(Based on the presentation in *Introduction to Algorithms* by Cormen, Leiserson and Rivest.)

**Motivation:** Representing an \( n - 1 \) degree polynomial \( A(x) \) with the \( n \) coefficients \( a_0 \) through \( a_{n-1} \) allows polynomial evaluation and addition to be done in \( \Theta(n) \) time, but the straightforward method for polynomial multiplication takes \( \Theta(n^2) \) time.

Representing an \( n - 1 \) degree polynomial \( A(x) \) with \( n \) point-value pairs \( (x_0, A(x_0)) \) through \( (x_{n-1}, A(x_{n-1})) \) allows polynomial addition to be done in \( \Theta(n) \) time, assuming the two polynomials to be added are represented by pairs with the same \( x \)'s. Polynomial multiplication can also be done in \( \Theta(n) \) time assuming, in addition, that \( 2n - 1 \) pairs are available for each polynomial, since the product polynomial has degree \( 2n - 2 \). However, evaluating the polynomial for an arbitrary \( x \) takes time \( \Omega(n^2) \).

We'd like to have the best of both worlds: Represent polynomials using the coefficients. When multiplication must be done, evaluate the polynomials at \( 2n \) carefully chosen points in \( O(n \log n) \) time using the FFT algorithm, do multiplication using the resulting pairs, and then extract the coefficients from the resulting pairs (called interpolation) in \( O(n \log n) \) time using the inverse FFT algorithm.

**The DFT:** Let \( m = 2n \). For convenience, assume \( m \) is a power of 2.

The carefully chosen points are powers of the complex number

\[
\omega_m = e^{2\pi i/m} = \cos(2\pi/m) + i \cdot \sin(2\pi/m),
\]

where \( i \) is the square root of \(-1\). The powers of interest are \( \omega_m^0, \omega_m^1, \omega_m^2, \ldots, \omega_m^{m-1} \).

The **Discrete Fourier Transform** of \( m \)-vector \( a \), denoted \( \text{DFT}(a) \), is defined to be the \( m \)-vector \( y \) whose \( k \)-th element is

\[
y_k = \sum_{j=0}^{m-1} a_j \cdot (\omega_m^k)^j
\]

for \( k = 0, \ldots, m - 1 \).

Therefore, when \( a \) consists of the coefficients \( a_0 \) through \( a_{n-1} \) of an \( n - 1 \) degree polynomial \( A \), padded with \( n \) zeroes, the \( k \)-th element of \( \text{DFT}(a) \) equals \( A(\omega_m^k) \), the polynomial \( A \) evaluated at \( \omega_m^k \).

**The FFT:** The Fast Fourier Transform (FFT) is an algorithm to compute the DFT. Here's the recursive version of the FFT. The input is an \( m \)-vector \( a \) and the output is an \( m \)-vector \( y \) such that \( y = \text{DFT}(a) \).

function Recursive-FFT(a):
  if \( m = 1 \) then return \( a \) endif
  \( p := \text{Recursive-FFT}([a_0, a_2, a_4, \ldots, a_{m-2}]) \)
  \( q := \text{Recursive-FFT}([a_1, a_3, a_5, \ldots, a_{m-1}]) \)
  for \( k := 0 \) to \( m/2 - 1 \) do
    \( y_k := p_k + \omega_m^k \cdot q_k \) (line 1)
    \( y_{k+m/2} := p_k - \omega_m^k \cdot q_k \) (line 2)
  endfor
  return \( y \)
Running Time: Let $T(m)$ be the running time on input of size $m$. Then $T(1) = 1$ and $T(m) = 2 \cdot T(m/2) + c \cdot m$ for some constant $c$. Solving this recurrence using the standard techniques shows that $T(m) = O(m \log m)$.

Correctness: We must show that $y = \text{DFT}(a)$. We use induction on $m$, the size of $a$.

Basis: $m = 1$. When $m = 1$, $a = [a_0]$. The vector DFT($a$), by definition, consists solely of $a_0 \cdot (\omega_m^k)^0$. This equals $a_0$, which is what the algorithm returns.

Inductive hypothesis: Assume that for input of size $m/2$, the algorithm returns the DFT of its input.

