step 1 requires $O(n^2)$ time.

In the intelligent method step 3 requires $O(n)$ time. If steps 1,2, and 4 are performed naively they would all require $O(n^2)$ time and this method becomes dumber than the naive method. If we choose, however, the x_i intelligently then, steps 1 and 2 can be performed in $O(n \lg n)$ time using a divide-and-conquer method, and step 4 becomes an operation similar to those of steps 1 and 2 and can also be performed in time $O(n \lg n)$. Thereforee total running time for the Intelligent method becomes $O(n \lg n)$ compared to the $O(n^2)$ of the naive approach.

As a final note, whether $A(x)$ is evaluated at $2n$ points or $2n - 1$ is not that important. It helps the fact that $2n$ is an even number, and so some extra redundancy makes calculations easier.

Convolution

Let $A(x)$ be a polynomial in indeterminate x of degree $n-1$, i.e. $A(x) = \sum_j a_j x^j$. Let also $B(x)$ be another polynomial of the same degree, i.e. $B(x) = \sum_j b_j x^j$. Then the product $C(x) = A(x)B(x)$ of A and B is a polynomial $C(x)$ of degree the sum of the degrees of $A(x)$ and $B(x)$ i.e. $C(x) = \sum_{j=0}^{2n-2} c_j x^j$. Elementary calculations show that

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

i.e. $a_m b_n$ is a term of c_{m+n}. The sequence of the c_j is also called the convolution of a_i's and b_i's. Each one of the c_j requires time $O(n)$ for computation.

Theorem Convolution. Convolution can be done in $5 \lg n$ operations per c_j.

Let w_n be the n-th primitive (or principal) root of unity, i.e. w_n is the solution of $x^n = 1$ such that $w_n = cos(2\pi/n) + i\ sin(2\pi/n) = e^{2\pi i/n}$,

All n distinct roots of unity, i.e. the solutions of $x^n = 1$ are denoted w_n^j in terms of the n-th primitive root of unity.

All roots $w_n^0 = 1, w_n^1 = w_n, \ldots w_n^{n-1}$ 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 $n/2$ primitive roots of unity, i.e. w_n^{2j} is a $n/2$-th root of unity, and the n squares correspond to $n/2$ different values (two n-th primitive roots of unity have squares that are mapped to a single $n/2$-th primitive root of unity).

Proof Sketch (P4) $(w_n^k)^2 = w_{n/2}^k$ by P2. 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 (P5).

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

Let $A(x) = a_0 + a_1x + a_2x^2 + \ldots + a_{n-1}x^{n-1}$, be an $n-1$-st degree polynomial. Our objective is to evaluate this polynomial in the n roots of unity fast (faster than $O(n^2)$). We first observe that

$$\begin{aligned}
A_0 &= a_0 + a_1w_n^{0\cdot 1} + a_2w_n^{0\cdot 2} + \ldots + a_{n-1}w_n^{0\cdot(n-1)} \\
A_1 &= a_0 + a_1w_n^{1\cdot 1} + a_2w_n^{1\cdot 2} + \ldots + a_{n-1}w_n^{1\cdot(n-1)} \\
&\ldots \\
A_k &= a_0 + a_1w_n^{k\cdot 1} + a_2w_n^{k\cdot 2} + \ldots + a_{n-1}w_n^{k\cdot(n-1)} \\
&\ldots \\
A_{n-1} &= a_0 + a_1w_n^{(n-1)1} + a_2w_n^{(n-1)2} + \ldots + a_{n-1}w_n^{(n-1)(n-1)}
\end{aligned}$$

Then $A_k = A(w_n^k)$, $0 \leq k \leq n-1$. The vector of A_k can be written in matrix form as follows.

$$\begin{bmatrix} A_0 \\ A_1 \\ A_2 \\ \ldots \\ A_{n-1} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & \ldots & 1 \\ 1 & w_n & w_n^2 & \ldots & w_n^{n-1} \\ 1 & w_n^2 & w_n^{2\cdot 2} & \ldots & w_n^{2(n-1)} \\ \ldots & \ldots & \ldots & \ldots & \ldots \\ 1 & w_n^{n-1} & w_n^{(n-1)\cdot 2} & \ldots & w_n^{(n-1)(n-1)} \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ \ldots \\ a_{n-1} \end{bmatrix}.$$

The vector A_k is called the **Discrete Fourier Transform** of the coefficient vector a_k. We sometimes write $\vec{A} = DFT_n(\vec{a})$. This implies that the relationship between vectors A and 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 and $w_n^{(i-1)(j-1)}$ if i,j start from 1.

