FFT

DFT Computation:

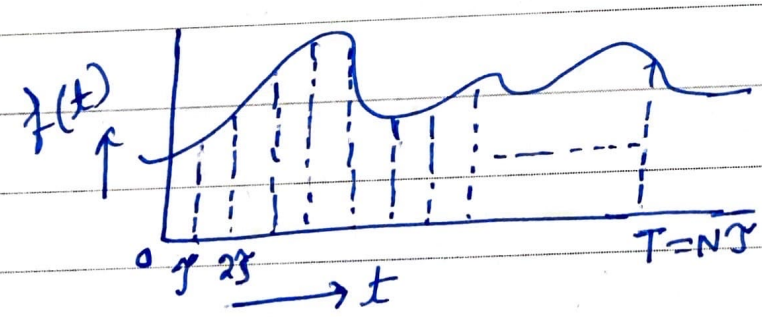
Fourier Transform of a function $f(t)$:

$$F(w) = \int_{-\infty}^{+\infty} f(t)\, e^{-jwt}\, dt \quad \text{---- (1)}$$

Inverse Fourier Transform:

$$f(t) = \frac{1}{2\pi} \int_{-\infty}^{+\infty} F(w)\, e^{jwt}\, dw \quad \text{---- (2)}$$

To evaluate $F(w)$ from a given function $F(t)$ by a computer, we approximate the time interval of integration ranging from $t=0$ to $t=T$, (i) and sample the function $f(t)$ at some discrete time instants to get a finite number of sample values and (ii) replace the integration by summation over these finite number of sample values.



T = sampling interval

within the time interval $T = NT$, we take N samples at $t = 0, T, 2T, \ldots, (N-1)T$. We denote the sampled values as $f(0) = f_0$, $f(T) = f_1$, $f(2T) = f_2$, $f(3T) = f_3$, ..., $f((N-1)T) = f_{N-1}$.

Look at the integral now:

$$F(w) = \int_{-\infty}^{+\infty} f(t)\, e^{-jwt}\, dt$$

Replace the integral from 0 to T:

$$F(w) = \int_{0}^{T} f(t)\, e^{-jwt}\, dt$$

Replace $t = kT$, the discrete time instant corresponding to k^{th} sampling instant. $0 \le k \le N-1$. As this is an aperiodic signal $w = r\omega_0$ where, ω is r^{th} harmonic of the fundamental angular frequency $\omega_0 = \frac{2\pi}{T}$ $\therefore \omega_0 = \frac{2\pi}{NT}$. After NT it will repeat from origin. *

$$\therefore w = r\omega_0 = \frac{r \cdot 2\pi}{NT} = \frac{2\pi r}{NT}$$

Now replace $f(t)$ by $f(kT) = f_k$

$$A_r \Rightarrow \sum_{k=0}^{N-1} f_k \cdot e^{-jwt}$$

$$A_r = \sum_{k=0}^{N-1} f_k \cdot e^{-j \cdot \frac{2\pi r}{NT} \cdot kT}$$

$$= \sum_{k=0}^{N-1} f_k \cdot e^{-\frac{2\pi j}{N} \cdot rk}$$

where $r = 0, 1, 2, \cdots, N-1$

$$\therefore A_r = \sum_{k=0}^{N-1} f_k \cdot W^{rk} \quad \text{where } W = e^{-2\pi j/N}$$

[from Argand's plane, $W = \left(\cos\left(\frac{2\pi}{N}\right) - j\sin\left(\frac{2\pi}{N}\right)\right)$]

So, we get the discrete Fourier Transform (DFT) co-efficients as:

$$A_r = \sum_{k=0}^{N-1} f_k \cdot W^{rk} \quad \text{---------- (3)}$$

The inverse discrete Fourier transform (IDFT) co-efficients are likewise defined as

$$f_k = \frac{1}{N} \sum_{r=0}^{N-1} A_r W^{-rk} \quad \text{---- (4)}$$

Computational problem:

How to evaluate the DFT co-efficients, $A_0, A_1, \cdots, A_{N-1}$?

