We've done

- Introduction to divide and conquer paradigm
  - Quick Sort
  - Selection in linear time
  - Integer multiplication
  - Matrix multiplication

Now

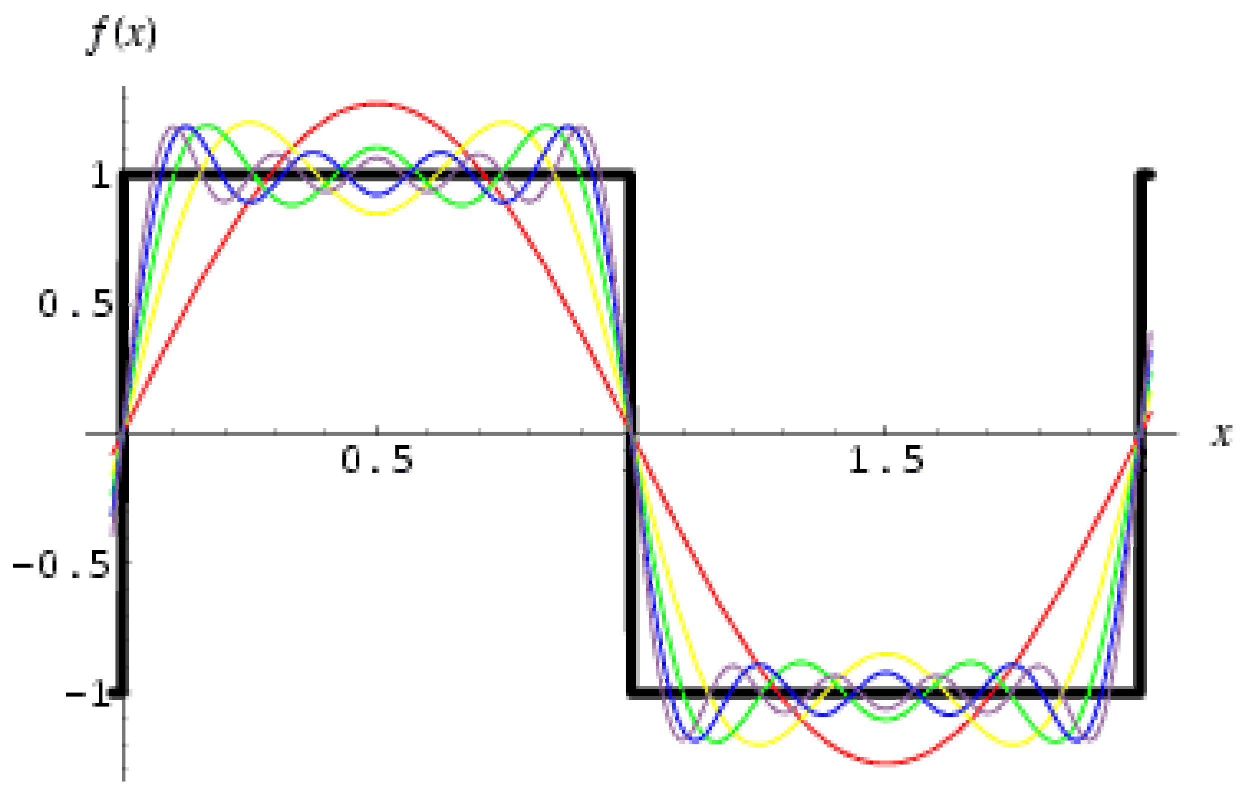
- "Fast Fourier Transform" using divide and conquer
  - Featuring polynomial multiplication as an application

Next

- Greedy Method

Fourier Transforms

- Roughly, Fourier Transforms allow us to look at a function in two different ways
- In (analog and digital) communication theory:
  - time domain $\xrightarrow{FT}$ frequency domain
  - time domain $\xleftarrow{FT^{-1}}$ frequency domain
  - For instance: every (well-behaved) periodic signal (waveform) can be written as a sum of sine and cosine waves (sinusoids), whose frequencies are multiples of a fundamental frequency



Fourier Series of periodic functions

The **sine-cosine representation** of $x(t)$ of period T:

$$x(t) = \frac{1}{2}a_0 + \sum_{n=1}^{\infty} a_n \cos(2\pi n f_0 t) + \sum_{n=1}^{\infty} b_n \sin(2\pi n f_0 t)$$

- $f_0 = 1/T$ is the **fundamental frequency**.
- Multiples of f_0 are **harmonics**.