Therefore we can also write this in a simpler form as follows. Let $\vec{A} = F_n\vec{a}$, for the two vectors $\vec{a}, \vec{A}$.

DFT can be computed efficiently through a method that is called FFT (Fast Fourier Transform).

The following theorem tells us how fast we can perform this polynomial evaluation problem on the n roots of unity. The method is a divide and conquer strategy.

Theorem Tukey-Cooley. FFT, i.e. the computation of vector $\vec{A}$ such that $\vec{A} = F_n\vec{a}$ can be computed by $5n \lg n$ real operations provided that $n = 2^m$.

Note: If n is not a power of n we can always make it so 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 at most 2.

Proof of Tukey-Cooley theorem.

Let

$$\begin{aligned} A(x) &= a_0 + a_1x + \ldots + a_{n-1}x^{n-1} = (a_0 + a_2x^2 + \ldots + a_{n-2}x^{n-2}) + x(a_1 + x_3x^2 + \ldots + a_{n-1}x^{n-2}) \\ &= L(x^2) + xR(x^2) \end{aligned}$$

where $L(x) = a_0 + a_2x + \ldots + a_{n-2}x^{n/2-1}$, is the $n/2 - 1$-degree polynomial of the even coefficients and $R(x)$ $R(x) = a_1 + a_3x + \ldots + a_{n-1}x^{n/2-1}$ is the $n/2 - 1$-degree polynonial of the odd coefficients of $A(x)$. Note that $A(x)$ has n coefficients and $R(x), L(x)$ have $n/2$ each.

Then, since $A_k = A(w_n^k)$ we have that for $0 \leq k \leq n/2 - 1$,

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

Similarly, $A_{k+n/2}$, $0 \leq k \leq n/2 - 1$, is given by the following expression. We also note that $w_n^{n/2} = -1$, and $w_n^n = 1$.

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

(3)

Observation. The set of n A_k values depend on $n/2$ $L(.)$ values and $n/2$ $R(.)$ values. In fact A_k and $A_{k+n/2}$ that are $n/2$ part are expressed in terms of the same $L(.)$ and $R(.)$ values. In addition the computation of L_k, R_k requires a DFT on $n/2$ values, the $w_{n/2}^k$ $n/2$-roots of unity.

In vector form we have therefore established the following.

By assumption we have that

$$\begin{bmatrix} A_0 \\ \dots \\ A_{n-1} \end{bmatrix} = F_n(w_n) \begin{bmatrix} a_0 \\ \dots \\ a_{n-1} \end{bmatrix}.$$

By definition following the derivations of Eq. 1 and Eq. 2 we obtain that

$$\begin{bmatrix} L_0 \\ L_1 \\ \dots \\ L_{n/2-1} \end{bmatrix} = F_{n/2}(w_{n/2}) \begin{bmatrix} a_0 \\ a_2 \\ \dots \\ a_{n-2} \end{bmatrix}.$$

and

$$\begin{bmatrix} R_0 \\ R_1 \\ \dots \\ R_{n/2-1} \end{bmatrix} = F_{n/2}(w_{n/2}) \begin{bmatrix} a_1 \\ a_3 \\ \dots \\ a_{n-1} \end{bmatrix}.$$

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

$$\begin{bmatrix} A_0 \\ A_1 \\ \cdots \\ A_{n/2-1} \\ A_{n/2} \\ A_{n/2+1} \\ \cdots \\ A_{n-1} \end{bmatrix} = \begin{bmatrix} L_0 \\ L_1 \\ \cdots \\ L_{n/2-1} \\ L_0 \\ L_1 \\ \cdots \\ L_{n/2-1} \end{bmatrix} + \begin{bmatrix} w_n^0 R_0 \\ w_n^1 R_1 \\ \cdots \\ w_n^{n/2-1} R_{n/2-1} \\ -w_n^0 R_0 \\ -w_n^1 R_1 \\ \cdots \\ -w_n^{n/2-1} R_{n/2-1} \end{bmatrix} \tag{4}$$

We note that the evaluation of $A(t)$ at the n roots of unity is equivalent to the evaluation of the vector A_k (i.e. $\vec{A}$).

