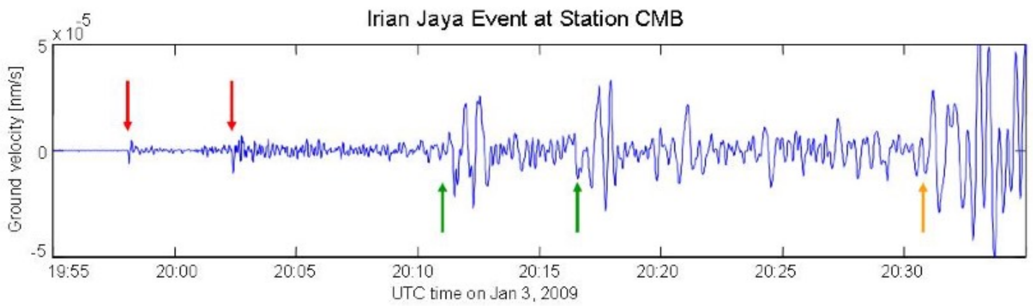
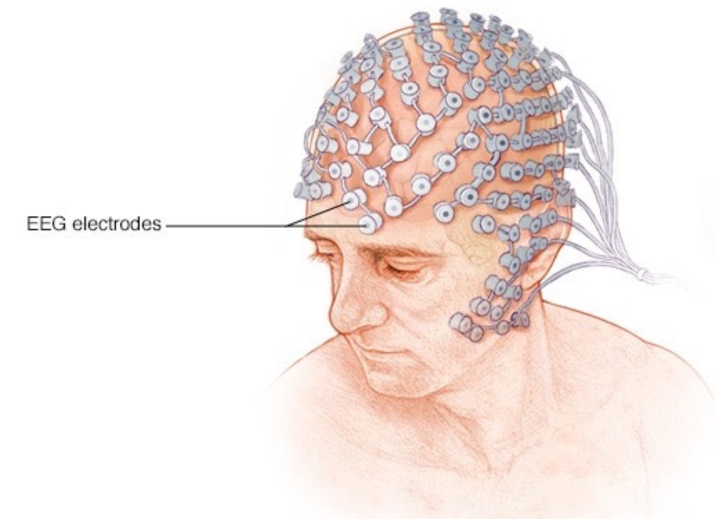
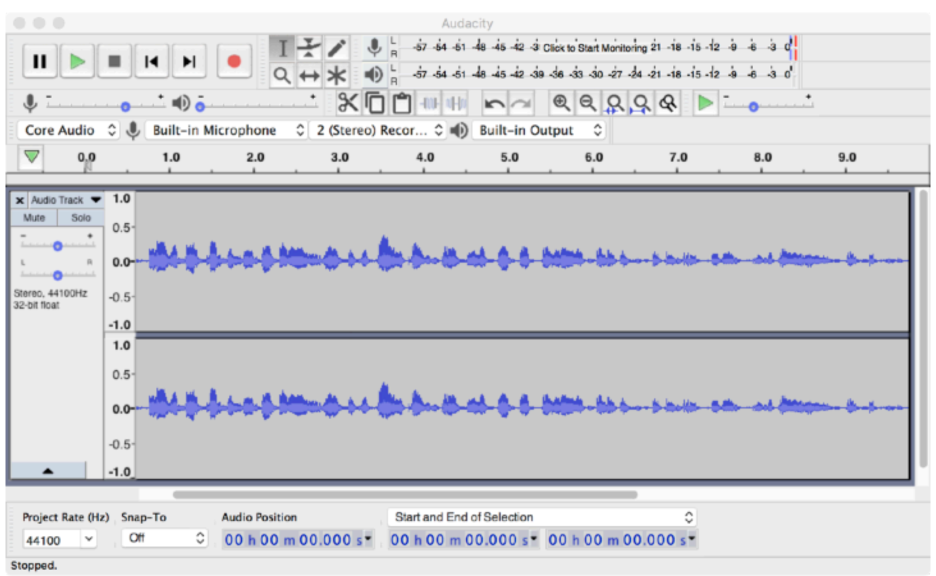
CSE202
Design and Analysis of Algorithms

Week 3 — Divide & Conquer 2:
The Fast Fourier Transform (FFT) -
From Sound Compression to Polynomial Multiplication

