LECTURE 4
9/4/18
ch 2.6
FAST FOURIER TRANSFORM

- RECALL
  - -> integer multiplication
    $O(n^2) \to O(n^{\log_2 3})$
  - -> matrix multiplication
    $O(n^3) \to O(n^{\log_2 7})$

  > by saving one multiplication

- POLYNOMIAL MULTIPLICATION (EVALUATION)
  - => example of polynomial multiplication
    $(1+2x+3x^2)\cdot(4+3x+2x^2)$
    - -> expect polynomial of degree 4
      - we have to find list of coefficients,

x^0	$1\cdot 4$	1 way
x^1	$1\cdot 3+2\cdot 4$	2
x^2	$3\cdot 4+2\cdot 1+2\cdot 3$	3
x^3	⋮	
x^4	⋮	

  - => in general
    input: list: $a_0, a_1, \dots, a_d$ $(A(x) = a_0 + a_1 x + \dots + a_d x^d)$
    list: $b_0, b_1, \dots, b_d$
    output: list: $c_0, c_1, \dots, c_{2d}$ $(P(x) = A(x)\cdot B(x))$

    -> STRAIGHTFORWARD way
    for $i = 0, \dots, 2d$:

$$c_i = \sum_{k=0}^{i} a_k b_{i-k}$$

-> all possible combinations to get power i

    -> runtime
    $1 + 2 + 3 + \dots + d + \dots + 2 + 1$
    ↳ first sum, third sum = problem of size 3

    $= O(d^2)$

- suppose I want to know $C(1)$
  - → compute $A(1)$
  - → compute $B(1)$
  - → multiply $A(1) \cdot B(1)$
- suppose I want $C(1), \dots, C(m)$
  - → compute $A(1), \dots, A(m)$
  - → compute $B(1), \dots, B(m)$
  - → multiply $A(1) \cdot B(1), \dots, A(m) \cdot B(m)$

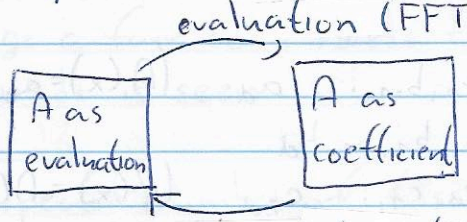
⇒ if you have bunch of evaluations of A and B, you can in linear time derive from them evaluation of C

⇒ **approach to find C**

- → evaluate A: $A(1), A(2), \dots$ (evaluate it at many points)
- → evaluate B: $B(1), B(2), \dots$ → $Eval(A,S), Eval(B,S)$
- → pointwise multiply $C(1), C(2), \dots$ $O(|S|)$
- → interpolate to find C

S ... set of evaluation points

$|S| \geq 2d+1$



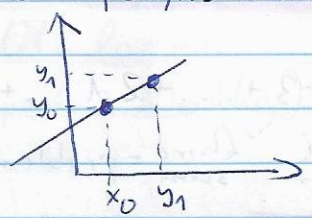
↳ inverse Fast Fourier Transform)

polynomial can be characterized
- → by coefficients $a_0, \dots, a_d$
- → by values $A(x_0), \dots, A(x_d)$

- → example
  - line (degree 1 polynomials)

    $a_0 + a_1 x$

    2 points to identify the line



- → example
  - parabola (degree 2 polynomial)

    $a_0 + a_1 x + a_2 x^2$

    - need 3 distinct points

=> in general
- degree d polynomial -> need (d+1) evaluation point
-> $|S|$ should be at least $(2d+1$ $|S| \geq 2d+1$
-> we need to evaluate A and B at $(2d+1)$ points

• POLYNOMIAL INPUT
- input: $a_0, a_1, \dots, a_d$ -> coefficients
- output: $A(x_0), A(x_1), \dots A(x_d)$ -> we pick xs

=> straightforward way
$A(1), \dots, A(d+1)$

for $i = 1, \dots, d+1$: $\times d$
compute $A(i)$ -> by Horner's Rule $O(d)$
- runtime $O(d^2)$