We conclude from Eq. 1, 2 that the evaluation of $\vec{A}$ is equivalent to the evaluation first of the two vectors $\vec{L}$ and $\vec{R}$, and then, their combination through Eq. 4.

Each one of $\vec{L}, \vec{R}$ is of length $n/2$, half of that of $\vec{A}$. The evaluation of $\vec{L}$ is equivalent to the evaluation of $L(x)$ in the $n/2$ $n/2$-th roots of unity (evident from Eq. 1, 2).

Therefore divide and conquer reduced the original problem of (vector) size n into two subproblems of size $n/2$. After a solution of the two subproblems is found, i.e. L_k, R_k become available, then from Eq. 4, the determination of the two vector elements A_k and $A_{k+n/2}$ requires three additional operations (one multiplication, one addition for A_k, and one subtraction for $A_{k+n/2}$) in order to be found, i.e. a total of $3n/2$ additional complex operations for the $n/2$ pairs of vector elements A_k and $A_{k+n/2}$, for all $0 \leq k \leq n/2 - 1$.

Note that the operations performed are in general complex additions and multiplications. A complex addition requires 2 real additions. A complex multiplication require 4 real multiplications and 2 real additions.

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. Then

$$M_c(n) = 2M_c(n/2) + n/2$$

The solution to this recurrence is $M_c(n) = n \lg n/2$.

Similarly

$$A_c(n) = 2A_c(n/2) + n$$

The solution to this recurrence is $A_c(n) = n \lg n$.

In order to derive the number of real additions and multiplications $A(n)$ and $B(n)$ respectively we take into consideration the observation of the previous page.

We then obtain that $M(n) = 2n \lg n$ and $A(n) = 3n \lg n$. respectively.

Therefore the total number of operations (real additions, subtractions, and multiplications performed) is the total of $T(n) = M(n) + A(n) = 5n \lg n$ operations as originally claimed.

```
   // w_n is n-th root of unity ; w_n/2 is n/2-th root of unity.
   FFT (a,n) // a is a  vector a=[a_0..a_n-1] of n points
1. if (n==1)
2.   return (a);
3. w_n = exp(2*pi*i/n); // pi=3.14... i is i*i=-1 complex number.
4. w   = 1;
5. l = [ a_0 , a_2 , ... , a_n-2 ]; // L is a      vector
6. r = [ a_1 , a_3 , ... , a_n-1 ]; // R is also a vector
7. L = FFT(l,n/2);
8. R = FFT(r,n/2);
9. for(k=0;k<n/2;k++) {
10. A[k]     = L[k] + w R[k];
11. A[k+n/2] = L[k] - w R[k];
12. w = w * w_n;
13 }
14.return(A);
```

In the code above, all variables are complex and so are the matrices/vectors. The operation in line 3 is not exponentiation; the cosine and sine of real numbers need to be established.

So far we have solved the following polynomial evaluation.

Let $A(x) = a_{n-1}x^{n-1} + \ldots + a_1 x + a_0$ be an $n-1$-degree polynomial in indeterminate x.

The evaluation of A at the n-th roots of unity w_n^i yields the n point-value pairs $(w_n^i, f(w_n^i))$, where $A(w_n^i) = a_{n-1}w_n^{i(n-1)} + \ldots + a_1 w_n^i + a_0$ and therefore

$$\begin{bmatrix} A(w_n^0) \\ \ldots \\ A(w_n^{n-1}) \end{bmatrix} = F_n(w_n) \begin{bmatrix} a_0 \\ \ldots \\ a_{n-1} \end{bmatrix}$$

Given the n point-value pairs at the roots of unity for A, the interpolation problem determines the coefficients of A, i.e. the coefficient vector of a_i's. This is expressed as a matrix operation as follows.

$$\begin{bmatrix} a_0 \\ \ldots \\ a_{n-1} \end{bmatrix} = F_n^{-1}(w_n) \begin{bmatrix} A(w_n^0) \\ \ldots \\ A(w_n^{n-1}) \end{bmatrix}$$

The question that arises is how much $F_n^{-1}(w_n)$ is, and how efficiently the new matrix vector product can be computed.