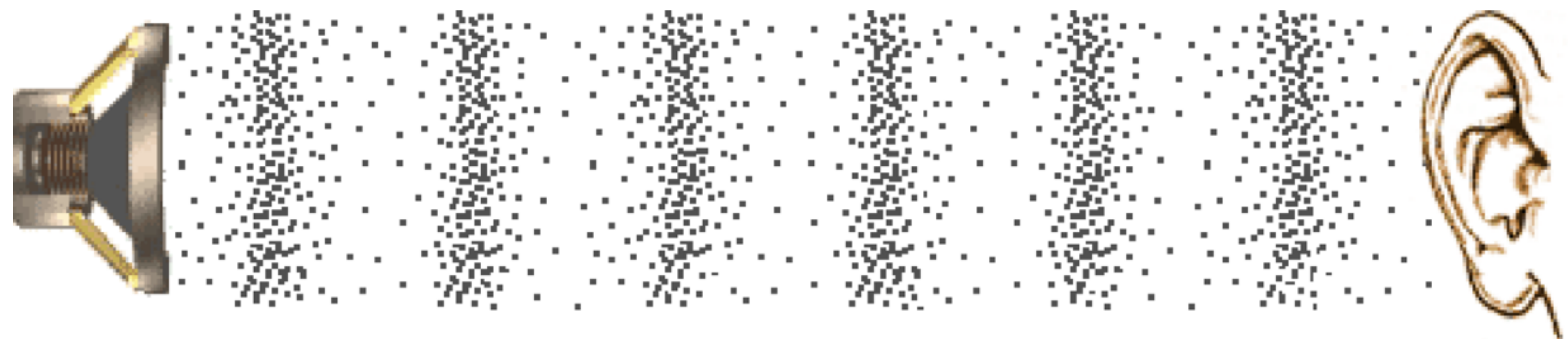
Signals are Everywhere







Sound: Frequency and Amplitude



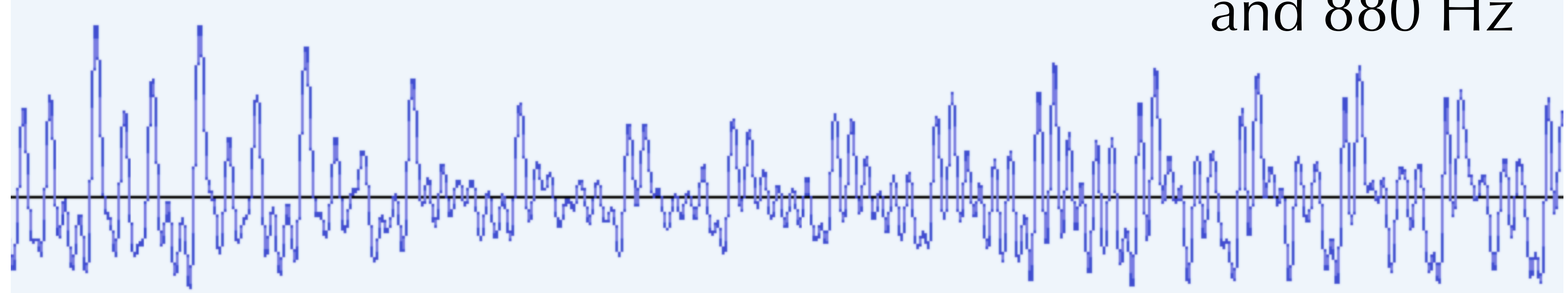
A pure note

Concert A:
440 Hz

Superposition of notes

Harmonics:
440 Hz
and 220 Hz
and 880 Hz

Voice

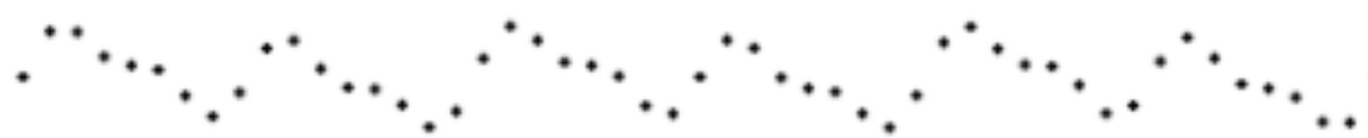


Fourier transform decomposes *periodic* signals as linear combinations of sines and cosines.

