Chapter -9-

~~Fast Fourier Transform (FFT)~~

(Computation of DFT)

* FFT alogrithms were a major landmark
  - performance ~~improved~~ improvment in ~~computation~~.

$$(DFT)\quad X(k) = \sum_{n=0}^{N-1} x(n)\, W_N^{kn}$$

$$(\cancel{IDFT})\quad \cancel{DC(w)} = \frac{1}{N} \sum_{k=0}^{N-1} X(k)\, W_N^{-nk}$$

computation is mainly the same.

$$W_N = e^{-j\frac{2\pi}{N}}$$

so,

* all methods used for computing DFT are also used for Complig of IDFT.

Direct Computation:

general case is complex.

$$x(n)\, W_N^{kn} \Rightarrow 1 \text{ Complex Multiply } (4 \underset{\text{real}}{\text{ Mults}}, 2 \underset{\text{(real)}}{\text{ adds}})$$

* to compute DFT for one value of (k), we need N Mults, and (N-1) adds.

$X(k)$, $k = 0, 1, 2, \ldots, N-1$

N^2 complex Mults.

$N(N-1)$ complex adds

$\approx N^2$ MADS

dependancy on N^2

Fast DFT algorithms take advantage of symmetry of W_N^{kn} to reduce the Computation to half N^2 or to quarter N^2.

① symmetry: $W_N^{k(N-n)} = W_N^{-kn} = (W_N^{kn})^*$

③ periodicity in (n) and (k)

$$W_N^{kn} = W_N^{k(n+N)} = W_N^{n(k+N)}$$

(1)

* If $N = 1000 \Rightarrow$ 1 million MADS

Computation grows very fast as N grows to 2000, 3000, etc.

* Fast Fourier Transform (FFT).

$N = P_1 \cdot P_2 \cdots P_N$ ($P_i \rightarrow$ primes) — no need to be primes (just prods of numbers)

can be expressed as product of Primes.

Complex MADS $\propto N[P_1 + P_2 + \cdots + P_N]$ — not to N^2 but

* Larger the no of primes and the smaller these numbers the higher the efficiency of FFT.

$N = 2^{\sim} \Rightarrow N \log_2 N$

$$X(k) = \sum_{n=0}^{N-1} x(n) W_N^{nk}$$

$$= \underbrace{\sum_{n \text{ even}} x(n) W_N^{nk}}_{n=2r} + \underbrace{\sum_{n \text{ odd}} x(n) W_N^{nk}}_{n=2r+1}$$

$r = 0, 1, \ldots, \frac{N}{2}-1$

$$X(k) = \sum_{r=0}^{\frac{N}{2}-1} x(2r) W_N^{2rk} + \sum_{r=0}^{\frac{N}{2}-1} x(2r+1) W_N^{(2r+1)k}$$

$$W_N^{(2r+1)k} = W_N^{k} W_N^{2rk}$$

$$W_N^2 = e^{\frac{-j2\pi}{N} \cdot 2} = e^{-j2\pi/(N/2)} = W_{N/2}$$

So

$$X(k) = \sum_{r=0}^{\frac{N}{2}-1} x(2r) \underbrace{W_{N/2}^{rk}}_{\frac{N}{2}\text{ DFT}} + W_N^k \sum_{r=0}^{\frac{N}{2}-1} x(2r+1) \underbrace{W_{N/2}^{rk}}_{\frac{N}{2}\text{ DFT}}$$

combine to get $X(k)$

$$X(k) = \underbrace{\sum_{r=0}^{\frac{N}{2}-1} x(2r)\, W_{\frac{N}{2}}^{rk}}_{\frac{N}{2}\text{ point DFT}} + W_N^k \underbrace{\sum_{r=0}^{\frac{N}{2}-1} x(2r+1)\, W_{\frac{N}{2}}^{rk}}_{\frac{N}{2}\text{ point DFT}}$$

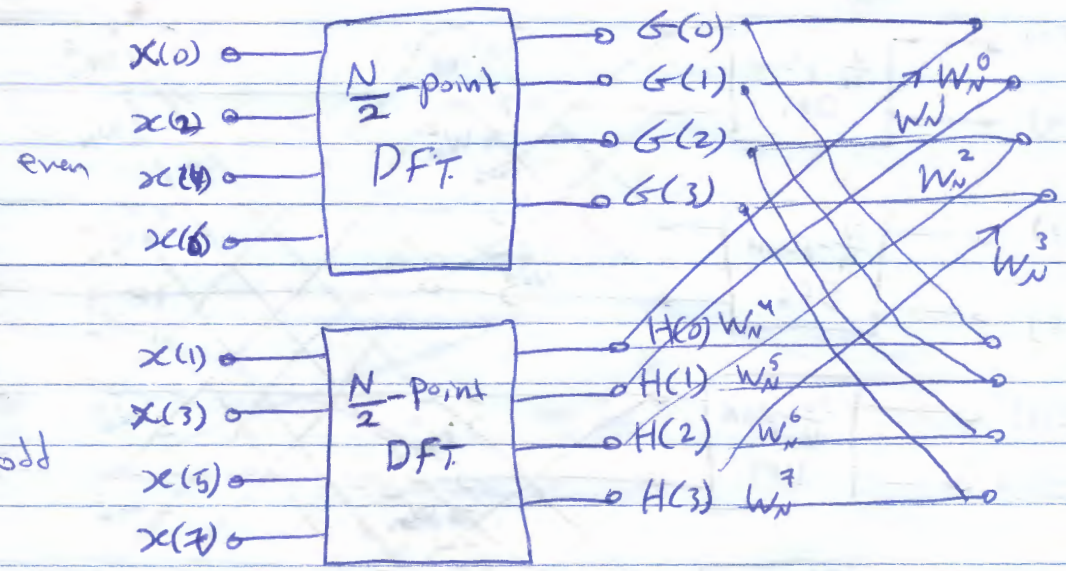
$G(k)$ $\qquad$ $H(k)$

$$X(k) = \underbrace{G(k)}_{(\frac{N}{2})^2 \text{ MADS}} + \underbrace{W_N^k}_{N} \underbrace{H(k)}_{(\frac{N}{2})^2 \text{ MADS}}$$

N-point DFT $\propto N^2$ MADS

So, Computation is $2\left(\frac{N}{2}\right)^2 + N = N + \frac{N^2}{2}$ MADS.