• FAST FOURIER TRANSFORM
-> group together odd and even terms

$$A(x) = A_e(x^2) + xA_o(x^2)$$

-> example

$$A(x) = 4 + 12x + 20x^2 + 13x^3 + 6x^4 + 7x^5$$
$$= \underbrace{(4 + 20x^2 + 6x^4)}_{A_e(x)} + x\underbrace{(12 + 13x^2 + 7x^4)}_{A_o(x^2)}$$

- want $A(1)$ and $A(-1)$, $A(i)$ and $A(-i)$

need $A_e(1), A_e(-1), A_o(1), A_o(-1)$

$A(1) = A_e(1) + 1 \cdot A(1)$
$A(-1) = A_e(1) - 1 \cdot A(1)$ } need $A_e(1), A_o(1)$
$A(i) = A_e(-1) + i \cdot A_o(-1)$
$A(-i) = A_e(-1) - i \cdot A_o(-1)$

⇒ generalize

- start with S of size $d+1 = n$

$|S| = |\{\alpha_0, \ldots, \alpha_{n-1}\}| = n$

$|\{\alpha_0^2, \ldots, \alpha_{n-1}^2\}| = \frac{n}{2}$

some elements in set have the same values

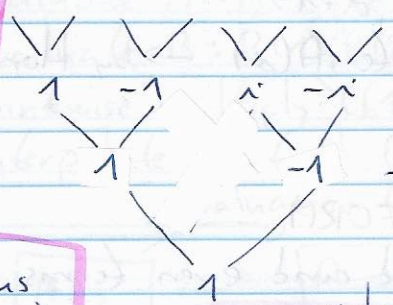
$|\{\alpha_0^4, \ldots, \alpha_{n-1}^4\}| = \frac{n}{4}$

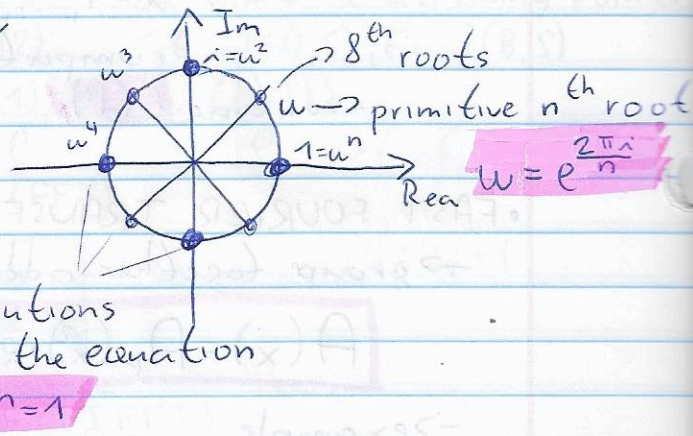
$\vdots$

$\log n$ steps

$|\{\alpha_0^n, \ldots, \alpha_{n-1}^n\}| = n$

- we want to generalize this pattern



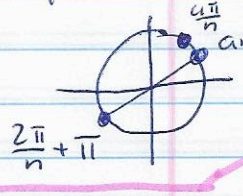


n^{th} complex roots of unity

$z^n = 1$

→ solutions are $z = (1, \theta)$ for multiple of $\frac{2\pi}{n}$

- for even n
  - numbers are plus-minus paired $-(1, \theta) = (1, \theta + \pi)$



- we want n to be power of 2
  - because we want a nice structure in the unit circle
  - we can always pad $d+1$ to next power of 2
  - → n^{th} roots are plus-minus pairs: $w^{\frac{n}{2}+j} = -w^j$
  - → squaring them produces $(\frac{n}{2})^{nd}$ roots of unity
- if we start with these numbers for some n that is power of 2, then at successive levels of recursion, we will have $(\frac{n}{2^k})^{th}$ root of unity
  - → all these sets are numbers plus-minus paired, so divide and conquer works perfectly