Discrete Signals

Functions from some $D \subseteq \mathbb{Z}^m$ to $\mathbb{R}^n$

often obtained by sampling a continuous signal



(Here, $m = n = 1$).

Digital sound is often sampled at 44,100 samples/sec.

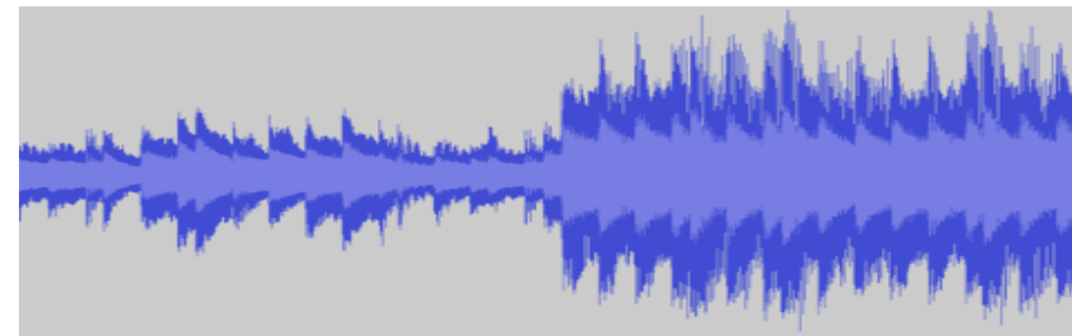
Tools that
- filter out low/high frequency
- change pitch or speed
- compress (e.g., MP3)

all rely on FFT.

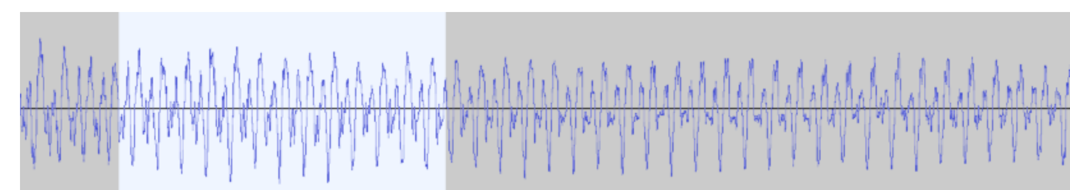
One of the top 10 algorithms of the 20th century.
IEEE J. Comput. in Science & Engineering

A Very Simplified View of Sound Compression

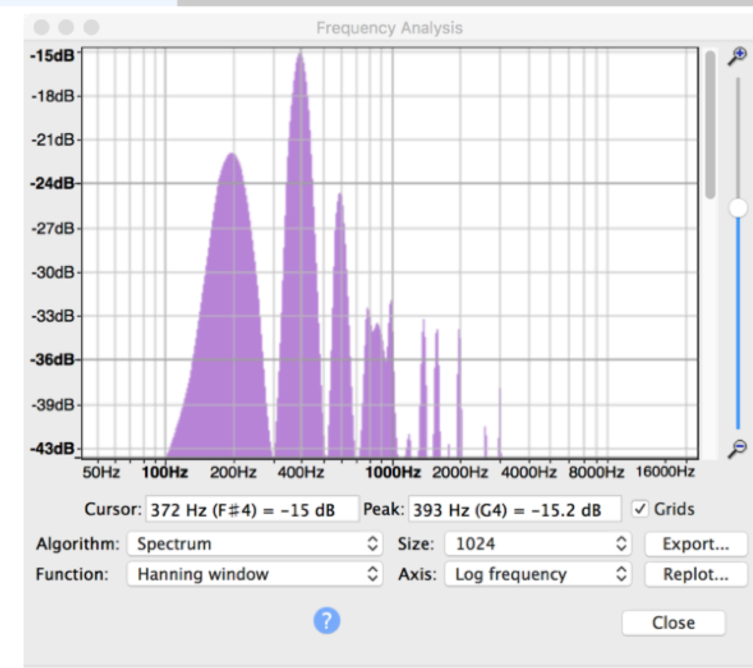
1. Sound is sampled



2. The samples are split into smaller windows



3. Their spectrum is computed by FFT



4. And then compressed by removing low intensity, frequencies that are too high or too low…

You can experiment with this using Audacity (free).