* in computing $G(k)$ $\left(k \to 0 \to \frac{N}{2}-1\right)$, then we use these values for the other values $\frac{N}{2} \to N-1$, because $G(k)$ is periodic in k.



$$X(k) = G(k) + W_N^k H(k)$$

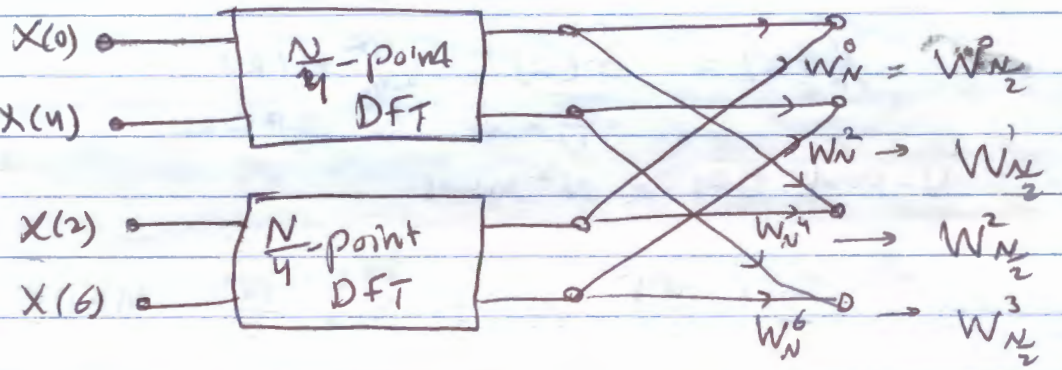
$G(4) = G(0)$, $G(k+4) = G(k)$

$G(k) \to$ periodic in k. $\qquad$ $H(k+4) = H(k)$

$$2\left(\frac{N}{2}\right)^2 + N \text{ MADS} < N^2 \text{ MADS}$$

Now, $\frac{N}{2}$-point DFT is direct computation

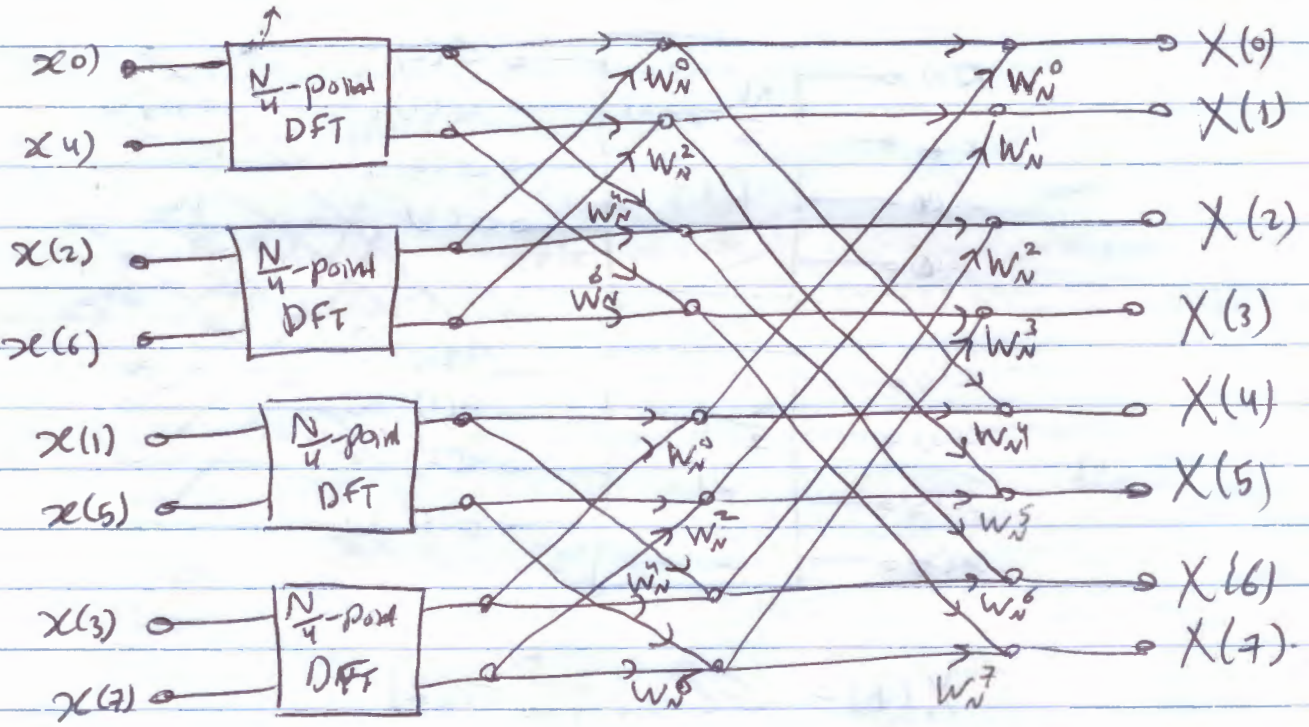
but, if $\frac{N}{2}$ is a composite number, then we can apply the same idea on $\frac{N}{2}$-point DFT.