- THE FAST FOURIER TRANSFORM ALGORITHM

-input: $a_0, a_1, \dots, a_{n-1}, w \in \mathbb{C}$

-output: $A(w^0), A(w^1), \dots, A(w^{n-1})$

$A_e(w^{2\cdot 0}), A(w^{2\cdot 1}), \dots, A_e(w^{2\cdot(\frac{n}{2})})$ $\leftarrow FFT(A_e, w^2)$ ($\frac{n}{2}^{th}$ root)

$A_o(w^{2\cdot 0}), A_o(w^{2\cdot 1}), \dots, A_o(w^{2\cdot \frac{n}{2}})$ $\leftarrow FFT(A_o, w^2)$

for $i = 0, 1, \dots, n-1$

$A(w^i) = A_e(w^{2i}) + w^i A_o(w^{2i})$

or

for $i = 0, 1, \dots, \frac{n}{2} - 1$

$A(w^i) = A_e(w^{2i}) + w^i A_o(w^{2i})$

$A(w^{i+\frac{n}{2}}) = A_e(w^{2i}) + w^{i+\frac{n}{2}} A_o(w^{2i})$

$\Rightarrow$ runtime

$$T(n) = 2T(\tfrac{n}{2}) + O(n)$$

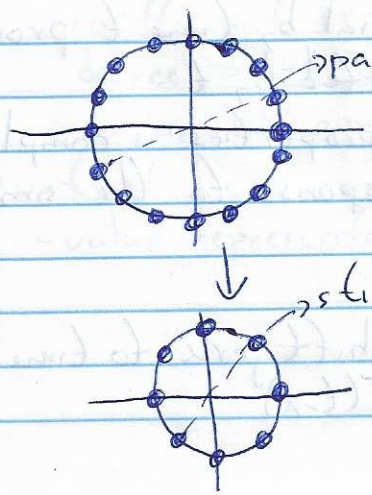
- $2T(\frac{n}{2})$: 2 recursive calls; evaluate polynomial of half of size in domain of half a size
- $O(n)$: computing linear combinations

$\rightarrow$ both degree and domain halved
- degree halved because you look just at even or odd power
- domain halved because you squared it

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

$\Rightarrow$ divide and conquer step

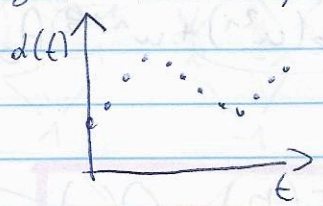
evaluate $A(x)$ at n^{th} roots of unity



Evaluate $A_e(x)$ and $A_o(x)$ at $(\frac{n}{2})^{nd}$ roots

• WHY MULTIPLY POLYNOMIALS?

- fastest algorithms we have for multiplying integers rely heavily on polynomial multiplication
  - polynomials and binary integers are quite similar
    - → just replace variable x by base two and watch out for carries
- multiplying polynomials is crucial for signal processing
  - signal = quantity that is function of time or position
    - → examples - capture voice by measuring fluctuation in air pressure close to mouth
  - to extract information from signal
    1. digitize it by sampling

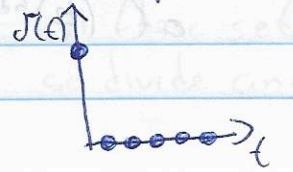


2. to put it through system we will transform it in some way
   - output = response of system

signal → [SYSTEM] → response

- linear systems
  - response to the sum of two signals is just the sum of two individual responses
- time invariant
  - shifting the input signal by time t produces the same output, also shifted by t

⇒ any system with these properties is completely characterized by its response to the simplest possible input signal
= unit impulse $\delta(t)$



→ shift jerk to time x
$\delta(t-x)$

→ any signal $a(t)$ can be expressed as linear combination of these

$$a(t) = \sum_{i=0}^{T-1} a(i) \cdot \delta(t-i)$$ (if signal consists of T samples)

- system response to input $a(t)$ is determined by response to various $\delta(t-i)$
  - → by time variance, these are in turn just shifted copies of impulse response $b(t)$, the response to $\delta(t)$
  - → output of system at time k

$$c(k) = \sum_{i=0}^{k} a(i) \cdot b(k-i)$$

(formula for polynomial multiplication)