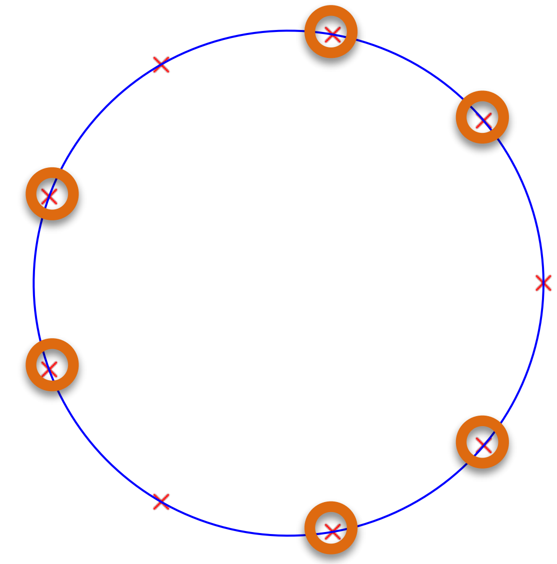
II. Discrete Fourier Transform

Primitive Roots of Unity

Def. $\omega \in \mathbb{C}$ is a nth root of unity if $\omega^n = 1$.

It is *primitive* if moreover,

$$\omega^t \neq 1 \text{ for } t \in \{1, \ldots, n-1\}.$$



$n = 9$

Prop. If ω is a primitive nth root of unity, then

1. so is ω^{-1};
2. if $n = pq$, then ω^p is a primitive qth root of unity;
3. for $\ell \in \{1, \ldots, n-1\}$,

$$\sum_{j=0}^{n-1} \omega^{\ell j} = 0.$$

Detailed proof on the blackboard.

Discrete Fourier Transform

Def. $\mathrm{DFT}_\omega : A \in \mathbb{C}[X] \mapsto (A(1), A(\omega), \ldots, A(\omega^{n-1}))$, where ω is a primitive nth root of unity.

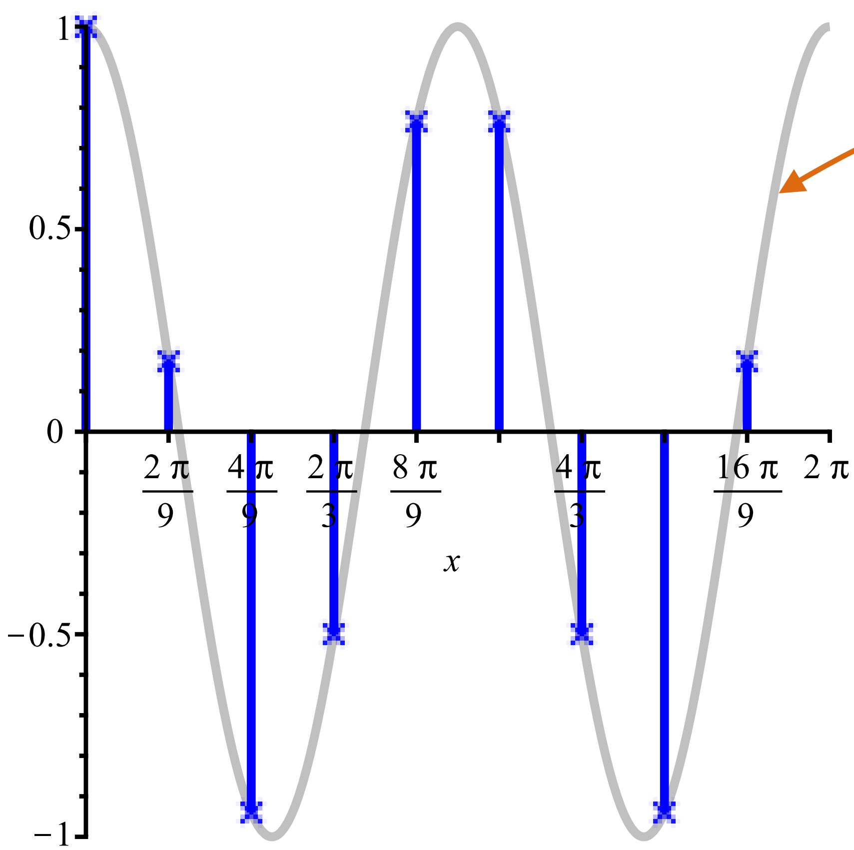
If $\deg A < n$, naive complexity is quadratic.

