Reading: 5.6.

Last time:
- Divide and Conquer
- Recurrences
- Mergesort, Integer Multiply

Today:
- Polynomial Multiplication
- Fast Fourier Transform

Convolution and Polynomial Multiplication

Example:
\[ A(x) = -2 + 2x \]
\[ B(x) = 3/2 - x/2 \]
\[ C(c) = A(x) \cdot B(x) = -3 + 4x - x^2 \]

Fact: let
\[ A(x) = a_0 + a_1 x + \ldots + a_{n-1} x^{n-1} \]
\[ B(x) = b_0 + b_1 x + \ldots + b_{n-1} x^{n-1} \]
be degree \( n-1 \) polynomials. Then,
\[ C(x) = A(x) \cdot B(x) \]
\[ = c_0 + c_1 x + \ldots + c_{2n-2} x^{n-2} \]
with
\[ c_k = \sum_{i,j : i+j=k} a_i b_j \]

Def: \( a = (a_0, \ldots, a_{n-1}) \) is an \( n \)-vector.

Def: \( c \) (above) is the convolution of \( a \) with \( b \), denoted \( c = a \ast b \).

Def: for \( a \) and \( b \), the pointwise vector product is \( c = a \cdot b \) with \( c_k = a_j \cdot b_j \).

Runtimes:
- pointwise product: \( O(n) \).
- convolution: \( O(n^2) \). [[optimal?]]

[[can we do convolution faster?]]
Polynomial multiplication via evaluation

Fact: A degree \( n - 1 \) polynomial is uniquely determined by \( n \) points.

Example: \( A(x) = -2 + 2x \) determined by \((1, 0), (2, 2)\).

Interpolate: \( C(x) = -3 + 4x - x^2 \).

[last step comes from “Algebra 2”]

Conclusion: Given
- \( x \)-coordinates \( x_0, \ldots, x_{n-1} \)
- function values \( A_0, \ldots, A_{n-1} \)

(with \( A_i = A(x_i) \))

there is correspondence:
- coefficients \( a = (a_0, \ldots, a_{n-1}) \)
- evaluate \( A = (A_0, \ldots, A_{n-1}) \)
- interpolate \( c \).

Algorithm: Polynomial Mult (degree \( n - 1 \))

1. choose \( x \) as \( 2n - 1 \) points \( x_0, \ldots, x_{2n-2} \)
2. evaluate on \( x \): \( a, b \Rightarrow A, B \).
3. pointwise multiply: \( C = A \cdot B \).
4. interpolate: \( C \Rightarrow c \).

Runtime: \( T(n) = O(n^2) \)

[e.g., evaluate degree \( n \) poly on \( 2n \) points is \( O(n^2) \). need a better idea]

Example:

\[ A(x) = -2 + 2x \]
\[ B(x) = 3/2 - x/2 \]

Evaluate:

<table>
<thead>
<tr>
<th>( x )</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>( A(x) )</td>
<td>0</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>( B(x) )</td>
<td>3/2</td>
<td>1/2</td>
<td>0</td>
</tr>
</tbody>
</table>

Multiply:

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

What degree \( 2 \) poly goes through these points?
Idea: Choose $x$ to make evaluation/interpolation faster.

**Fast Fourier Transform**

Fact (Euler’s Formula):

$e^{i \theta} = \cos \theta + i \sin \theta$

Proof: E.g., via Taylor series, see Wikipedia.

Recall: trigonometry

\[
ed
\]

Example: Evaluate $e^{i \theta}$ at $\theta = \{0, \pi/2, \pi, 3\pi/2, 2\pi\}$

Fact: multiplying $\equiv$ adding angles

$e^{i \theta_1} e^{i \theta_2} = e^{i (\theta_1 + \theta_2)}$

**Fact (Euler’s Identity):**

$e^{i 2\pi} = 1$

[[from Euler’s formula]]

Def: $n$th roots of unity are $e^{ij2\pi/n}$ for $j = 0, \ldots, n-1$.

Fact: $n$th roots of unity are solutions to $x^n = 1$.

[[intuition: multiplying $=\,$ adding angles]]

Proof: $(e^{ij2\pi/n})^n = e^{ij2\pi} = (e^{2\pi})^j = 1^j = 1$.

Idea: use $2n$th roots of unity as $x_0, \ldots, x_{2n-1}$.

**Problem:** Fourier Transform

Input: coefficients of degree $n-1$ poly.

\[
A(x) = a_0 + a_1 x + \ldots + a_{n-1} x^{n-1}.
\]

Output: $A_0, \ldots, A_{n-1}$

with $A_j = A(e^{ij2\pi/n})$. 

Fact: multiplying $\equiv$ adding angles
Divide and Conquer FFT

Idea: write $A(x) = A''(x^2) + xA'(x^2)$
- $A'(.), A''(.)$, degree $n/2 - 1$ polys on $x^2$.
- $x^2$ on $n$th roots of unity
  $\equiv x$ on $n/2$th roots of unity

Formally:
- $A''(x) = a_0 + a_2x + \cdots + a_{n-2}x^{n/2-1}$
- $A'(x) = a_1 + a_3x + \cdots + a_{n-1}x^{n/2-1}$

and

$$A(e^{ij2\pi/n}) = A''(e^{ij2\pi/n^2}) + e^{ij2\pi/n} A'(e^{ij2\pi/n^2})$$

$$= A''(e^{ij\pi/n}) + e^{ij\pi/n} A'(e^{ij\pi/n})$$

$n/2$th root of unity

Subproblems: evaluate $n/2 - 1$ degree
polys $A'(\cdot), A''(\cdot)$ on $n/2$th roots of unity.

Algorithm: FFT (evaluates $n - 1$ degree
poly on $n$th roots of unity)
0. if $n = 1$, return $A_0 = a_o$.
1. divide $a$ into even & odd coefs, $a'$ and $a''$
2. $A' = FFT(a')$; $A'' = FFT(a'')$.
3. for each $n$th root of unity $e^{ij2\pi/n}$:
   $$A_j = A''_j + e^{ij2\pi/n} A'_{j (mod 2)}$$

Runtime: $T(n) = 2T(n/2) + n$
$\Rightarrow T(n) = O(n \log n)$.

Proof: See text.

Algorithm: Poly Mult w. FFT

[use FFT/de-FFT for evaluate/interpolate in poly mult algorithm]

1. take $2n$ bit FFTs: $a, b \Rightarrow A, B$
2. pointwise multiply: $C = A \cdot B$
3. take $2n$ bit de-FFT: $C \Rightarrow c$.

Runtime: $T(n) = O(n \log n)$

Note: FFT with complex roots of unity can have numerical errors, with integer coefs, round solution to be integers.

[Also, can get roots of units via number theory, e.g., integers modulo a prime.]

Note: Can use FFT to integer multiply in $O(n \log^2 n)$.

Poly Mult w. FFT

Claim: can de-FFT ($A \Rightarrow a$) with similar divide and conquer alg.

4