- FUNCTION FOR FAST FOURIER TRANSFORM

function FFT(A, w)
  if $w = 1$:
    return $A(1)$
  express $A(x)$ in form $A_e(x^2) + x A_o(x^2)$
  call FFT to evaluate A_e at even powers of w: FFT(A_e, w^2)
  call FFT to evaluate A_o at even powers of w: FFT(A_o, w^2)
  for $j = 0$ to $n-1$:
    compute $A(w^j) = A_e(w^{2j}) + w^j A_o(w^{2j})$
  return $A(w^0), \ldots, A(w^{n-1})$

→ input
  - coefficient representation of polynomial $A(x)$ of degree $\leq n-1$, where n is power of 2
  - w, an n^{th} power of unity

→ output
  - value representation $A(w^0), \ldots, A(w^{n-1})$

- INTERPOLATION
  - from value representation to coefficient representation
    $A(x_0), \ldots, A(x_{n-1})$ $\qquad a_0, \ldots, a_{n-1}$
  - → evaluation
    $\langle values \rangle = FFT(\langle coefficients \rangle, w)$
  - → interpolation
    $\langle coefficients \rangle = \frac{1}{n} FFT(\langle values \rangle, w^{-1})$

⇒ MATRIX REFORMULATION

- relationship between our two representations for polynomial $A(x)$ of degree $\leq n-1$
  - → linear transformation

$$\begin{bmatrix} A(x_0) \\ \vdots \\ A(x_{n-1}) \end{bmatrix} = \begin{bmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^{n-1} \\ 1 & x_1 & x_1^2 & & \\ \vdots & & & & \\ 1 & x_{n-1} & x_{n-1}^2 & \cdots & x_{n-1}^{n-1} \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_{n-1} \end{bmatrix}$$

$=$ M $\qquad$ $=$ coefficient representation

- M - Vandermonde matrix
  - → if $x_0, \ldots, x_{n-1}$ are distinct numbers, then M is invertible
    - existence of M^{-1} allows us to express coefficients in terms of values
- evaluation = multiplication by M
- interpolation = multiplication by M^{-1}
- Vandermonde matrix can be inverted in $O(n^2)$, instead

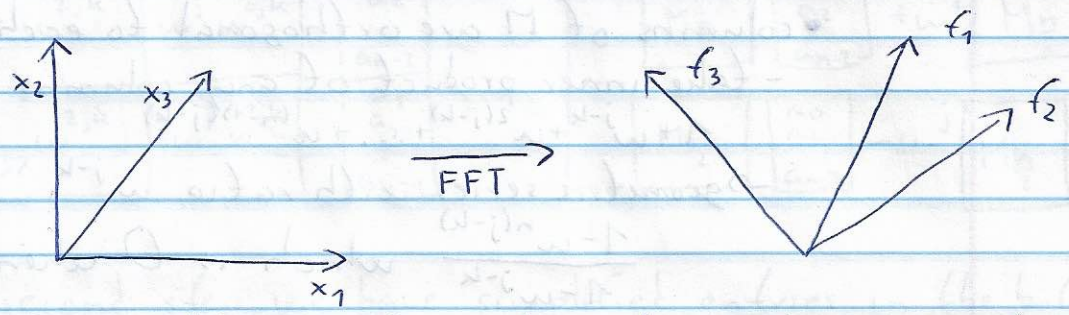
⇒ INTERPOLATION RESOLVED

$$M_n(w) = \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & w & w^2 & & w^{n-1} \\ \vdots & & & & \vdots \\ 1 & w^j & w^{2j} & & w^{(n-1)j} \\ \vdots & & & & \vdots \\ 1 & w^{(n-1)} & w^{2(n-1)} & \cdots & w^{(n-1)(n-1)} \end{bmatrix}$$

← row for $w^0 = 1$; ← w; ← w^j; ← w^{n-1}

- its (j,k) entry is w^{jk}
  - starting row and column count at 0

- note that columns of M are orthogonal to each other



→ FFT takes points in the standard coordinate system. (axes x_1, x_2, x_3) and rotates them into the Fourier basis, whose axis are columns of $M_n(w)$, shown as f_1, f_2, f_3
- for example points in direction x_1 get mapped into direction f_1

