Divide & Conquer

\[ T(n) = aT\left(\frac{n}{b}\right) + f(n) \]

- time to divide into sub-problems
- plus time to combine solutions to sub-problems.

Examples
- Merge-Sort
- Quick-Sort
- Binary search
- Median finding

* Recursive algorithms.
* Proof by induction.
Fast Fourier Transform (Cooley-Tukey, 60's, but was discovered before: Gauss)

- A divide & conquer algorithm, very useful.
- Has n outputs; each requires time \( \Omega(n) \);
  together require only \( O(n \log n) \) time (!)

\[
\text{n} \quad \text{FFT} \quad \text{n}
\]

- By the end of the lecture, you'll know what it is and when to use it.
- We'll work our way there with an example:
  Multiplication

---

**Third-grade multiplication**

\[
\begin{array}{c}
\frac{3}{5} \\
+ \frac{4}{6} \\
\hline \\
\frac{23}{10} \\
\end{array}
\]

<table>
<thead>
<tr>
<th>385</th>
<th>426</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>770</td>
</tr>
<tr>
<td>1540</td>
<td>164010</td>
</tr>
</tbody>
</table>

hard - need to carry...

---

**Junior-high multiplication**

\[
\begin{array}{c}
\frac{3x^2 + 8x + 5}{4x^2 + 2x + 6} \\
\hline \\
18x^2 + 48x + 30 \\
6x^3 + 16x^2 + 10x \\
12x^4 + 32x^3 + 20x^2 \\
12x^4 + 36x^3 + 54x^2 + 58x + 30 \\
\end{array}
\]

no carry!
Running time for both = $O(n^2)$

We'll focus on polynomials multiplication (no need to worry about carry).

With FFT we’ll get: $O(n \log n)$ time!

More work gives $O(n \log n \cdot \log \log n)$ time for multiplication of numbers [Schönhage-Strassen, 71].

Even more ideas give you $O(n \log n \cdot 2^{O(\log^* n)})$ time [Fürer, 2007].

Still open whether integer mult. can be done in $O(n \log n)$ time.
Polynomials

\[ p(x) = a_n x^{n-1} + a_{n-2} x^{n-2} + \ldots + a_0, \quad a_{n-1} \neq 0, \text{ degree } = n - 1 \]

Observation

For every sequence \((x_1, y_1), \ldots, (x_n, y_n)\), \(x_i \neq x_j\) different, there is a unique poly \(p\) of degree \(\leq n-1\) that satisfies:

\[ p(x_i) = y_i \quad \text{for } i = 1, 2, \ldots, n \]

\[ p(x) = ax + b \]

\[ p(x) = ax^2 + bx + c \]

Polynomial Multiplication Idea

If we represent any polynomial \(p\) by

\((x_1, p(x_1)), \ldots, (x_n, p(x_n))\)

then multiplication takes \(O(n)\) time.

\((x_1, p(x_1)) - \cdots - (x_n, p(x_n)) \Rightarrow (x_1, p(x_1)q(x_1)) - \cdots - (x_n, p(x_n)q(x_n))\)
Coefficients \(a_0, a_1, a_2, \ldots, a_n\) \[\text{FFT} \] \[p(w^n), p(w^n^2), p(w^n^{n-1}) \]
evaluations on \(n\)'th roots of unity

\[W_n = e^{\frac{2\pi i}{n}}\] is \(n\)'th root of unity.

**Important Property:** Squaring the \(n\)'th roots of unity gives the \(\left(\frac{n}{2}\right)\)'th roots of unity:

\[W_n^2 = e^{\frac{2\pi i}{n/2}} = W_n^{n/2}\]

**FFT**

1. If \(n=1\), return \(a_0\)

2. Write:
   \[P_{\text{odd}}(x) = a_1 + a_3 x + a_5 x^2 + a_7 x^3 + \ldots \] \(\deg \leq \frac{n}{2}\)
   \[P_{\text{even}}(x) = a_0 + a_2 x + a_4 x^2 + a_6 x^3 + \ldots\]
   
   So:
   \[p(x) = P_{\text{even}}(x^2) + x P_{\text{odd}}(x^2) \] (*)

3. Run FFT on \(a_0, a_2, a_4, \ldots\) \(P_{\text{even}}\)

4. Run FFT on \(a_1, a_3, a_5, \ldots\) \(P_{\text{odd}}\)

5. Compute \(p(W_n^k) = P_{\text{even}}(W_n^{k/2}) + W_n^k P_{\text{odd}}(W_n^{k/2})\) for \(k = 0, 1, \ldots, n-1\) \((\text{Using } (*))\)
Running time

\[ T(n) = 2T(\frac{n}{2}) + \Theta(n) = \Theta(n \log n) \]

What about computing the inverse?