Extend to discrete signals $(a_0, \ldots, a_{n-1})$, with

$$A(X) = a_0 + a_1 X + \cdots + a_{n-1} X^{n-1}.$$

Important observation: DFT_ω is a linear map.

DFT Recovers Frequencies — Example



$f(x) = \cos(2x)$

Ex. $n = 9, \quad \omega = e^{-2\pi i/9}, \quad \omega^{-1} = \omega^{8}$

Sample points: $\left(0, 2\frac{\pi}{9}, 4\frac{\pi}{9}, \ldots, 16\frac{\pi}{9}\right)$

$$(a_0, \ldots, a_8) = \left(1, \cos(\frac{4\pi}{9}), \ldots, \cos(\frac{32\pi}{9})\right)$$

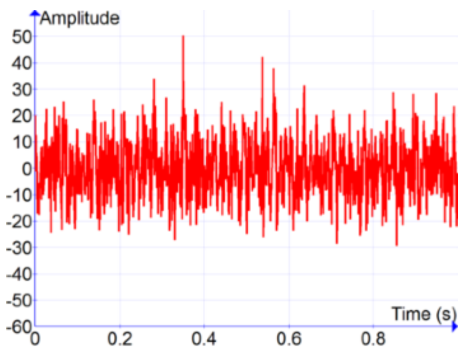
$$= \frac{1}{2}(1, \omega^2, \ldots, \omega^{16}) + \frac{1}{2}(1, \omega^{-2}, \ldots, \omega^{-16})$$

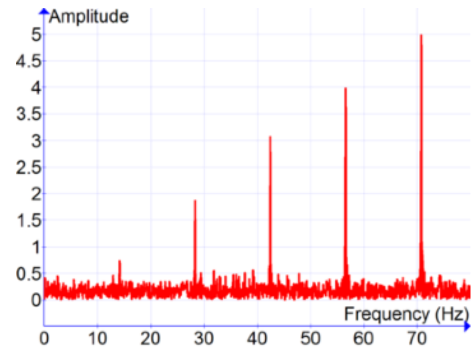
jth entry in $\mathrm{DFT}_\omega(a_0, \ldots, a_8): \quad a_0 + a_1\omega^j + a_2\omega^{2j} + \cdots$

$$= \frac{1 + \omega^{2+j} + \omega^{4+2j} + \cdots}{2} + \frac{1 + \omega^{-2+j} + \omega^{-4+2j} + \cdots}{2} = \begin{cases} n/2 & \text{if } j = 2 \text{ or } n-2, \\ 0 & \text{otherwise.} \end{cases}$$

$$\mathrm{DFT}_\omega(a_0, \ldots, a_8) = 9/2 \times (0,0,1,0,0,0,0,1,0)$$

DFT Recovers Frequencies — General Case





n fixed (large enough), $\omega = e^{-2i\pi/n}$

frequencies

Signal $f(x) = c_1 \cos(k_1 x + \ell_1) + \cdots + c_m \cos(k_m x + \ell_m)$

known by sampling via: $\left(f(0), f(\frac{2\pi}{n}), \ldots, f(\frac{2(n-1)\pi}{n}) \right)$.

By linearity, the entries of its DFT_ω are

$\dfrac{nc_j e^{-i\ell_j}}{2}$ at index k_j, $\quad \dfrac{nc_j e^{i\ell_j}}{2}$ at index $n - k_j$, $\quad j \in \{1, \ldots, m\}$

0 everywhere else.

Blackboard proof

Inverse DFT

How to Recover Signal from DFT

Recall the Vandermonde matrix

$$\begin{pmatrix} P(1) \\ P(\omega) \\ \vdots \\ P(\omega^{n-1}) \end{pmatrix} = \underbrace{\begin{pmatrix} 1 & 1 & \cdots & 1 \\ 1 & \omega & \cdots & \omega^{n-1} \\ & \cdots & \cdots & \\ 1 & \omega^{n-1} & \cdots & \omega^{(n-1)^2} \end{pmatrix}}_{V_\omega} \begin{pmatrix} p_0 \\ p_1 \\ \vdots \\ p_{n-1} \end{pmatrix}$$

Lemma. $V_\omega V_{\omega^{-1}} = n \, \mathrm{Id}_n$.