- effect of multiplying vector by M
  - rotates it from standard basis into Fourier basis
- Fourier basis = defined by columns of M
- FFT = change of basis (rigid rotation)

- inverse of M = opposite rotation

→ inversion formula

$$M_n(w)^{-1} = \frac{1}{n} M_n(w^{-1})$$

- w^{-1} is n^{th} root of unity
  → interpolation = multiplication by $M_n(w)^{-1}$
  = FFT with w replaced by w^{-1}

- take $w = e^{\frac{2\pi i}{n}}$, think about columns of M as vectors in $\mathbb{C}^n$
- recall angle between vectors $u = (u_0, \ldots, u_{n-1})$ and $v = (v_0, \ldots, v_{n-1})$ in $\mathbb{C}^n$ is just scaling factor times their inner product

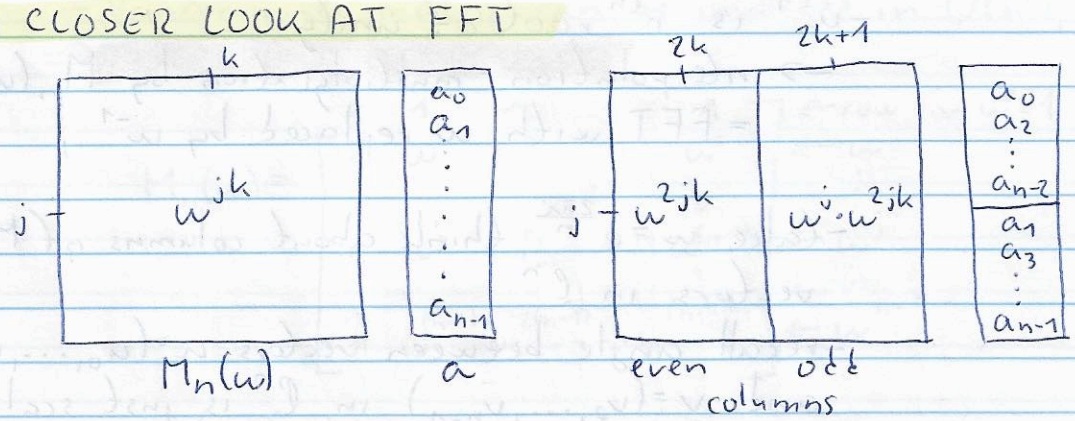
$$u \cdot v^* = u_0 v_0^* + u_1 v_1^* + \ldots + u_{n-1} v_{n-1}^*$$

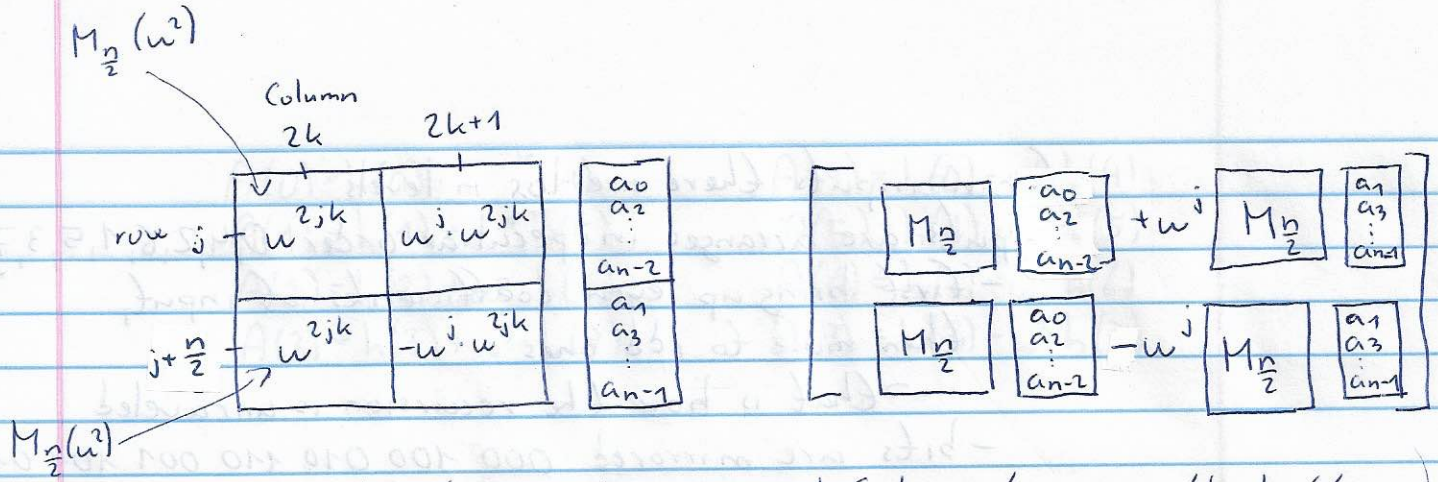
↳ complex conjugate ($re^{i\theta}$ and $re^{-i\theta}$)