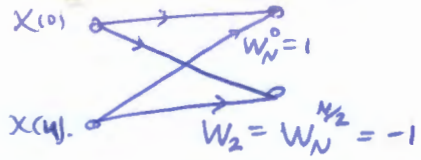
see ①

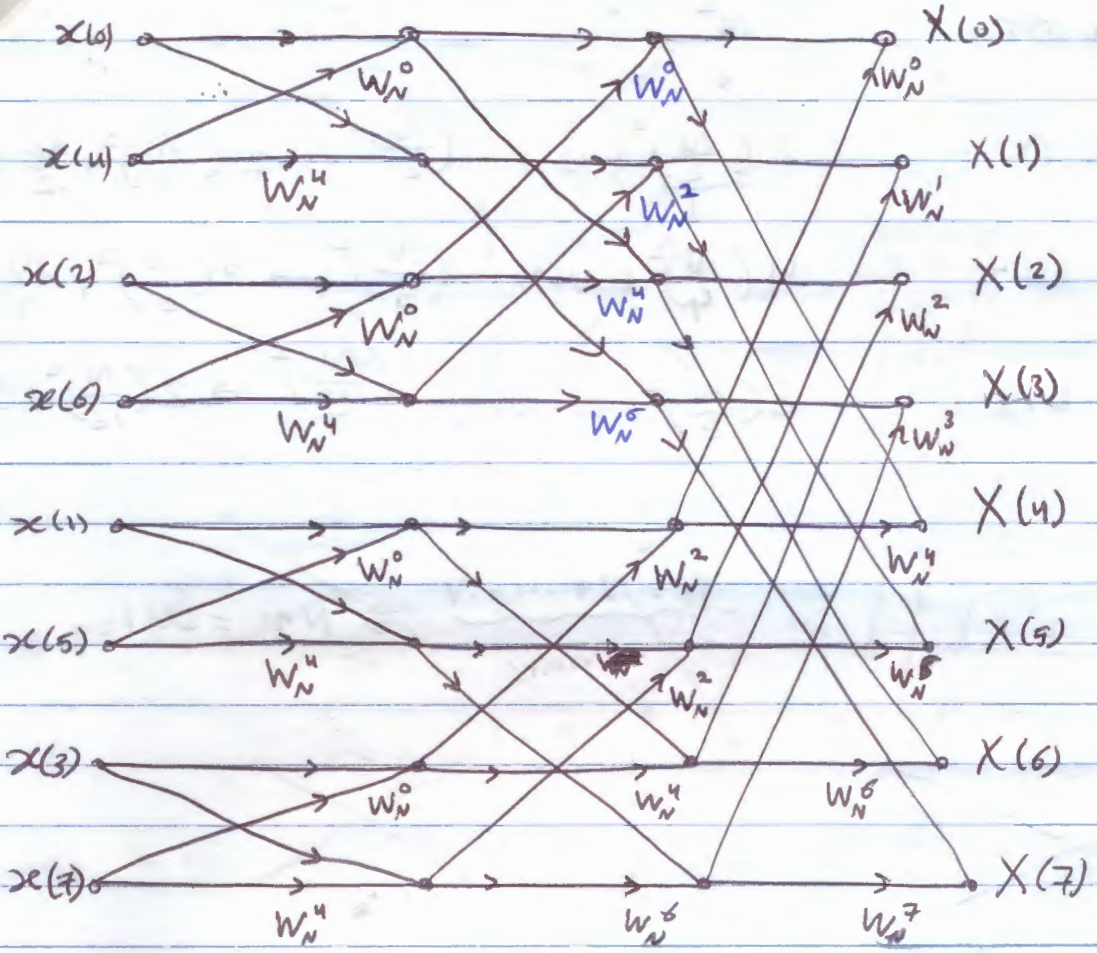
$$W_{\frac{N}{2}} = W_N^2$$

N=8 2-point DFT.

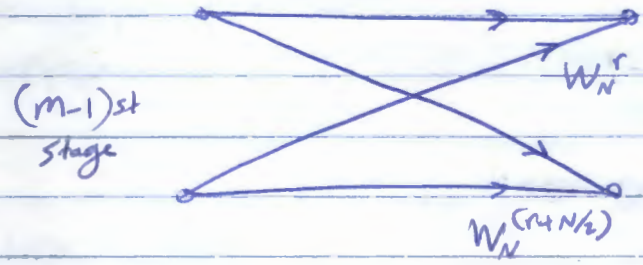


2-point DFT





* Basic butterfly computation (flow graph).



* by counting branches with transmittance of W_N^r, we note that each stage has N complex Mult. and N complex Add.

Since, we have $N \log_2$ stages, we have in total $N \log_2 N$ complex MADS.

e.g. if $N = 1024 = 2^{10} \Rightarrow N^2 = 2^{20} = 1,048,576$

$N \log_2 N = 10,240$ ← reduction of more than two orders of magnitude.

$N = 2^{\nu}$

1 N-point DFT $\quad N^2$

2 $\frac{N}{2}$-point DFT $\quad 2\left(\frac{N}{2}\right)^2 + N$ $\quad \left(\frac{N}{2}\right)^2 \rightarrow 2\left(\frac{N}{4}\right)^2 + \frac{N}{2}$

4 $\frac{N}{4}$-point DFT $\quad 4\left(\frac{N}{4}\right)^2 + N + N$ $\quad \left(\frac{N}{4}\right)^2 \rightarrow 2\left(\frac{N}{8}\right)^2 + \frac{N}{4}$

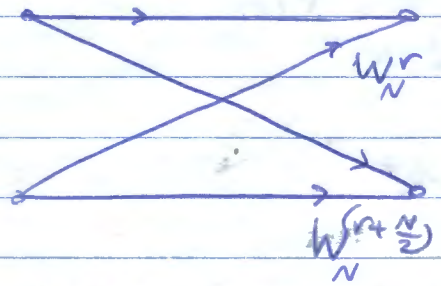
8 $\frac{N}{8}$-point DFT $\quad 8\left(\frac{N}{8}\right)^2 + N + N + N$ $\quad \left(\frac{N}{8}\right)^2 \rightarrow 2\left(\frac{N}{16}\right)^2 + \frac{N}{8}$

