5 March 2021  The Fast Fourier Transform (§ 5.6)

---

**Announcements.**

1. Prof. Kleinberg extra office hour today 11-12. *(My office hour link is on the course website.)*

2. Homework 4 (one problem) will be released within 24 hours, due Friday 3/12.

3. March 9-10 wellness days.
   Note office hour changes on the Google Calendar on 4820 website.

---

We saw a reduction from multiplying integers to multiplying polynomials with integer coefficients.

Karatsuba’s Algorithm accomplishes both in $O(n \log_2(3)) \approx O(n^{1.58})$. 
Most recent discovery of a faster (asymptotically) integer multiplication algorithm...

2019: Harvey & van der Hoeven announced a $O(n \log n)$ multiplication alg.
Under a well-known conjecture in information theory, this running time
is optimal up to constant factors.

At the heart of this algorithm is the FFT, which can be thought of as a
method for multiplying polynomials using
$O(n \log n)$ arithmetic operations.

Multiplying polynomials is equivalent to convolution of signals.

A "signal" in this lecture is a sequence
of numbers $a_0, a_1, \ldots, a_{n-1}$.
(measurements in discrete time)

Convolution takes
- signal $a_0, \ldots, a_{n-1}$
- "mask" $b_0, \ldots, b_{n-1}$
and outputs $c_0, \ldots, c_{2n-2}$ s.t.
\[ C_k = \sum_{i+j=k} a_i b_j \]

Example: Modify a signal by averaging each number with its 2 neighbors.

\[
\begin{array}{cccccc}
\text{a}_0 & \text{a}_1 & \text{a}_2 & \text{a}_3 & \text{a}_4 \\
5 & 9 & -1 & 2 & 6 \\
\text{b}_0 & \text{b}_1 & \text{b}_2 & \text{b}_3 & \text{b}_4 \\
0 & \frac{1}{3} & \frac{1}{3} & \frac{1}{3} & 0 \\
\end{array}
\]

\[
\begin{array}{cccccc}
c_0 & c_1 & c_2 & c_3 & c_4 & c_5 & c_6 \\
0 & \frac{5}{2} & \frac{14}{3} & \frac{8+9-1}{3} & \frac{9-1+2}{3} & \cdots & \cdots \\
\end{array}
\]

When we multiply polynomials

\[
A(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_{n-1} x^{n-1}
\]

\[
B(x) = b_0 + b_1 x + b_2 x^2 + \cdots + b_{m-1} x^{m-1}
\]

the coefficient of \( x^k \) in the polynomial

\[
C(x) = A(x) B(x)
\]

is

\[
c_k = \sum_{i+j=k} a_i b_j.
\]
So convolving signals is equivalent to multiplying polynomials.

Convolving in \( O(n \log n) \) operations:
use the fact that the coefficients of a polynomial of degree \( d \)
are uniquely determined by its values at \( d+1 \) distinct points.

Recipe for multiplying \( A(x) \cdot B(x) \).

(1) Let \( d = \deg(A) + \deg(B) = \deg(A \cdot B) \).

(2) Select \( d+1 \) distinct numbers \( z_0, \ldots, z_d \).

(3) Evaluate \( A(z_0), \ldots, A(z_d), \quad B(z_0), \ldots, B(z_d) \).

(4) Multiply value to compute \( C(z_0) = A(z_0) \cdot B(z_0) \).

(5) Use polynomial interpolation to find the coefficients of \( C \),
given values \( C(z_0), \ldots, C(z_d) \).
Running time will depend on how fast we can implement steps $3 + 5$.

\[ A(z) = a_0 + a_1 z + a_2 z^2 + \ldots + a_{n-1} z^{n-1} \]

Thus we can evaluate $A(z)$ for any $z$ using $O(n)$ arithmetic ops.

Since $d = \deg(A), \deg(B) = (n-1) + (n-1) = 2n - 2$.

Evaluating $A(z_0), \ldots, A(z_d)$ naively takes $O((d+1)n) = O(n^2)$ operations.

Divide-and-conquer with carefully chosen $\mathbb{F}_d$ makes it faster.

Complex roots of unity: the numbers

\[ e^{2\pi ik/N} \text{ for } k = 0, 1, \ldots, N-1 \]

are called the complex $N^{th}$ roots of unity. They are the solutions of $z^N = 1$ in the complex numbers.
Fourier transform of signal $a_0, a_1, \ldots, a_{N-1}$ is the sequence of complex numbers

$$A(1), A(\omega), A(\omega^2), \ldots, A(\omega^{N-1})$$

where

$$A(\omega) = a_0 + a_1 \omega + \ldots + a_{N-1} \omega^{N-1}$$

and $\omega = e^{\frac{2\pi i}{N}}$.

When $N$ is a power of 2 you can compute $A(1), A(\omega), \ldots, A(\omega^{N-1})$ in $O(N \log N)$ using divide and conquer, the FFT!
\[ A(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_{N-1} x^{N-1} \]
\[ = \left( a_0 + a_2 x^2 + a_4 x^4 + \ldots + a_{N-2} x^{N-2} \right) A_{\text{even}}(x^{\frac{N}{2}}) \]
\[ + x \cdot \left( a_1 + a_3 x^2 + a_5 x^4 + \ldots + a_{N-1} x^{N-2} \right) A_{\text{odd}}(x^{\frac{N}{2}}) \]

\[ A(x) = A_{\text{even}}(x^{\frac{N}{2}}) + x \cdot A_{\text{odd}}(x^{\frac{N}{2}}) \]

As \( x \) runs through 1, \( \omega, \omega^2, \ldots, \omega^{N-1} \),
\( x^2 \) runs through 1, \( \omega^2, \omega^4, \ldots, \omega^{2N-2} \).

This sequence is 1, \( \omega, \omega^2, \ldots, \omega^{N-2} \), repeated twice.

\[ \text{FFT:} \quad \text{For } N = 2^k, \text{ let } \omega = e^{2\pi i/N}. \]
Given \( A(x) = a_0 + a_1 x + \ldots + a_{N-1} x^{N-1} \),
1. Form
\[ A_{\text{even}}(x) = a_0 + a_2 x + \ldots + a_{N-2} x^{(N-2)/2} \]
\[ A_{\text{odd}}(x) = a_1 + a_3 x + \ldots + a_{N-1} x^{(N-2)/2} \]
2. Recursively use FFT of order \( \frac{N}{2} \)
   to evaluate \( A_{\text{even}}, A_{\text{odd}} \)
   at each of the points
   \( 1, \omega^2, \omega^4, \ldots, \omega^{N-2} \).

3. For each \( x \in \{1, \omega, \omega^2, \omega^3, \ldots, \omega^{N-1}\} \)
   \[
   A(x) = A_{\text{even}}(x) + x \cdot A_{\text{odd}}(x^2).
   \]

\( T(N) \) denotes \# arithmetic ops
   to evaluate \( A(1), A(\omega), \ldots, A(\omega^{N-1}) \).

**Step 1.** No arithmetic. Just splitting
   one array into 2 arrays.

**Step 2.** \( 2 \cdot T\left(\frac{N}{2}\right) \) operations.

**Step 3.** \( N \) addition, \( N \) multiplication.

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

\( \Rightarrow \)

\[
T(N) = O(N \log N).
\]