→ PROOF

- columns of M are orthogonal to each other
  - take inner product of any columns j and k of M
    $1 + w^{j-k} + w^{2(j-k)} + \ldots + w^{(n-1)(j-k)}$
    → geometric series with ratio w^{j-k}
    $\frac{1 - w^{n(j-k)}}{1 - w^{j-k}}$ which is 0 when $j \neq k$
  - → therefore
    $MM^* = nI$
    - since $(MM^*)_{ij}$ is the inner product of i^{th} and j^{th} columns of M
      → this implies
      $M^{-1} = \frac{1}{n} M^*$ → inversion formula
      - because $M^* = M(w^{-1})$
  - looking at FFT geometrically
    → polynomial multiplication is easier in Fourier basis than in standard basis
    → rotate vectors into Fourier basis = evaluation
    → perform task = multiplication
    → rotate back = interpolation
    [initial vectors are coefficient representations
    [rotated counterparts are value representations
    → to switch between these use FFT

- A CLOSER LOOK AT FFT





- in second step, we have simplified entries in the bottom half of matrix using $w^{\frac{n}{2}} = -1$ and $w^n = 1$

• FFT ALGORITHM FOR MATRIX APPROACH

FFT(a, w):

```
if w = 1:
    return a
(s_0, s_1, ..., s_{n/2-1}) = FFT((a_0, a_2, ..., a_{n-2}), w^2)
(s'_0, s'_1, ..., s'_{n/2-1}) = FFT((a_1, a_3, ..., a_{n-1}), w^2)
for j = 0 to n/2 - 1:
    r_j = s_j + w^j s'_j
    r_{j+n/2} = s_j - w^j s'_j
return (r_0, r_1, ..., r_{n-1})
```

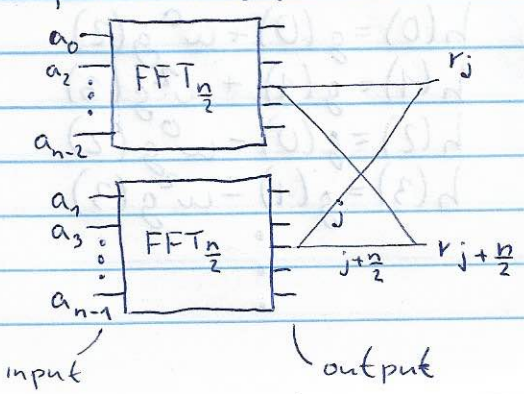
→ input - array $a = (a_0, a_1, \ldots, a_{n-1})$, for n of power of 2
- a primitive n^{th} root of unity, w