$\vdots$

$\underbrace{N + N + \cdots + N}_{\nu \text{ times}} \Rightarrow N\nu = N \log_2 N$

see (2)

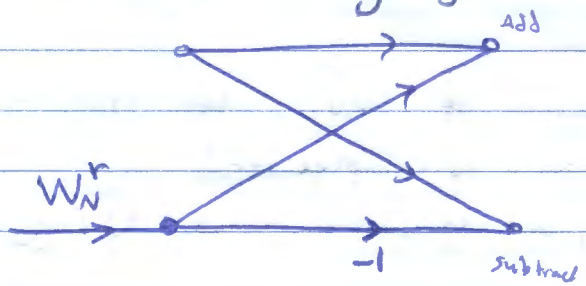
butterfly



$$W_N^{(r+\frac{N}{2})} = W_N^r \, W_N^{\frac{N}{2}}$$

$$W_N^{\frac{N}{2}} = e^{-j\frac{2\pi}{N}\frac{N}{2}} = e^{-j\pi} = -1$$

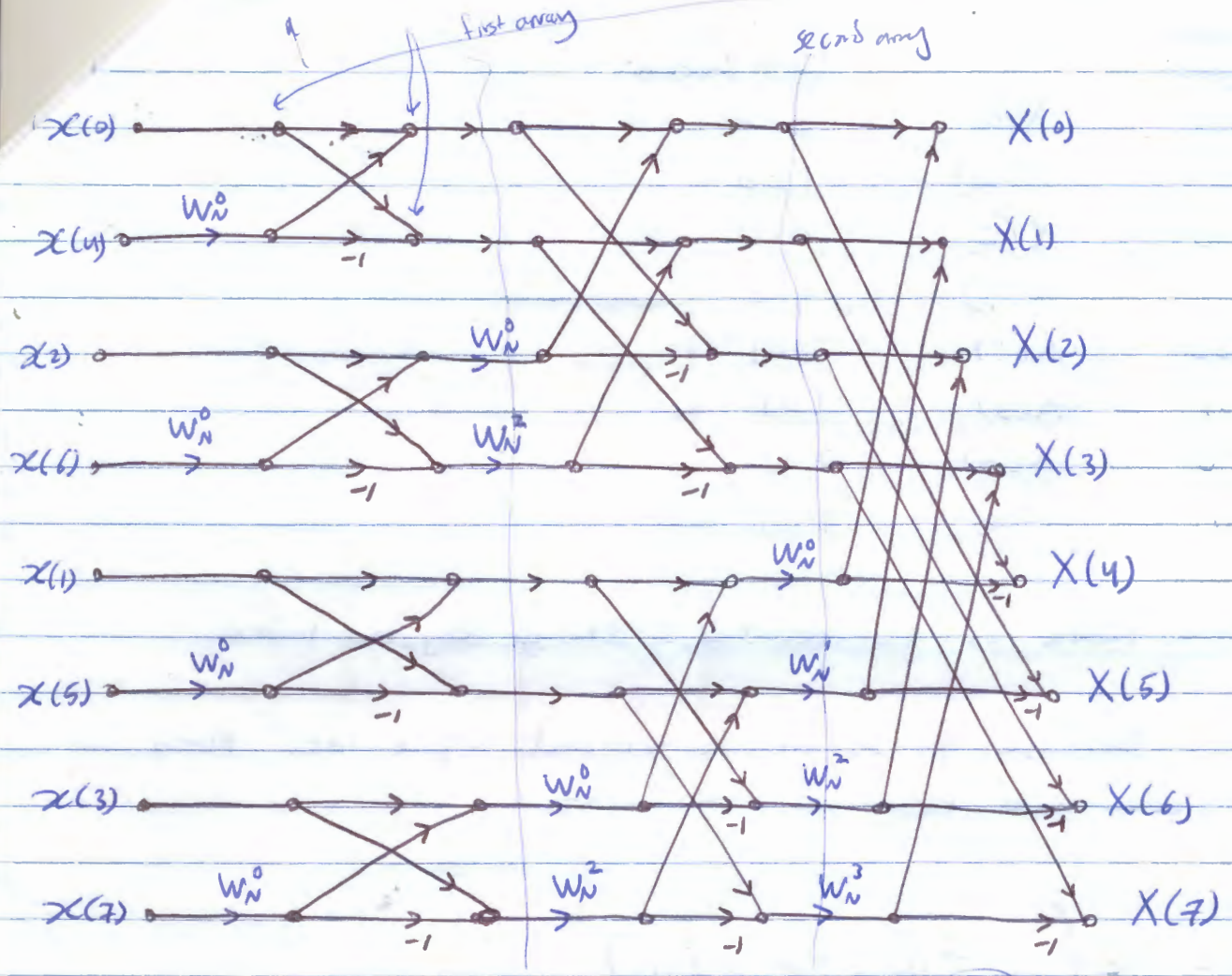
replace basic butterfly by this Network:



by this, we reduce multiplications by factor of 2

don't need it later.

depend on in_1 and in_2 (so, we can update input memory location)



This is one algorithm for computing (FFT) DFT. (decimation-in-time)
(flow-graph form)

~~Some notice points:~~

* ~~Input seq.~~ re-arrange → bit-reversed order of binary representation of indeces. → how to arrange memory storage.

- In-place Computation (we can use the same array for input, first array, second array, and output array.)