Evaluations of poly on roots of unity

\[ p(w_n^0) \quad p(w_n^1) \quad \cdots \quad p(w_n^{n-1}) \]

\[ a_0 \quad a_1 \quad \cdots \quad a_{n-1} \]

Coefficients of unique poly defined by evals

Turns out \( \text{FFT}^{-1} \approx \text{FFT} \). Why?

\[ \text{FFT} = \begin{pmatrix} w_n^0 & w_n^1 & \cdots & w_n^{n-1} \\ w_n^1 & w_n^2 & \cdots & w_n^{n} \\ \vdots & \vdots & \ddots & \vdots \\ w_n^{n-1} & w_n^{n-2} & \cdots & w_n^0 \end{pmatrix} = (w_n^i)_{ij} \]

Linear transformation

\[ \text{FFT}^{-1} = \begin{pmatrix} w_n^{-0} & w_n^{-1} & \cdots & w_n^{-(n-1)} \\ w_n^{-1} & w_n^{-2} & \cdots & w_n^{-(n)} \\ \vdots & \vdots & \ddots & \vdots \\ w_n^{-(n-1)} & w_n^{-(n-2)} & \cdots & w_n^{-0} \end{pmatrix} = (\frac{1}{n} \cdot w_n^{-ji})_{ij} \]

Indeed, \( (\text{FFT} \times \text{FFT}^{-1})_{ij} = \sum_{k} w_n^{ik} \cdot \frac{1}{n} w_n^{-jk} = \frac{1}{n} \sum_{k} w_n^{k(i-j)} = \begin{cases} 1 & i = j \\ 0 & i \neq j \end{cases} \)
To compute the inverse:

\[
\begin{pmatrix}
\frac{1}{n} w^{-ji}
\end{pmatrix}
\begin{pmatrix}
p(w^n_0) \\
p(w^n_1) \\
\vdots \\
p(w^n_{n-1})
\end{pmatrix}
= \frac{1}{n}
\begin{pmatrix}
\overline{w^n}_j \\
\overline{w^n}_j \\
\vdots \\
\overline{w^n}_j
\end{pmatrix}
\begin{pmatrix}
p(w^n_0) \\
p(w^n_1) \\
\vdots \\
p(w^n_{n-1})
\end{pmatrix}
\]

\[W_n^* = \overline{W_n} \quad \text{(conjugate)}\]

because \(\frac{1}{a+ib} = \frac{a^2+b^2}{a^2+b^2} \cdot \frac{a-ib}{a-ib} \quad \text{conjugate}\)

\[
\frac{1}{n}
\begin{pmatrix}
\overline{w^n}_j \\
\overline{w^n}_j \\
\vdots \\
\overline{w^n}_j
\end{pmatrix}
\begin{pmatrix}
p(w^n_0) \\
p(w^n_1) \\
\vdots \\
p(w^n_{n-1})
\end{pmatrix}
\]

\[x \cdot y = \overline{x} \cdot \overline{y}\]

FFT:\n
1. Conjugate \(p(w^n_0), \ldots, p(w^n_{n-1})\)
2. Apply FFT on conjugate.
3. Take conjugate of the result.
4. Multiply by \(\frac{1}{n}\).
Fourier Transform in General

\[
\begin{array}{cccccccc}
  & 0 & 1 & 2 & \ldots & n-1 & n & \ldots & 2n-2 \\
p & a_0 & a_1 & a_2 & \ldots & 0 & 0 & \ldots & 0 \\
q & b_0 & b_1 & b_2 & \ldots & 0 & 0 & \ldots & 0 \\
\end{array}
\]

Coefficient of \( x^i \) in \( p \cdot q \) is

\[
\sum_{j} p(j)q(i-j)
\]

Arithmetic is \( \text{mod } 2n-1 \).

\((p \ast q)(i) = \frac{1}{2n} \sum_{j \in \mathbb{Z}_{2n-1}} p(j)q(i-j)\) is called the "convolution" of \( p \) & \( q \).

For every \( i \), computing \((p \ast q)(i)\) requires time \( \Omega(n) \). FFT lets you compute \((p \ast q)(i)\) for all \( 0 \leq i \leq 2n-2 \) in time \( \Theta(n \log n) \).

FFT is useful because convolutions occur frequently.
Appendix!

Convolution for "smoothening"/compression

Idea

I have a very complicated audio

I'll smoothen it, so I get something similar but much easier to describe

HUGE AUDIO
FICE

MP3

\[(f * g)(n) = \frac{1}{2^n} \sum_{j} f(j) g(i-j)\]

New audio
Original audio
Weight

When everything is continuous, you have an integral here instead of a sum.

value here = average over values in other points, where each other point is weighted according to its difference from i