Euler's formulas:

$$\begin{aligned}
\frac{a_0}{2} &= f_0 \int_{t_0}^{t_0+T} x(t)dt \\
\frac{a_n}{2} &= f_0 \int_{t_0}^{t_0+T} x(t)\cos(2\pi n f_0 t)dt \\
\frac{b_n}{2} &= f_0 \int_{t_0}^{t_0+T} x(t)\sin(2\pi n f_0 t)dx
\end{aligned}$$

[Problem: find a natural science without an Euler's formula]

The **amplitude-phase representation**:

$$x(t) = \frac{c_0}{2} \sum_{n=1}^{\infty} c_n \cos(2\pi n f_0 t + \theta_n)$$

Continuous Fourier Transforms of aperiodic signals

- Basically, just a limit case of Fourier series when $T \to \infty$
- Applications are numerous: digital signal processing, digital image processing, astronomical data analysis, seismic, optics, acoustics, etc.
- **Forward Fourier transform**

$$F(\nu) = \int_{-\infty}^{\infty} f(t) e^{-2\pi i \nu t} dt.$$

- **Inverse Fourier transform**

$$f(t) = \int_{-\infty}^{\infty} F(\nu) e^{2\pi i t \nu} d\nu.$$

(Physicists like to use the *angular frequency* $\omega = 2\pi\nu$)

Discrete Fourier Transforms

- Computers can't handle continuous signals ⇒ discretize it
- Sampling at n places:

$$f_k = f(t_k), \ \ t_k = k\Delta, \ \ k = 0, \dots, n-1$$

- DFT: (when going from continuous to discrete, integral becomes sum)

$$F_m = \sum_{k=0}^{n-1} f_k (e^{-2\pi i m/n})^k, \ 0 \leq m \leq n-1$$

- DFT^{-1}:

$$f_k = \frac{1}{n} \sum_{m=0}^{n-1} F_m (e^{2\pi i k/n})^m, \ 0 \leq k \leq n-1$$

Fundamental problem: compute DFT and DFT^{-1} efficiently

Polynomials

- A polynomial $A(x)$ over the complex numbers $\mathbb{C}$:

  $$A(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_{n-1} x^{n-1} = \sum_{j=0}^{n-1} a_j x^j.$$

  (*over* $\mathbb{C}$ means $x \in \mathbb{C}$).

- $a_0, \dots, a_{n-1}$ are the coefficients of A.

- $A(x)$ is of degree k if a_k is the highest non-zero coefficient. For instance,

  $$B(x) = 3 - (2 - 4i)x + x^2 \text{ has degree 2.}$$

- An integer m strictly greater than the degree is called a degree bound of the polynomial. For instance, $B(x)$ above has degree bounds $3, 4, \dots$

- In the generic form of $A(x)$ given above, n is a degree bound of $A(x)$.

Common operations on polynomials

Given two polynomials

$$\begin{aligned} A(x) &= a_0 + a_1 x + \cdots + a_{n-1}x^{n-1} \\ B(x) &= b_0 + b_1 x + \cdots + b_{n-1}x^{n-1} \end{aligned}$$

Addition

$$\begin{aligned} C(x) &= A(x) + B(x) \\ &= (a_0 + b_0) + (a_1 + b_1)x + \cdots + (a_{n-1} + b_{n-1})x^{n-1} \end{aligned}$$

Multiplication

$$\begin{aligned} C(x) &= A(x)B(x) \\ &= c_0 + c_1 x + \cdots + c_{2n-2}x^{2n-2} \end{aligned}$$

where, for $0 \leq k \leq 2n - 2$

$$c_j = \sum_{j=0}^{k} a_j b_{k-j}$$

Efficiently computing sums and products of polynomials is a very important problem in scientific computing!