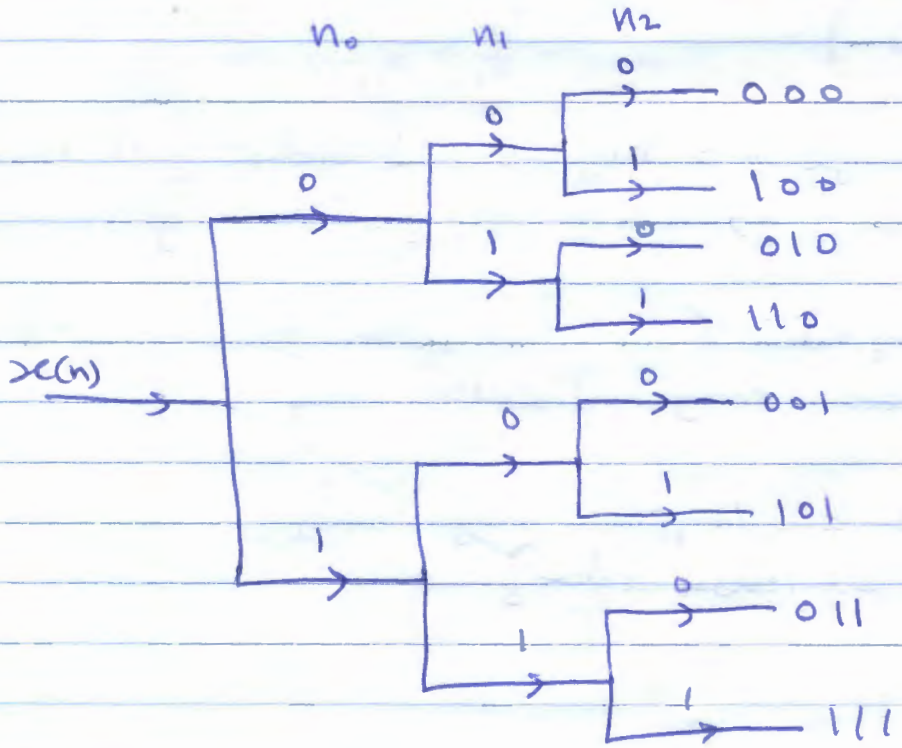
{so, this algorithm is an in-place algorithm}.
{input should be in a bit-reversed order}.

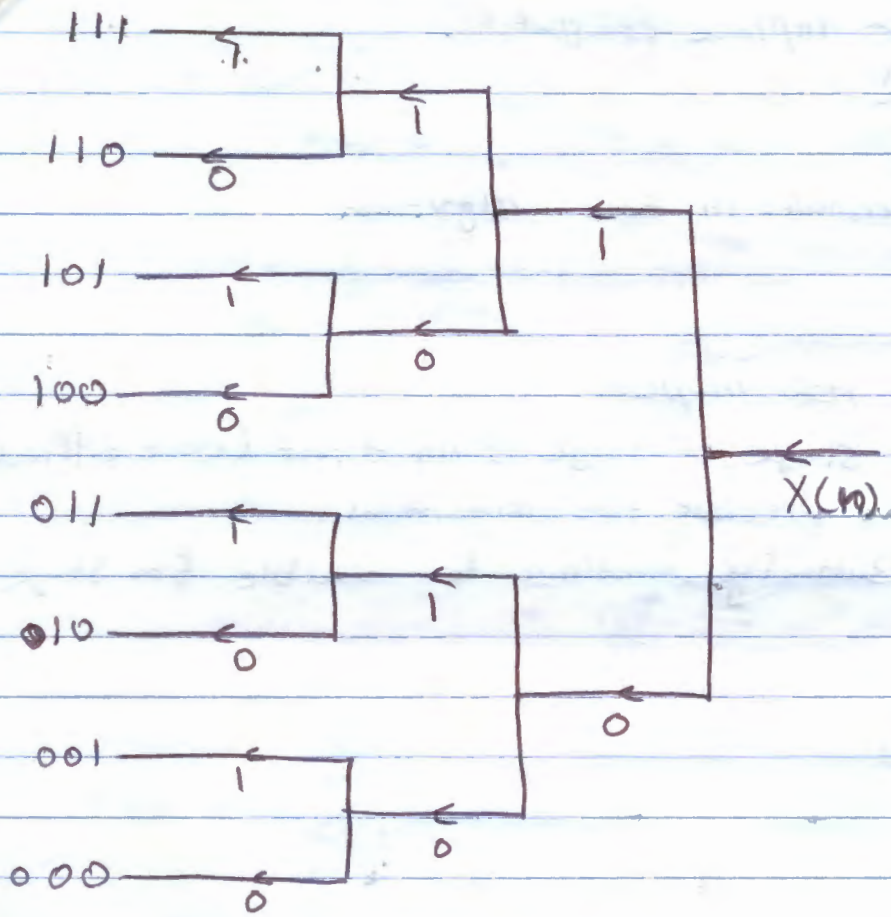
	Storage Register		Data Index
0	000	X(0)	000
1	001	X(4)	100
2	010	X(2)	010
3	011	X(6)	110
4	100	X(1)	001
5	101	X(5)	101
6	110	X(3)	011
7	111	X(7)	111

* Data index is bit-reversed of storage Register index.

*reason, because of we divide original input into even and odd and then even into even and odd ... etc

$$x(n) = x\left(\ldots 2^2 n_2 + 2^1 n_1 + 2^0 n_0\right)$$





* Sorting input data into bit-reversed order ⇒ output is normal index order.

* back to find flow-graph of FFT algorithm, to keep input data in normal order, we can re-order the horizontal lines in the graph to get inputs in normal order. But in this way the order of outputs will be bit-reversed order.

x(0)	X(0)
x(1)	X(4)
x(2)	X(2)
x(3)	X(6)
x(4)	X(1)
x(5)	X(5)
x(6)	X(3)
x(7)	X(7)

* This is not In-place computation algorithm

* Input - normal order ⇒ output → normal order
* Computation is no longer inplace computation
* butterflys one distorted.

+ This is the formal Decimate-in-time algorithm:

disadvages.