Proof. The entry (i, j) of the product is $\displaystyle\sum_{k=0}^{n} \omega^{ik} \omega^{-jk}$.

The inverse DFT is computed by one DFT and one division by n .

Convolution

Naive complexity is quadratic.

$$a := (a_0, \ldots, a_{n-1}),\ b := (b_0, \ldots, b_{n-1})$$

$$\mapsto a \star b := (c_0, \ldots, c_{n-1}),\ \text{with } c_\ell = \sum_{i+j=\ell \bmod n} a_i b_j$$

Ex. $(n = 2)$: $(a_0, a_1) \star (b_0, b_1) = (a_0 b_0 + a_1 b_1, a_0 b_1 + a_1 b_0)$

In terms of polynomials:

$$(A(X), B(X)) \mapsto A(X)B(X) \bmod X^n - 1$$

Applications:
. filter a signal;
. **multiply polynomials.**

transforms by DFT into termwise product:

$$a \star b \xrightarrow{\mathrm{DFT}_\omega} \left(A(1)B(1), A(\omega)B(\omega), \ldots, A(\omega^{n-1})B(\omega^{n-1})\right)$$

Convolution is computed by 3 DFT's and O(n) operations.

III. Fast Fourier Transform

Cooley & Tuckey, 1965

Gauss, 1805 (unpublished till 1866 and then unnoticed)

Divide & Conquer

Lemma. $n = 2k$, ω primitive nth root of 1 $\Longrightarrow \omega^k = -1$.

By Euclidean division, A(X) can be written in two ways:

$$A(X) = Q_e(X)(X^k - 1) + R_e(X), \quad A(X) = Q_o(X)(X^k + 1) + R_o(X)\,.$$

Then, $A(\omega^\ell) = \begin{cases} R_e(\omega^\ell), & \text{if } \ell \text{ is even,} \\ R_o(\omega^\ell), & \text{otherwise.} \end{cases}$

Evaluating A of degree $n - 1$ at $1, \omega, \ldots, \omega^{n-1}$ splits into evaluating

$$\left.\begin{matrix} R_e(X) \\ R_o(\omega X) \end{matrix}\right\} \text{ of degree } k - 1 \text{ at } 1, \omega^2, \ldots, \omega^{2(k-1)}\,.$$

Fast Euclidean Division for Very Special Polynomials

$$
\begin{aligned}
A(X) &= a_0 + a_1 X + \cdots + a_{n-1} X^{n-1}, \\
&= \underbrace{(a_0 + \cdots + a_{k-1} X^{k-1})}_{Q_\ell} + X^k \underbrace{(a_k + \cdots + a_{n-1} X^{k-1})}_{Q_h} \\
&= (Q_\ell + Q_h) + (X^k - 1) Q_h = R_e + (X^k - 1) Q_h, \\
&= \underbrace{(Q_\ell - Q_h)} + (X^k + 1) Q_h = R_o + (X^k + 1) Q_h .
\end{aligned}
$$

only k operations

FFT Algorithm

Input. $A = a_0 + \cdots + a_{n-1}X^{n-1}; \quad (1, \omega, \ldots, \omega^{n-1})$
with ω primitive n th root of 1, n power of 2.

Output. $\mathrm{DFT}_\omega(A) = \big(A(1), \ldots, A(\omega^{n-1})\big)$

1. If $n = 1$, return a_0
2. Set $k := n/2$ and compute
$$R_e(X) = \sum_{j=0}^{k-1}(a_j + a_{j+k})X^j, \quad S_o(X) := R_o(\omega X) = \sum_{j=0}^{k-1}(a_j - a_{j+k})\omega^j X^j .$$
3. Compute recursively $\mathrm{DFT}_{\omega^2}(R_e), \mathrm{DFT}_{\omega^2}(S_o)$
4. Return $\big(R_e(1), S_o(1), R_e(\omega^2), \ldots, S_o(\omega^{2(k-1)})\big)$

Easily rewritten without polynomials.