We know from prior discussion that the form of F is

$$F_n(w_n) = (w_n^{ij}).$$

i.e. the element at row i and column j of $F_n(w_n)$ is w_n^{ij}. Matrices with such a form are known in matrix algebra as Vandermonde matrices. Then, the inverse of $F_n(w_n)$ is given by

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

and therefore

$$F_n^{-1}(w_n) = (w_n^{-ij}/n).$$

Therefore in order to establish the a_i's from the A_i's we switch the roles of A and a in the FFT pseudocode, replace w_n by $1/w_n$ and divide each element of the result by n.

Interpolation at roots of unity

If FFT(a,w,n,A) is the function that returns A given a, w, n, then the function that interpolates polynomial a, i.e. returns vector a given A, w, n can be defined as follows. This shows that Interpolation is also an FFT problem plus $O(n)$ additional operations.

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

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

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


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

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

We distinguish two cases.

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

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

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.

Therefore polynomial interpolation is as difficult as polynomial evaluation (at the n-th roots of unity): it requires the solution of an FFT problem on the inverses of the n-th roots of unity (which are also roots of unity).

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 $B(x) = b_{m-1}b^{m-1} + \ldots b_1 x + b_0$,

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 $\vec{b} = (b_0, \ldots, b_{m-1})$, computer vector $\vec{c} = (c_0, \ldots, c_{n+m-2})$, where the elements of vector $\vec{c}$ are given by Eq 5.

$$c_j = \sum_{k=0}^{j} a_k b_{j-k} \tag{5}$$

Vector $\vec{c}$ is also called the **convolution vector**.

Polynomial Multiplication and Convolution

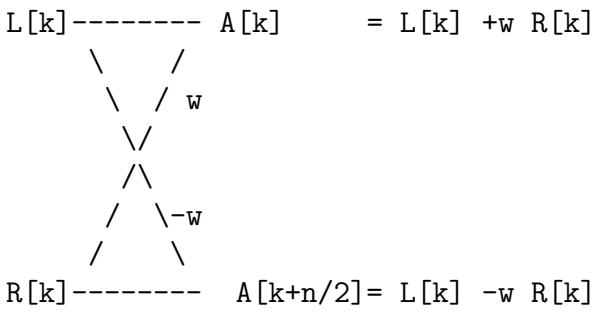
Polynomial multiplication can be solved as follows.

- 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.
- 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.

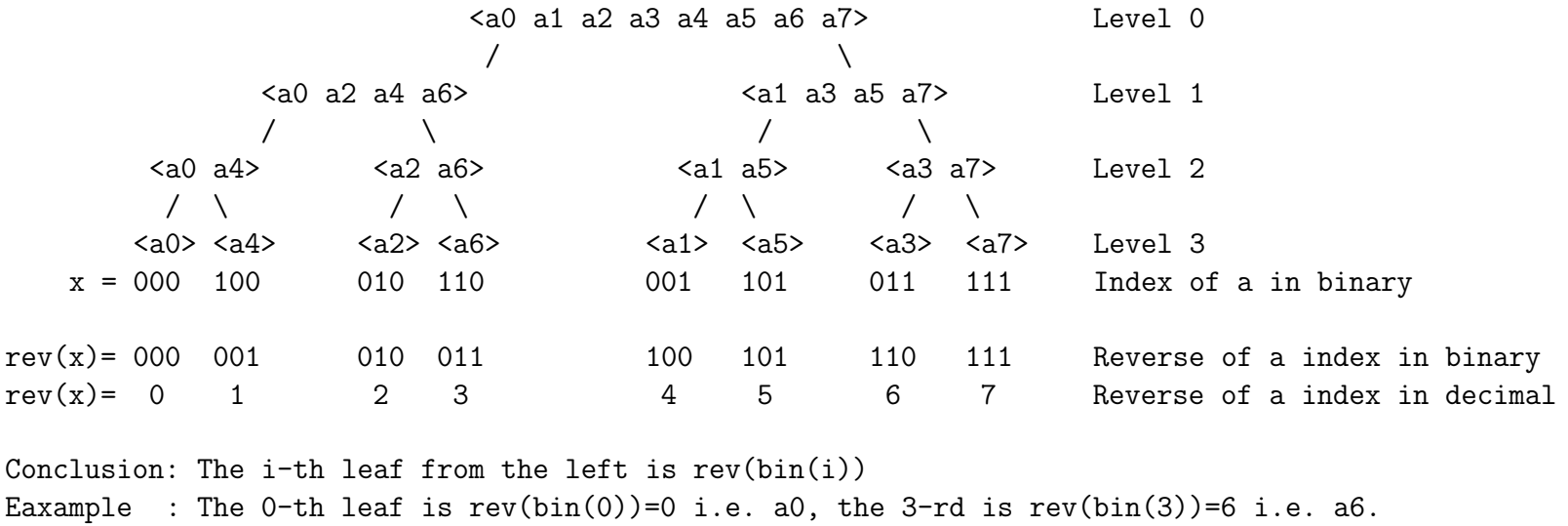
Theorem. $A(x)B(x)$ can be computed in $15N \lg N + O(N)$ real operations, where $N = deg(A) + deg(B) + 1$.