Polynomial representations

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

Coefficient representation: a vector a

$$a = (a_0, a_1, \dots, a_{n-1})$$

Point-value representation: a set of point-value pairs

$$\{(x_0, y_0), (x_1, y_1), \dots, (x_{n-1}, y_{n-1})\}$$

where the x_j are distinct, and $y_j = A(x_j), \forall j$

Question: how do we know that a set of point-value pairs represent a unique polynomial? What if there are two polynomials with the same set of point-value pairs?

Uniqueness of point-value representation

Theorem 1. *For any set* $\{(x_0, y_0), \dots, (x_{n-1}, y_{n-1})\}$ *where the* x_j *are distinct, there is a unique polynomial* $A(x)$ *of degree bound* n *such that* $A(x_j) = y_j, \forall j = 0, \dots, n-1$.

(The operation of finding the coefficients from the point-value pairs is called polynomial interpolation)

Proof. We solve a system of n linear equations for n unknowns

$$\begin{bmatrix} 1 & x_0 & x_0^2 & \dots & x_0^{n-1} \\ 1 & x_1 & x_1^2 & \dots & x_1^{n-1} \\ \vdots & \vdots & \vdots & \dots & \vdots \\ 1 & x_{n-1} & x_{n-1}^2 & \dots & x_{n-1}^{n-1} \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_{n-1} \end{bmatrix} = \begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_{n-1} \end{bmatrix}$$

The matrix is called the **Vandermonde matrix** $V(x_0, \dots, x_{n-1})$, which has non-zero determinant

$$\det(V(x_0, \dots, x_{n-1})) = \prod_{p<q} (x_p - x_q).$$

□

Solving the interpolation problem

- Gaussian elimination helps solve the interpolation problem (via the system of linear equations) in $O(n^3)$ time.
- Lagrange's formula helps solve it in $\Theta(n^2)$ time:

$$A(x) = \sum_{k=0}^{n-1} y_k \frac{\prod_{j\neq k}(x - x_j)}{\prod_{j\neq k}(x_k - x_j)}$$

  (how to get $\Theta(n^2)$ is a homework problem!)

- Fast Fourier Transform (FFT) helps perform the inverse DFT operation (another way to express interpolation) in $\Theta(n \lg n)$-time.

Solving the evaluation problem

The evaluation problem: Given $A(x)$ in coefficient representation, compute $A(x_0), \dots, A(x_{n-1})$

- Horner's rule gives $\Theta(n^2)$
- Again FFT helps perform the DFT operation in $\Theta(n \lg n)$-time

Pros and cons

Coefficient representation:

- computing the sum $A(x) + B(x)$ takes $\Theta(n)$,
- evaluating $A(x_k)$ take $\Theta(n)$ with **Horner's rule**

  $$A(x_k) = a_0 + x_k(a_1 + x_k(a_2 + \cdots + x_k(a_{n-2} + x_k a_{n-1}) \dots)$$

  (we assume $+$ and $*$ of numbers take constant time)
- very convenient for user interaction
- computing the product $A(x)B(x)$ takes $\Theta(n^2)$, however

Point-value representation:

- computing the sum $A(x) + B(x)$ takes $\Theta(n)$,
- computing the product $A(x)B(x)$ takes $\Theta(n)$ (need to have $2n$ points from each of A and B though)
- inconvenient for user interaction

> Problem: how can we compute products in coefficient representation in time better than $\Theta(n^2)$?