Example:

Let $N = 4$

We then have four samples f_0, f_1, f_2, f_3

$$W = e^{\frac{-2\pi j}{4}}$$

$$A_0 = f_0 \cdot W^0 + f_1 W^0 + f_2 W^0 + f_3 W^0 = f_0 + f_1 + f_2 + f_3$$

$$A_1 = f_0 W^0 + f_1 W^1 + f_2 W^2 + f_3 W^3$$

$$A_2 = f_0 W^0 + f_1 W^2 + f_2 W^4 + f_3 W^6$$

$$A_3 = f_0 W^0 + f_1 W^3 + f_2 W^6 + f_3 W^9$$

* As we assumed after NT it will repeat from origin.

$$\therefore A_{r+N} = \sum_{k=0}^{N-1} f_k \cdot W^{(r+N)k}$$

Now $W^{NK} = e^{\frac{-2\pi j}{N} \cdot Nk}$

$$= e^{-2\pi j k}$$

$$= \cos(2\pi k) - j \sin(2\pi k)$$

$$= 1 - j \cdot 0 = 1$$

$$\therefore A_{r+N} = \sum_{k=0}^{N-1} f_k \cdot W^{rk} \cdot W^{NK} = \sum_{k=0}^{N-1} f_k \cdot W^{rk} = A_r$$

(proved)

$$\begin{bmatrix} A_0 \\ A_1 \\ A_2 \\ A_3 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & W & W^2 & W^3 \\ 1 & W^2 & W^4 & W^6 \\ 1 & W^3 & W^6 & W^9 \end{bmatrix} \begin{bmatrix} f_0 \\ f_1 \\ f_2 \\ f_3 \end{bmatrix}$$

In general, we can express DFT co-efficients $A_0, A_1, A_2, \cdots, A_{N-1}$ as

$$\begin{bmatrix} A_0 \\ A_1 \\ A_2 \\ \vdots \\ A_{N-1} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & W & W^2 & \cdots & W^{N-1} \\ 1 & W^2 & W^4 & \cdots & W^{2(N-1)} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ 1 & W^{N-1} & W^{2(N-1)} & \cdots & W^{(N-1)(N-1)} \end{bmatrix} \begin{bmatrix} f_0 \\ f_1 \\ f_2 \\ \vdots \\ f_{N-1} \end{bmatrix}$$

There is a matrix by vector multiplication involved in the computation. Multiplication of each row of W-matrix by $\begin{bmatrix} f_0 \\ f_1 \\ \vdots \\ f_{N-1} \end{bmatrix}$ involves N scalar multiplications. So, the DFT computation needs N^2 scalar (complex) multiplication as $N = e^{-2\pi j/N}$

$$= \cos\left(\frac{2\pi}{N}\right) - j \sin\left(\frac{2\pi}{N}\right)$$

Fast Fourier Transform (FFT):

Let the samples $f_0, f_1, f_2, \cdots, f_{N-1}$ be the N sample values obtained dividing the time for $t=0$ to $t=T$.

Let us assume that N is a power of 2, i.e., let $N=2^R$.

We divide the N samples in even-numbered and odd-numbered samples so that the even-numbered samples X_i, $i = 0, 1, 2, \cdots, N/2-1$ are given as

$X_0 = f_0, X_1 = f_2, X_2 = f_4, \cdots, X_{N/2-1} = f_{N-2}$

odd-numbered samples Y_i, $i = 0, 1, 2, \cdots, N/2-1$ are given as

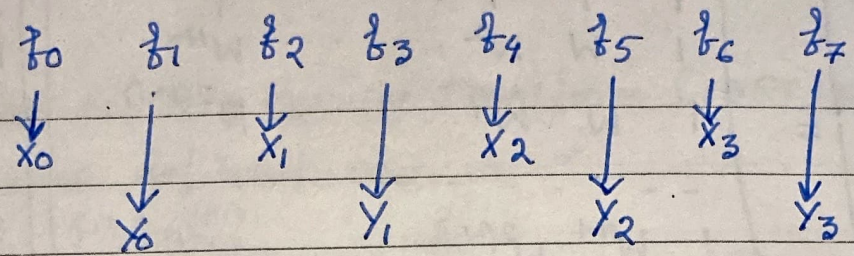
$Y_0 = f_1, Y_1 = f_3, Y_2 = f_5, \cdots, Y_{N/2-1} = f_{N-1}$

$$\boxed{\begin{aligned} X_k &= f_{2R} \\ Y_k &= f_{2R+1} \end{aligned}}$$

$$\{f_0, f_2, f_4, \cdots, f_{N-2}\} = \{X_0, X_1, X_2, \cdots, X_{\frac{N}{2}-1}\}$$

$$\{f_1, f_3, f_5, \cdots, f_{N-1}\} = \{Y_0, Y_1, Y_2, \cdots, Y_{N/2-1}\}$$

Example: $N=8$



Now, write the DFT co-efficients A_r as

$$A_r = \sum_{k=0}^{N-1} f_k \cdot W^{rk}$$

$$= \sum_{k=0}^{N/2-1} f_{2k} \cdot W^{r \cdot 2k} + \sum_{k=0}^{N/2-1} f_{2k+1} \cdot W^{r(2k+1)}$$

$$= \sum_{k=0}^{N/2-1} f_{2k} \cdot (W^2)^{rk} + \sum_{k=0}^{N/2-1} f_{2k+1} \cdot W^r \cdot (W^2)^{rk}$$

$$= \sum_{k=0}^{N/2-1} x_k \cdot (W^2)^{rk} + \sum_{k=0}^{N/2-1} y_k \cdot W^r \cdot (W^2)^{rk}$$