① Computation is not In-place.

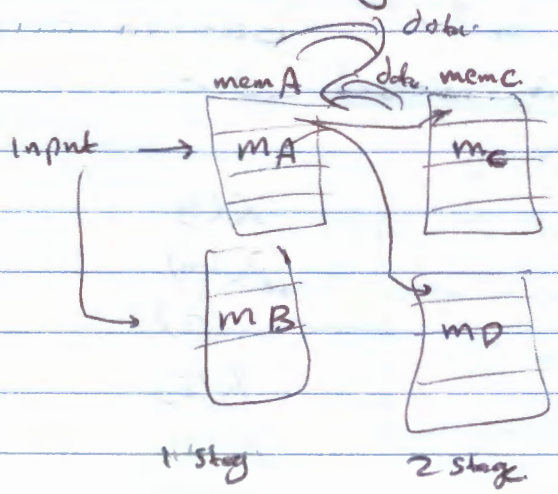
② Indexing from stage to stage is not direct (more difficult) compared with the previous two algorithms.

③ Hieght or width of Butterfly continue to double from stage to stage.

- RAM is required

* New algorithm (sequential memory is used rather than Random access memory) (shift Reg. memory).

- arrangment of stages is identical.
(Indexing is identical in all stages).



* flush data into memA and memB then into memC and memD
Then flush back data from memC and memD into memA and memB.

Four separate seq. memories

[not. in-place computation]

* Decimation in Frequency

* Instead of separating input points into even and odd numbers, we compute DFT for even output points and odd output points.

$$X(k) = \sum_{n=0}^{N-1} x(n)\, W_N^{nk} \quad \rightarrow \text{two}$$

$$= \sum_{n=0}^{\frac{N}{2}-1} x(n)\, W_N^{nk} + \sum_{n=\frac{N}{2}}^{N-1} x(n)\, W_N^{nk}$$

$$= \sum_{n=0}^{\frac{N}{2}-1} x(n)\, W_N^{nk} + \sum_{n=0}^{\frac{N}{2}-1} x\left(n+\frac{N}{2}\right) W_N^{\left(n+\frac{N}{2}\right)k}$$

$$= \sum_{n=0}^{\frac{N}{2}-1} x(n)\, W_N^{nk} + W_N^{\frac{Nk}{2}} \sum_{n=0}^{\frac{N}{2}-1} x\left(n+\frac{N}{2}\right) W_N^{nk}$$

$$X(k) = \sum_{n=0}^{\frac{N}{2}-1} \left[x(n) + (-1)^k x\left(n+\frac{N}{2}\right) \right] W_N^{nk}$$

looks like $\frac{N}{2}$-point DFT, but it is not because W_N^{nk}

Let's look at these output points when even, odd

k even $\rightarrow X(2r)$

$$\sum_{n=0}^{\frac{N}{2}-1} \left[x(n) + x\left(n+\frac{N}{2}\right) \right] W_N^{2rn}$$

$r = 0, 1, 2, \ldots, \frac{N}{2}-1$

k odd $\rightarrow X(2r+1) \quad / \; r = 0, 1, 2, \ldots, \frac{N}{2}-1$

$$\sum_{n=0}^{\frac{N}{2}-1} \left[x(n) - x\left(n+\frac{N}{2}\right) \right] W_N^{n} W_N^{2rn}$$

$$W_N^{2rn} = W_{\frac{N}{2}}^{rn}$$

- k even

$$X(2r) = \sum_{n=0}^{\frac{N}{2}-1} g(n)\, W_{\frac{N}{2}}^{rn} \quad , \quad \frac{N}{2}\text{-point DFT}$$

where, (first half of data point, last half of input points)

$$g(n) = x(n) + x\left(n+\frac{N}{2}\right)$$

- k odd

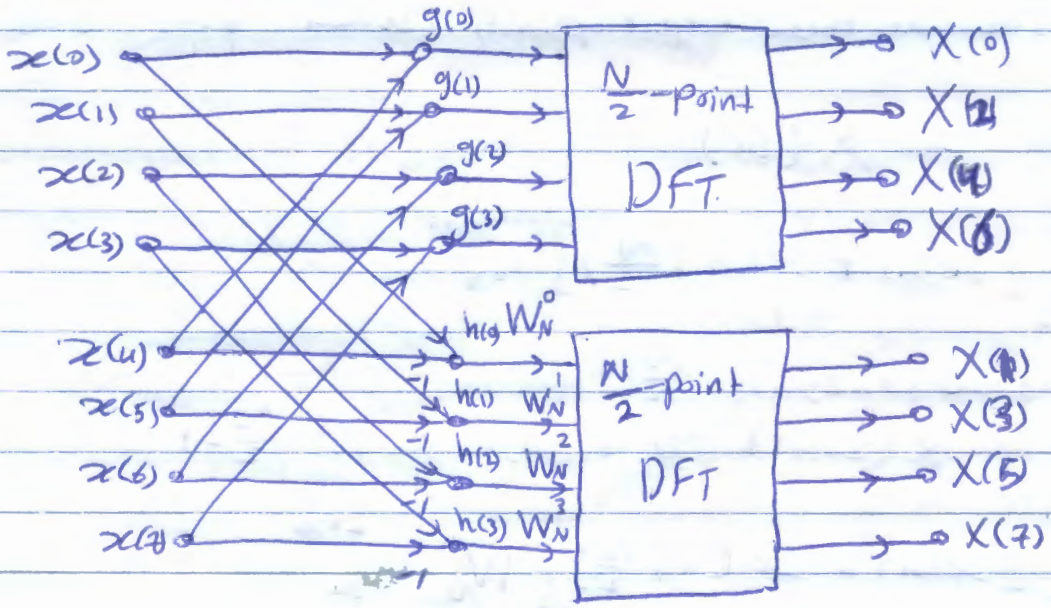
$$X(2r+1) = \sum_{n=0}^{\frac{N}{2}-1} \left\{ h(n)\, W_N^{n} \right\} W_{\frac{N}{2}}^{rn}$$