→ output - $M_n(w) \cdot a$

• THE FFT UNRAVELED

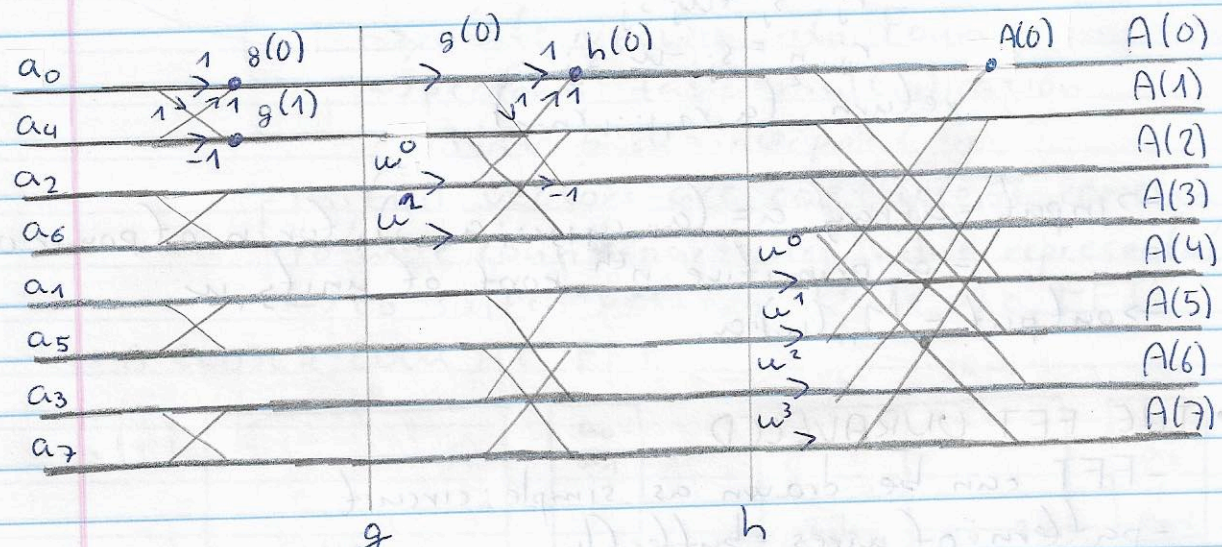
- FFT can be drawn as simple circuit
- pattern of wires = butterfly

$r_j = s_j + w^j s'_j$
$r_{j+\frac{n}{2}} = s_j - w^j s'_j$



- edges carry complex numbers from left to right
- weight of j means: multiply number on this wire by w^j
- when two wires come into junction from left, number they carry get added up

- for n inputs there are $\log n$ levels
- inputs are arranged in peculiar order $0,4,2,6,1,5,3,7$ $\rightarrow n=8$
  - first bring up even coefficients of input, then move to odd ones
    - $\rightarrow$ that is how the recursion is unraveled
  - bits are mirrored $000\ 100\ 010\ 110\ 001\ 101\ 011\ 111$
    $000\ 001\ 010\ 011\ 100\ 101\ 110\ 111$
- there is unique path between each a_j to each output $A(\omega^k)$
- on the path between a_j and $A(\omega^k)$, the labels add up to $(jk \bmod 8)$
  - since $\omega^8=1$, contribution of input a_j to output $A(\omega^k)$ is $a_j\omega^{jk}$
    - $\rightarrow$ circuit computes correctly the values of polynomial $A(x)$
- FFT circuit is natural for parallel computation and direct implementation in hardware



$$g(0) = a_0 + a_4$$
$$g(1) = a_0 - a_4$$
$$g(2) = a_2 + a_6$$
$$\vdots$$

$$h(0) = g(0) + \omega^0 g(2)$$
$$h(1) = g(1) + \omega^2 g(3)$$
$$h(2) = g(0) - \omega^0 g(2)$$
$$h(3) = g(1) - \omega^2 g(3)$$
$$\vdots$$

$A(0) = h(0) + w^0 \cdot h(4)$
$A(1) = h(1) + w^1 h(5)$
$A(2) = h(2) + w^2 \cdot h(6)$
$A(3) = h(3) + w^3 \cdot h(7)$

$A(4) = h(0) - w^0 h(4)$
$A(5) = h(1) - w^1 h(5)$
$A(6) = h(2) - w^2 h(6)$
$A(7) = h(3) - w^3 h(7)$