Convolution. Convolution is asymptotically as difficult as polynomial evaluation.

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)

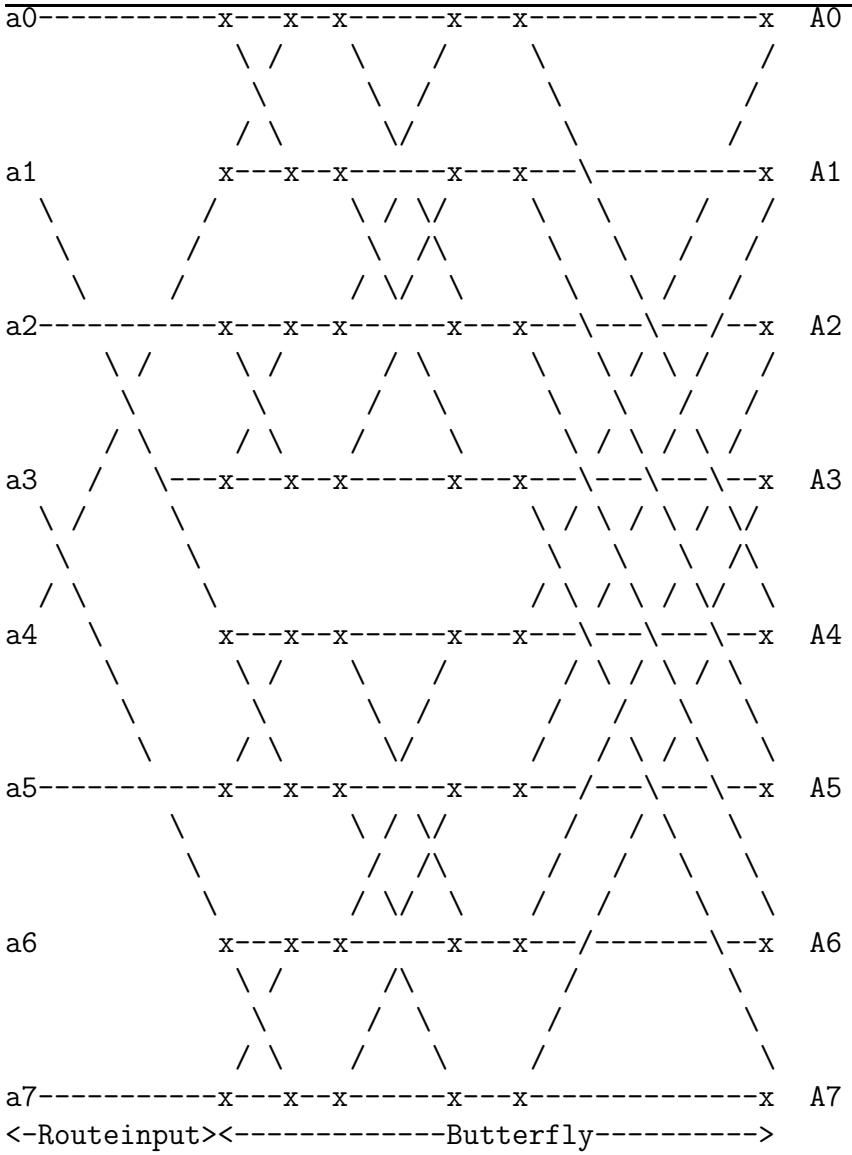
We also observe the following about the bit representation of the indices $(0 \ldots 7)$. For x a binary string of zeroes and ones, let $rev(x)$ be the reverse string of x where x is written in binary, i.e. $rev(100) = 001$.

```
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.

FFT on the butterfly



FFT on the butterfly

The butterfly network was introduced for the purpose of performing FFTs. Each parallel step of an n point FFT is carried out on one level of a $\lg n$-dimensional butterfly. For the sake of definition, let $bin(k)$ be the binary representation of decimal integer k representing a row of a butterlfly. Let $rev(i)$ be the reverse of $bin(k)$.

In the butterfly network that computes $\vec{A}$, input $\vec{a}$ is input through the vertices of level $\lg n$ and output is obtained through the vertices of level 0. This reverse butterfly network is depicted in the figure of the previous page so that level $\lg n$ appears on the left and level 0 on the right.

Therefore level 0 of a $\lg n + 1$-level butterfly computes n-element vector A.

The interesting observation that was made previously and will be proven rigorously on the following page is that a_k is input through line rev(bin(k)) of the butterfly.

The proof is going to be constructive and will use induction. As in level 0 outputs A_k are computed, the inputs in level 1 will be L_k in the top $n/2$ rows and R_i in the bottom $n/2$ rows. As $A_k = L_k + w_n^k R_k$ and $A_{k+n/2} = L_k + w_n^{k+n/2} R_k = L_i - w_n^k R_k$, the computations performed along the cross and level edges are obvious.