Inductive step: We must show that for input of size $m$, the algorithm returns the DFT of its input. By the inductive hypothesis, the recursive calls that define the vectors $p$ and $q$ are correct. Thus, for $k$ between 0 and $m-1$,

$p_k$ equals the polynomial $P$ evaluated at $\omega_m^k$, where $P(x) = a_0 + a_2 \cdot x + a_4 \cdot x^2 + \ldots + a_{m-2} \cdot x^{m/2-1}$, and

$q_k$ equals the polynomial $Q$ evaluated at $\omega_m^k$, where $Q(x) = a_1 + a_3 \cdot x + a_5 \cdot x^2 + \ldots + a_{m-1} \cdot x^{m/2-1}$.

Check the values computed in line 1:

$y_k = p_k + \omega_m^k \cdot q_k$ by the code

$= a_0 + a_2 \cdot \omega_m^k + a_4 \cdot (\omega_m^k)^2 + \ldots + a_{m-2} \cdot (\omega_m^k)^{m/2-1} + \omega_m^k(a_1 + a_3 \cdot \omega_m^k + a_5 \cdot (\omega_m^k)^2 + \ldots + a_{m-1} \cdot (\omega_m^k)^{m/2-1})$, by the inductive hypothesis.

This expression is in terms of $\omega_m^k$. We want an expression in terms of $\omega_{m/2}^k$. To convert, use the identity that $\omega_m^k = e^{2\pi i k/(m/2)} = e^{2\pi 2k/m} = \omega_{m/2}^k$.

After plugging in and doing some algebra, we get that

$y_k = a_0 + a_1 \cdot \omega_m^k + \ldots + a_{m-1} \cdot (\omega_m^k)^{m-1}$. Since this equals $A(\omega_m^k)$, this is correct.

Now check the values computed in line 2:

$y_{k+m/2} = p_k - \omega_m^k \cdot q_k$ by the code

$= p_k + \omega_{m+m/2}^k \cdot q_k$, since $(-1) \cdot \omega_m^k = \omega_{m/2} \cdot \omega_m^k$

$= a_0 + a_2 \cdot \omega_{m/2}^k + a_4 \cdot (\omega_{m/2}^k)^2 + \ldots + a_{m-2} \cdot (\omega_{m/2}^k)^{m/2-1} + \omega_{m/2}^k(a_1 + a_3 \cdot \omega_{m/2}^k + a_5 \cdot (\omega_{m/2}^k)^2 + \ldots + a_{m-1} \cdot (\omega_{m/2}^k)^{m/2-1})$, by the inductive hypothesis.

This expression is in terms of $\omega_{m/2}^k$. We want an expression in terms of $\omega_{m+m/2}^k$. To convert, use the identity that $\omega_{m/2}^k = \omega_{m/2}^{2k} = 1 \cdot \omega_m^{2k} = \omega_m^k \cdot \omega_m^{2k} = \omega_m^{2k+m} = (\omega_{m+m/2}^k)^2$.

After plugging in and doing some algebra, we get that

$y_{k+m/2} = a_0 + a_1 \cdot \omega_{m+m/2}^k + \ldots + a_{m-1} \cdot (\omega_{m+m/2}^k)^{m-1}$. Since this equals $A(\omega_{m+m/2}^k)$, this is correct.
Ch 32 - Polynomials & the DFT

Can represent a polynomial: \( p(x) = a_0 + a_1x + \ldots + a_{n-1}x^{n-1} \) of degree \( n-1 \) (largest power of \( x \) w/ nonzero coeff) 2 ways:

- \( \text{coeffs} : a_0, a_1, \ldots, a_{n-1} \)
- \( \text{with } n \text{ point-value pairs:} \)
  \[
  (x_0, p(x_0)), (x_1, p(x_1)), \ldots, (x_{n-1}, p(x_{n-1}))
  \]
where the \( x_j \)'s are distinct points.
(The reason why \( n \) points uniquely determine a polynomial is by Thm 32.1; won't cover it in class.)