Efficient polynomial product in coefficient form

Input: $A(x), B(x)$ of degree bound n in coefficient form
Output: $C(x) = A(x)B(x)$ of degree bound $2n - 1$ in coefficient form

1. *Double degree bound*: extend $A(x)$'s and $B(x)$'s coefficient representations to be of degree bound $2n$ $[\Theta(n)]$
2. *Evaluate*: compute point-value representations of $A(x)$ and $B(x)$ at each of the $2n$th roots of unity (with FFT of order $2n$) $[\Theta(n \lg n)]$
3. *Pointwise multiply*: compute point-value representation of $C(x) = A(x)B(x)$ $[\Theta(n)]$
4. *Interpolate*: compute coefficient representation of $C(x)$ (with FFT or order $2n$) $[\Theta(n \lg n)]$

Complex numbers, complex roots of unity

- $\mathbb{C} = \{a + bi \mid a, b \in \mathbb{R}\}$
- $w \in \mathbb{C}$, $w^n = 1$, then w is a complex nth root of unity
- There are n of them: $e^{2\pi ik/n}$, $k = 0, \dots, n-1$
- $e^{iu} = \cos(u) + i\sin(u)$
- $w_n = e^{2\pi i/n}$ is the principal nth root of unity
- all nth roots are of the form w_n^k, $k = 0, \dots, n-1$
- $1 = w_n^0, w_n^1, w_n^2, \dots, w_n^{n-1}, w_n^n = w_n^0 = 1, w_n^{n+1} = w_n, w_n^{n+2} = w_n^2, \dots$
- In general, $w_n^j = w_n^{j \bmod n}$.

Lemma 2 (Cancellation lemma). *For any integers $n \geq 0, k \geq 0$, and $d > 0$, then $w_{dn}^{dk} = w_n^k$.*

Corollary 3. $w_{2m}^m = w_2 = -1$.

Lemma 4 (Summation lemma). *Given $n \geq 1$, k not divisible by n, then $\sum_{j=0}^{n-1}(w_n^k)^j = 0$.*

Discrete Fourier Transform (DFT)

Given $A(x) = \sum_{j=0}^{n-1} a_j x^j$, let $y_k = A(w_n^k)$, then the vector

$$y = (y_0, y_1, \dots, y_{n-1})$$

is the Discrete Fourier Transform (**DFT**) of the coefficient vector $a = (a_0, a_1, \dots, a_{n-1})$. We write

$$y = \mathrm{DFT}_n(a).$$

Fast Fourier Transform (FFT)

is an efficient **algorithm** to compute DFT (a transformation)

Idea: suppose $n = 2m$

$$\begin{aligned}
A(x) &= a_0 + a_1 x + a_2 x + \cdots + a_{2m-1} x^{2m-1} \\
&= a_0 + a_2 x^2 + a_4 x^4 + \cdots + a_{2m-2} x^{2m-2} + \\
&\quad x(a_1 + a_3 x^2 + a_5 x^4 + \cdots + a_{2m-1} x^{2m-2}) \\
&= A^{[0]}(x^2) + x A^{[1]}(x^2),
\end{aligned}$$

where

$$\begin{aligned}
A^{[0]}(x) &= a_0 + a_2 x + a_4 x^2 + \cdots + a_{2m-2} x^{m-1} \\
A^{[1]}(x) &= a_1 + a_3 x + a_5 x^2 + \cdots + a_{2m-1} x^{m-1}
\end{aligned}$$

FFT (continue)

By the cancellation lemma,

$$\begin{aligned}
(w_{2m}^0)^2 &= w_m^0 \\
(w_{2m}^1)^2 &= w_m^1 \\
&\vdots \\
(w_{2m}^{m-1})^2 &= w_m^{m-1}
\end{aligned}$$

$$\begin{aligned}
(w_{2m}^m)^2 &= w_m^0 \\
(w_{2m}^{m+1})^2 &= w_m^1 \\
&\vdots \\
(w_{2m}^{2m-1})^2 &= w_m^{m-1}
\end{aligned}$$

we get two smaller evaluation problems for $A^{[0]}(x)$ and $A^{[1]}(x)$:

$$\begin{aligned}
A(w_{2m}^j) &= A^{[0]}((w_{2m}^j)^2) + w_{2m}^j A^{[1]}((w_{2m}^j)^2) \\
&= A^{[0]}(w_m^j) + w_{2m}^j A^{[1]}(w_m^j) \\
&= A^{[0]}(w_m^{j \bmod m}) + w_{2m}^j A^{[1]}(w_m^{j \bmod m})
\end{aligned}$$

FFT (continue)

$$a = (a_0, a_1, \dots, a_{2m-1}), \quad y = \text{DFT}_{2m}(a)$$

$$\begin{aligned} a^{[0]} &= (a_0, a_2, \dots, a_{2m-2}) \\ a^{[1]} &= (a_1, a_3, \dots, a_{2m-1}) \\ y^{[0]} &= \text{DFT}_m(a^{[0]}) \\ y^{[1]} &= \text{DFT}_m(a^{[1]}) \end{aligned}$$

Then, y can be computed from $y^{[0]}$ and $y^{[1]}$ as follows.