where

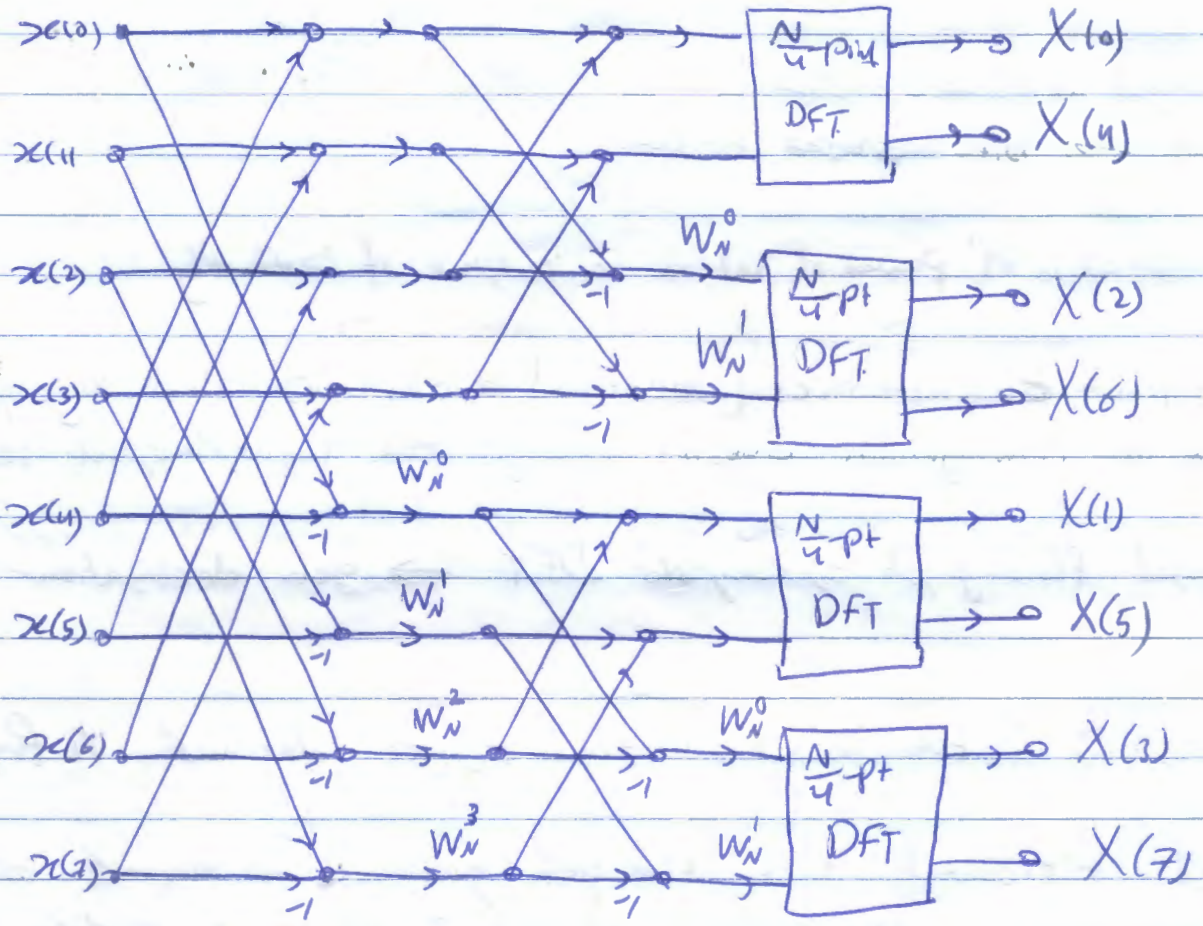
$$h(n) = x(n) - x\left(n+\frac{N}{2}\right)$$

(first half, second half)

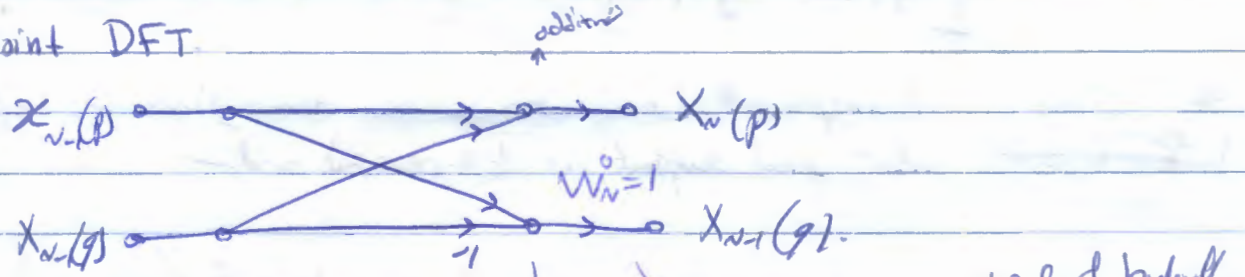
# of computations (MADS) same as decimate in time

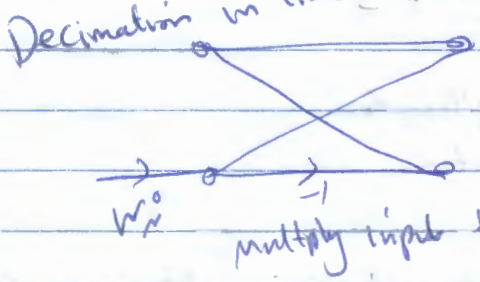


Similar to DIT, decompose $\frac{N}{2}$-point into $\frac{N}{4}$-point



* 2-point DFT





* show final flow graph for decimation-in-freq.

decimation in freq. - ... Continue

data input → Normal order
output → bit-reversed order.