Operations on polys:

- Evaluate at some point
- Add two polys:
  \[
  A(x) + B(x) = C(x), \quad \text{where } C_j = a_j + b_j, 0 \leq j \leq \max(\deg(A), \deg(B))
  \]
- Multiply two polys:
  \[
  A(x) \cdot B(x) = C(x), \quad \text{where}
  C_j = \sum_{k=0}^{j} a_k b_{j-k}, 0 \leq j \leq \deg(A) + \deg(B)
  \]

Let's see how we can do these ops. in the 2 representations:

- \( \text{eval: use Horner's rule} \quad a_0 + x_0a_1 + x_0^2a_2 + \ldots \) \( \Theta(n) \) time
- \( \text{add: add corresponding coeffs} \quad \Theta(n) \) time
- \( \text{mul: } \Theta(n^2) \) time
point-value pairs:
- eval: \textit{interpolate} (convert to coeff form) + evaluate. Interpolation can be done in $O(n^2)$ time (see text for details).
- add: add the $n$ values, $\Theta(n)$ time (assuming the 2 polynomials are represented using the same points $y_1, \ldots, y_n$).
- mult: multiply the values, $\Theta(n)$ time, assuming
  - same set of points and
  - we have enough values. How many are enough? $\log(A) + \log(B) + 1$
  (i.e., if $A$ and $B$ both have degree $n-1$, we need $2n-1$ points for each).

Let's try to get best of both worlds, i.e., use coeff form, but speed up time for mult.

Show how to do evaluation + interpolation in $\Theta(n \log n)$ time using FFT alg.
The particular points we'll use to evaluate the polynomials are taken from the complex plane + lie on a circle of unit radius. Let \( i = \sqrt{-1} \).

For convenience, let \( m = 2n \).

Example: \( m = 8 \)

Take 8 points on the circle centered at the origin, equally spaced.

From trig, the point between 1 and \( i \) is \( \cos\left(\frac{\pi}{4}\right) + i \cdot \sin\left(\frac{\pi}{4}\right) \).

In general, the \( k \)th point, going counter-clockwise from 1, is \( \cos\left(2\pi \cdot \frac{k}{m}\right) + i \cdot \sin\left(2\pi \cdot \frac{k}{m}\right) \).

**Fact from calculus:** \( \cos \theta + i \sin \theta = e^{i \theta} \)

where \( e \) is the base of the natural log.

Thus the points on the circle are \( e^{2\pi i k/m} \), \( k = 0, 1, \ldots, m-1 \).

**Notation:** \( \omega_m = e^{2\pi i / m} \). So points are \( \omega_m^k \), \( k = 0, 1, \ldots, m-1 \).

These are called complex \( m \)th roots of unity.

Why? \( (e^{2\pi i k/m})^m = e^{2\pi i k} \)

\[ = \left( \cos(2\pi) + i \cdot \sin(2\pi) \right)^k \]

\[ = (1 + 0)^k \]

\[ = 1. \]
Sec. 32.2 proves several other important properties of complex roots of unity.

So getting back to poly. mult., we need to evaluate $A(x)$ and $B(x)$ at $\omega^0, \omega^1, \ldots, \omega^{m-1}$.

The resulting vector of values is called the **Discrete Fourier Transform**, i.e.,

$$\text{DFT}(A) = y, \text{ where } y_k = A(\omega^k) = \sum_{j=0}^{m-1} a_j (\omega^j)^k.$$  

vector $y$  of values  \begin{align*}
&\text{vector } A \text{ of poly. } \text{coeffs. of } n \text{ O's} \\
&0 \leq k \leq m-1
\end{align*}

The **Fast Fourier Transform** is alg. to compute DFT:

- **Recursive FFT**:  
  - **Input:** vector $a$ of $m$ coeffs.; $m$ a power of 2  
  - **Output:** vector $y = \text{DFT}(a)$
  - if $m=1$ then return $a$
  - $p := \text{Recursive-FFT}([a_0, a_2, \ldots, a_{m-2}])$
  - $q := \text{Recursive-FFT}([a_1, a_3, \ldots, a_{m-1}])$
  - for $k := 0 \rightarrow \frac{m}{2} - 1$ do
    - $y_k := p_k + \omega^k \cdot q_k$ \hspace{1cm} (1)
    - $y_{k+\frac{m}{2}} := p_k - \omega^k \cdot q_k$ \hspace{1cm} (2)
  - end for
  - return $y$

What is this doing? Why is it correct?

Check for yourself why $m=1$ case is correct.