Correctness: should be clear

Complexity

$$C(n) \leq 2C(n/2) + \frac{3n}{2} \text{ operations on the coefficients}$$

iterate once

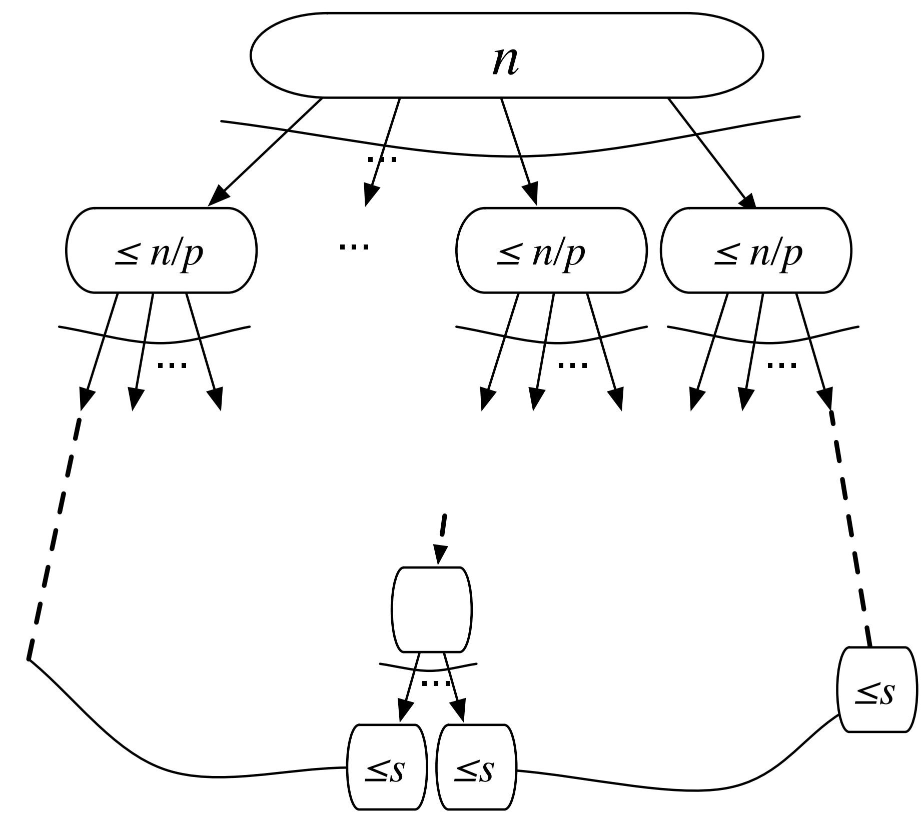
$$\leq \frac{3n}{2} + 2\frac{3n}{4} + 4C(n/4)$$

iterate k-1 times

$$\leq \frac{3n}{2}k + 2^k C(n/2^k)$$

use $k = \log_2 n$

$$\leq \frac{3}{2} n \log_2 n + O(n)$$



All levels of the recursion contribute equally.

IV. Applications

Convolution

Input. $a := (a_0, \ldots, a_{n-1})$, $b := (b_0, \ldots, b_{n-1})$, $(1, \omega, \ldots, \omega^{n-1})$ with ω primitive n th root of 1, n power of 2.

Output. $a \star b = (c_0, \ldots, c_{n-1})$, with $c_\ell = \sum_{i+j=\ell \bmod n} a_i b_j$

1. Compute

$$(\hat{a}_0, \ldots, \hat{a}_{n-1}) = \mathrm{DFT}_\omega(a), \quad (\hat{b}_0, \ldots, \hat{b}_{n-1}) = \mathrm{DFT}_\omega(b)$$

2. Multiply: $\hat{c} := (\hat{a}_0 \hat{b}_0, \ldots, \hat{a}_{n-1} \hat{b}_{n-1})$

3. Return $\frac{1}{n} \mathrm{DFT}_{\omega^{-1}}(\hat{c})$

Complexity: $3\,\mathrm{DFTs} + O(n) = O(n \log n)$.

Multiplication of Polynomials

Input. P and Q with $\deg P + \deg Q < n$,
ω primitive nth root of 1, with n a power of 2 .

Output. $P \times Q$

1. Compute $\omega^2, \ldots, \omega^{n-1}$.
2. Let $a = (p_0, \ldots, p_{\deg P}, 0, \ldots, 0)$, $b = (q_0, \ldots, q_{\deg Q}, 0, \ldots, 0)$
3. Return $a \star b$.

Complexity: 3 DFTs + $O(n) = O(n \log n)$.