For $0 \leq j \leq m-1$:

$$\begin{aligned} y_j &= A(w_{2m}^j) = A^{[0]}(w_m^j) + w_{2m}^j A^{[1]}(w_m^j) \\ &= y_j^{[0]} + w_{2m}^j y_j^{[1]}. \end{aligned}$$

For $m \leq j \leq 2m-1$:

$$\begin{aligned} y_j &= A(w_{2m}^j) = A^{[0]}(w_m^{j-m}) + w_{2m}^j A^{[1]}(w_m^{j-m}) \\ &= y_{j-m}^{[0]} + w_{2m}^j y_{j-m}^{[1]} = y_{j-m}^{[0]} - w_{2m}^{j-m} y_{j-m}^{[1]}. \end{aligned}$$

FFT – pseudo code

RECURSIVE-FFT(a)

$n \leftarrow \text{length}(a)$ // n is a power of 2
if $n = 1$ **then**
  return a
end if
$w_n \leftarrow e^{2\pi i/n}$ // principal nth root of unity
$a^{[0]} \leftarrow (a_0, a_2, \dots, a_{n-2})$
$a^{[1]} \leftarrow (a_1, a_3, \dots, a_{n-1})$
$y^{[0]} \leftarrow \text{RECURSIVE-FFT}(a^{[0]})$
$y^{[1]} \leftarrow \text{RECURSIVE-FFT}(a^{[1]})$
$w \leftarrow 1$ really meant $w \leftarrow w_n^0$
for $k \leftarrow 0$ to $n/2 - 1$ **do**
  $y_k \leftarrow y_k^{[0]} + w y_k^{[1]}$
  $y_{k+n/2} \leftarrow y_k^{[0]} - w y_k^{[1]}$
  $w \leftarrow w w_n$
end for
return y

$$T(n) = 2T(n/2) + \Theta(n) \Rightarrow T(n) = \Theta(n \lg n)$$

Inverse DFT – Interpolation at the roots

Now that we know y, how to compute $a = \text{DFT}_n^{-1}(y)$?

$$\begin{bmatrix} 1 & w_n & w_n^2 & \dots & w_n^{n-1} \\ 1 & w_n^2 & w_n^4 & \dots & w_n^{2(n-1)} \\ \vdots & \vdots & \vdots & \dots & \vdots \\ 1 & w_n^{n-1} & w_n^{2(n-1)} & \dots & w_n^{(n-1)(n-1)} \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_{n-1} \end{bmatrix} = \begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_{n-1} \end{bmatrix}$$

Need the inverse V_n^{-1} of $V_n := V(1, w_n, w_n^2 \dots, w_n^{n-1})$

Theorem 5. *For* $0 \leq j, k \leq n-1$,

$$[V_n^{-1}]_{j,k} = \frac{w_n^{-kj}}{n}.$$

Thus,

$$a_j = \sum_{k=0}^{n-1} [V_n^{-1}]_{j,k} y_k = \sum_{k=0}^{n-1} \frac{w_n^{-kj}}{n} y_k = \sum_{k=0}^{n-1} \frac{y_k}{n} (w_n^{-j})^k$$

$$a_j = Y(w_n^{-j}), \quad Y(x) = \frac{y_0}{n} + \frac{y_1}{n} x + \dots + \frac{y_{n-1}}{n} x^{n-1}$$

We can easily modify the pseudo code for FFT to compute a from y in $\Theta(n \lg n)$-time (homework 3!)