Assuming the recursive calls are correct,

$p_k = P(\omega^k), \text{ when } P(x) = a_0 + a_2 \cdot x + a_4 \cdot x^2 + \ldots + a_{m-2} \cdot x^{\frac{m}{2}-1}$  

+ $q_k = Q(\omega^k), \text{ where } Q(x) = a_1 + a_3 \cdot x + a_5 \cdot x^2 + \ldots + a_{m-1} \cdot x^{\frac{m}{2}-1}$
Check the values computed in line (1), $0 \leq k \leq \frac{m}{2} - 1$

So $y_k = p_k + w_m^k \cdot g_k$

$= a_0 + a_2 \cdot w_m^k + a_4 \cdot (w_m^k)^2 + \ldots + a_{m-2} \cdot (w_m^k)^{m-2} + \ldots + a_{m-1} \cdot (w_m^k)^{m-1}$

Fact: $w_{m/2}^k = w_m^k \quad \text{and} \quad w_{m/2}^k = e^{2\pi i k/(m/2)} = e^{2\pi i 2k/m}$

Why? See notes for technical details. For intuition:

look at circle: $-1 = w_8^3$ and $w_4^2$.

$= a_0 + a_2 \cdot w_m^k + a_4 \cdot (w_m^k)^2 + \ldots + a_{m-2} \cdot (w_m^k)^{m-1} + w_m^k \left( a_1 + a_3 \cdot w_m^k + a_5 \cdot (w_m^k)^2 + \ldots + a_{m-1} \cdot (w_m^k)^{m-1} \right)$

$= a_0 + a_2 \cdot w_m^k + a_4 \cdot (w_m^k)^2 + a_3 \cdot (w_m^k)^3 + \ldots + a_{m-1} \cdot (w_m^k)^{m-1}$

$= A(w_m^k)$.

Check the values computed in line (2).

Recall $0 \leq k \leq \frac{m}{2} - 1$.

$y_{k+\frac{m}{2}} = p_k - w_m^k \cdot g_k = p_k + w_m^k \cdot \frac{m}{2} \cdot g_k$ (see 13 Fact below)

$= a_0 + a_2 \cdot w_m^k + a_4 \cdot (w_m^k)^2 + \ldots + a_{m-2} \cdot (w_m^k)^{m-1} + w_m^k \left( a_1 + a_3 \cdot w_m^k + a_5 \cdot (w_m^k)^2 + \ldots + a_{m-1} \cdot (w_m^k)^{m-1} \right)$

Fact: $-w_m^k = (-1)^k \cdot w_m^k = (w_m^{m/2})^k \cdot w_m^k = \omega_m^k = \omega_m^{k+\frac{m}{2}}$

Fact: $w_m^k = 1 \cdot w_m^k = w_m^k \cdot \omega_m^k = \omega_m^{2k}$

Fact: $w_m^{2k} = 1 \cdot w_m^{2k} = w_m^k \cdot \omega_m^k = \omega_m^{2k+m}$

OR: $w_m^{2k} = (-w_m^k)^{2k}$ from previous fact

$= (-1)^2 = 1$
\[ \begin{align*}
&= a_0 + (a_2 \cdot \omega_m^{k+\frac{m}{2}})^2 + a_4 \cdot (\omega_m^{k+\frac{m}{2}})^4 + \ldots + a_{m-2} \cdot (\omega_m^{k+\frac{m}{2}})^{m-2} \\
&+ \omega_m^{k+\frac{m}{2}} (a_1 + a_3 \cdot (\omega_m^{k+\frac{m}{2}})^2 + \ldots + a_{m-1} \cdot (\omega_m^{k+\frac{m}{2}})^{m-2}) \\
&= A (\omega_m^{k+\frac{m}{2}}).
\end{align*} \]

:. Recursively FFT correctly computes $\text{DFT}(a)$.

**Running Time:**

\[ T(n) = 2 \cdot T\left(\frac{n}{2}\right) + \Theta(n) \]

\[ = \Theta(n \log n). \]

Now, how to do interpolation, going from 2$n$ point-value pairs to coeffs?

Can use inverse DFT, which can be computed in a similar way. (HW),

\[ \text{DFT}^{-1}(y) = \sum_{k=0}^{m-1} y_k \cdot \omega_m^{-kj}, 0 \leq j \leq m-1 \]

Rest of Chapter 3.2 discusses optimizations to the FFT alg - replace recursion with iterations, etc.

**Convolution Theorem:** If $a$ and $b$ are vectors of length $n$, then

\[ a \times b = \text{DFT}^{-1}(\text{DFT}(a) \cdot \text{DFT}(b)) \]

\[ \begin{align*}
\text{Convolution of } a &+ b\text{ - the vector } C, \text{ where } \\
c_j &= \sum_{k=0}^{m-1} a_k b_{j-k}.
\end{align*} \]

So polynomial mult. is a special case, an application of convolution $\times$ FFT.