Let us assume by induction (inductive hypothesis) that a $\lg n$ level buttefly computes $\vec{L}$ or $\vec{R}$ on $n/2$ inputs. Then a $\lg n + 1$-level butterfly on n inputs, consists of two $\lg n$-level butterflies if level 0 nodes are removed. Let $L_0, L_1, \ldots L_{n/2-1}$, $R_0, R_1, \ldots R_{n/2-1}$ be the outputs of the two butterflies in the inductive step. The operation (related to level 0 and 1) discussed in the previous paragraph shows that A_k is going to be output in the k-th row of level 0 of the $\lg n + 1$-level butterfly.

Question Given that the top $n/2$ butterfly computes $\vec{L}$ and the bottom $\vec{R}$ what are the inputs to each such butterfly?

Answer Vector $\vec{L}$ requires among all $a_0, \ldots a_{n-1}$ only the even indexed elements and $\vec{R}$ the odd-indexed elements. The former have a 0 in the 1-st rightmost bit poistion (also known as lsb position); the latter have an 1 there. The former even-indexed elements are to be associated with the top butterfly; all rows of the top butterfly have a 0 in the 1-st leftmost bit bosition. The latter odd-indexed a_k elements are to be associated with the bottom butterfly; all rows of the bottom butterfly have an 1 in the 1-st leftmost bit bosition. That is the 1-st leftmost bit position of the butterfly input line is the same as the 1-st rightmost bit position of the elements that will be associated with it.

Repeat this for the second recursive unfolding and so on. Then, the i-th rightmost bit position of an element a must be equal to the i-th leftmost bit position of the row this a is input.

Unfolding the recursion we get that a_i is input to row $rev(bin(i))$ of the butterfly.

A node $\langle b, j\rangle$ of the butterfly performs the following computation.

- the high-numbered node of previous level (i.e. the one with 1 in $j+1$-st bit position of b) is multiplied with $w^i_{n/2^j} = w^{i2^j}_n$.
- the low-numbered node of previous level (i.e. the one with 0 in $j+1$-st bit position of b) is then added to this result.

Remark. $w_n^{i2^j}$ is a function of row i and level j. Therefore all the values can be precomputed in time $O(\lg n)$ and stored locally.

Question. Hypercube implementation?