Extensions

1. Polynomials over rings $\neq \mathbb{C}$, provided primitive nth roots are available: $O(n \log n)$ operations in the ring.

2. When such primitive roots are not available, one can create them, leading to $O(n \log n \log \log n)$ operations in the ring.

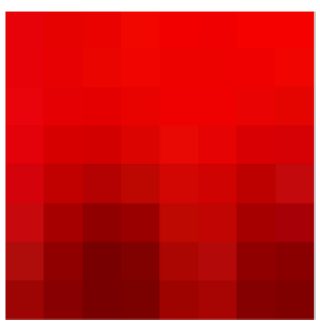
3. Product of integers in $O(n \log n \log \log n)$ bit operations.

New (Jan.19): Product of integers in $O(n \log n)$ bit operations.

Rough Sketch of Jpeg Compression

stored as
3arrays of
8 bit integers

Each of them is split in sub-windows of 8x8 pixels



that are 2D DFTed:

$$F(X,Y)=\sum f_{i,j}X^iY^j$$

$$\mapsto (F(\omega^k,\omega^\ell))_{0\le k,\ell<8}$$

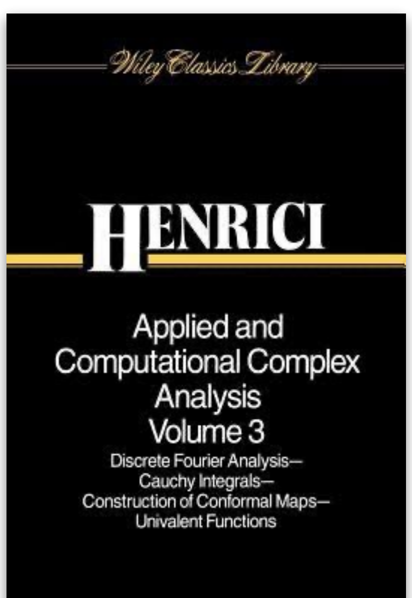
then compressed by

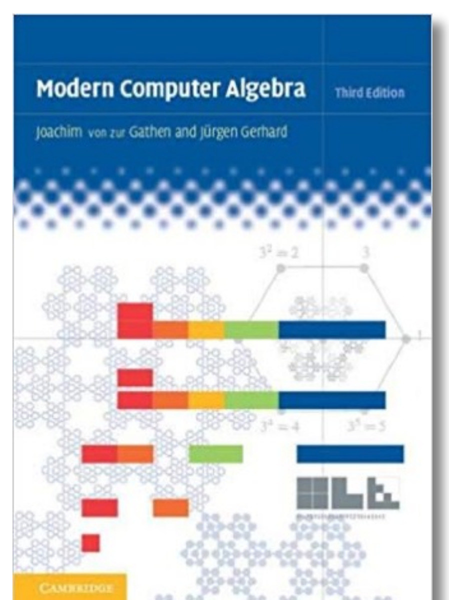
1. rounding, with lower accuracy for high frequencies;
2. storing only the differences between neighbouring F(1,1);
3. Huffman coding (later in the course).

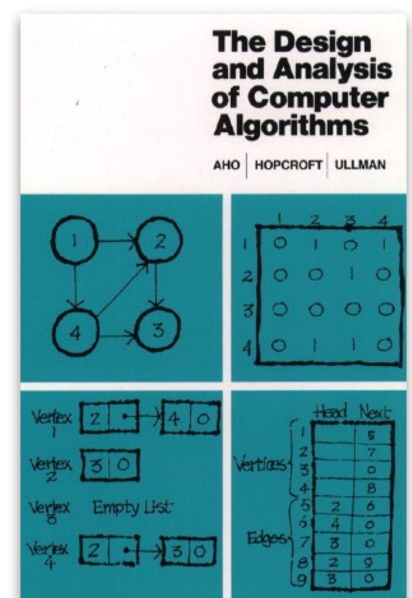
References for this lecture

The slides are designed to be self-contained.

They were prepared using the following
books that I recommend if you want to learn more:







Next

Assignment this week: another way of computing the FFT

Next Thursday: applications of the FFT to sound

Next week: DAC for division and other operations

Feedback

Moodle for the slides, TDs and exercises.

Questions or comments: Bruno.Salvy@inria.fr