→ multiplication of powers of W is on output of butterfly

* Note.
one of flow-graph characteristics (mentioned earlier) that by applying transposition on flow-graph doesn't change the input/output relationship

→ Transposed flow-graph ⇒ also computes DFT. → get decimation-in-time.

Transposition of ~~time~~ decimation in time ⇒ get decimation in freq.

⇒ sort hirizontal line to get outputs in normal order
inputs are in bit-reversed order (In-place computation)
[show graph]

* This is transposed (decimation_in_time) when input is in ~~bit-reversed~~ normal order and output in bit-reversed order

* we can also arrange it to get input (normal) and output in normal order.

[show graph]

- Indexing is more complicated.
- not in-place computation

* final re-arrangment of decim. in freq. [sequencial memory]

[show graph]

- Index is straightforward (identical from stage to stage).
- input (Normal), output bit-reversed order
- we sequential Memory (disk, ROM, tape, shift Reg), with no need for RAM.

- Four memories. A C
  B D

store first_half in A
" second " " B

first point in mem. A
" " " " B → store it in mem. C location 1 and
" " " " " location 2

---

*Some computational issues: (Considerations).

① Inverse DFT.

$$x(n) = \frac{1}{N} \sum_{k=0}^{N-1} X(k) W_N^{-nk}$$

Complex Conjugate of

DFT.

$$X(k) = \sum_{n=0}^{N-1} x(n) W_N^{kn}$$

To use algorithm of DFT for IDFT.

* $\frac{1}{N}$ → scaling.

W_N^{-kn} * -ve → conjugate powers of W. (Coeff
or flipping input * ⇒ $x(-n)$

② bit-reversal. (in-place Computation) (no-need for double space)

0 1 2 3 4 5 6 7

0 4 2 6 1 5 3 7

Swap place 1, 4
3, 6

* Implementing bit-reversed in HW, we just re-arrange the wires.

* To Implement bit-reversal in index register ✗

third comp. Issue

(3) dealing with Coefficients

① store them in table.

② generating coeff. as we need them (recursively).
power of W changes from stage to stage.

in FFT, Coeff are accessed in normal order

[show decim in freq., input-normal, output-bit-reversed)
W_N → normal order)

[show deci_in_time] [input-normal, output, bit reversed]

W_N → bit-reversal order

* Coefficients → stored in normal order or bit-reversed order.

---

* To use FFT for computing Convolution or Correlation with help
we find transform of sequences, Multiply in freq. domain
and then, we take IDFT of the product.

Now, in this case, we can avoid the bit-reversed order.

For example, we can choose algorithm for DFT where input is in normal order and output in bit-reversed order, and assume impulse response is stored in bit-reversed order, we do Multiplication, and we choose an algorithm for IDFT, where input is bit-reversed order and output in normal order.

for DFT

— decimation-in-time [normally order input, bit-reversal output]
Coefficients (W_N) are in bit-reversal order)

for IDFT

— decimation-in-freq. [bit-reversal input, normally ordered output]
Coeff. (W_N) are in normal order)

⇒ To match up Coeff. W_N order [deci. in-time and deci-in-freq.]

For — decimation in freq. [normally input, output bit-reversal order]
W_N → normal order
— decimation-in-time [input is in bit-reversal order
outputs are in normal order]
W_N Coefficients are also in normal order.

all of previous are called radix-2 FFT algorithms.

* general radix FFT

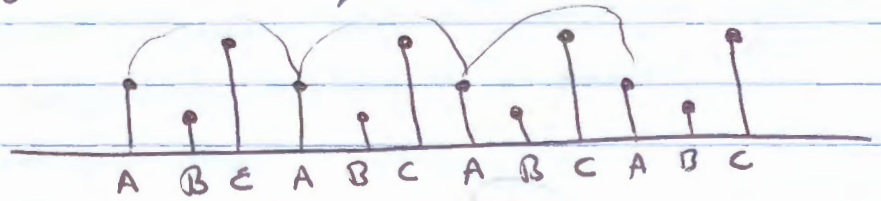
$$N = \underbrace{P_1 \cdot P_2 \cdots P_N}_{Q_1}$$

$$N = P_1 \cdot Q_1$$

decompose seq. into;
subsequence : every P_1^{th} point. [paralling decimation-in-time]

P_1 sequences of length Q_1

$N = 12$, $P_1 = 3$ chosen.
gen. 3 subsequences of length 4, every 3 point.



A AAA, BBBB, CCCC,

$$X(k) = \sum_{n=0}^{N-1} x(n)\, W_N^{kn}$$

$$= \sum_{r=0}^{q_1-1} x(p_1 r)\, W_N^{p_1 rk} + \sum_{r=0}^{q_1-1} x(p_1 r+1)\, W_N^{(p_1 r+1)k}$$

$$= \sum_{l=0}^{p_1-1} W_N^{lk} \sum_{r=0}^{q_1-1} x(p_1 r+l)\, W_N^{p_1 rk}$$

$$W_N^{p_1 rk} = e^{j\frac{2\pi}{N} p_1 rk} = e^{j\frac{2\pi}{q_1} rk} = W_{q_1}^{rk}$$ ($N = p_1 \cdot q_1$)

$$X(k) = \sum_{l=0}^{p_1-1} W_N^{lk} \underbrace{\sum_{r=0}^{q_1-1} x(p_1 r+l)\, W_{q_1}^{rk}}_{q_1\text{-point DFT}}$$

$q_1 = p_2 \cdot \underbrace{p_3 \cdots p_N}_{q_2} \Rightarrow$ we proceed

MADS $\Rightarrow$

$$N\left[p_1 + p_2 + \cdots + p_N - \nu\right]$$

* radix-2 algorithms are the most efficient algorithms, and to make N as 2^n we can augment it with zeros to make a power of 2.