Assume that we have only $N/2$ samples $x_0, x_1, x_2, \cdots, x_{N/2-1}$. The DFT coefficients corresponding to the above $N/2$ samples can be expressed as

$$B_r = \sum_{k=0}^{N/2-1} x_k \cdot e^{-\frac{2\pi j}{N/2} \cdot rk}, \quad r = 0, 1, 2, \cdots, N/2-1$$

$$= \sum_{k=0}^{N/2-1} x_k \cdot e^{-\frac{4\pi j}{N} \cdot rk} = \sum_{k=0}^{N/2-1} x_k \cdot (W^2)^{rk}$$

where $W = e^{-2\pi j/N}$

Similarly, the DFT coefficients corresponding to $N/2$ odd samples $y_0, y_1, y_2, \cdots, y_{N/2-1}$ are expressed as

$$C_r = \sum_{k=0}^{N/2-1} y_k \cdot e^{-\frac{2\pi j}{N/2} \cdot rk}, \quad r = 0, 1, 2, \cdots, N/2-1$$

$$= \sum_{k=0}^{N/2-1} Y_k \cdot e^{-\frac{4\pi j}{N} \cdot rk}$$

$$= \sum_{k=0}^{N/2-1} Y_k \cdot (W^2)^{rk} \quad \text{where, } W = e^{-\frac{2\pi j}{N}}$$

Hence, $$A_r = \sum_{k=0}^{N/2-1} X_k \cdot (W^2)^{rk} + \sum_{k=0}^{N/2-1} Y_k \cdot W^r (W^2)^{rk}$$

$$\therefore A_r = B_r + W^r C_r, \quad r = 0, 1, 2, \cdots, N/2-1$$

After N/2 sample points

Put $r = r + N/2$, in the above expression to get

$$A_{r+N/2} = \sum_{k=0}^{N/2-1} X_k \cdot (W^2)^{(r+N/2)k} + \sum_{k=0}^{N/2-1} Y_k \cdot W^{(r+N/2)} \cdot (W^2)^{(r+\frac{N}{2})k}$$

$$= \sum_{k=0}^{N/2-1} X_k \cdot (W^2)^{rk} \cdot W^{2 \cdot \frac{N}{2} \cdot k} + \sum_{k=0}^{N/2-1} Y_k \cdot W^r \cdot W^{N/2} \cdot (W^2)^{rk} \cdot W^{2 \cdot \frac{N}{2} \cdot k}$$

Now, $W = e^{-2\pi j/N}$

$$\therefore W^N = e^{-\frac{2\pi j}{N} \cdot N} = e^{-2\pi j}$$

$$= \cos(2\pi) - j\sin(2\pi) = 1 - j.0 = 1$$

& $$W^{N/2} = e^{\frac{-2\pi j}{N} \cdot \frac{N}{2}} = e^{-\pi j} = \cos(\pi) - j\sin(\pi) = -1 - 0 = -1$$

Hence, $$A_{r+N/2} = \sum_{k=0}^{N/2-1} X_k \cdot (W^2)^{rk} \cdot (1)^k + \sum_{k=0}^{N/2-1} Y_k \cdot W^r \cdot (-1) \cdot (W^2)^{rk} \cdot (1)^k$$

$$= \sum_{k=0}^{N/2-1} X_k \cdot (W^2)^{rk} - W^r \sum_{k=0}^{N/2-1} Y_k \cdot (W^2)^{rk}$$

$$= B_r - W^r C_r$$

Then we get, $A_r = B_r + W^r C_r, \quad r = 0, 1, 2, \cdots, N/2-1$

$A_{r+N/2} = B_r - W^r C_r, \quad r = 0, 1, 2, \cdots, N/2-1$

Hence, if we compute the $N/2$ DFT coefficients B_r, $r=0,1,2,\cdots,N/2-1$ from even samples, and also the $N/2$ DFT coefficients C_r, $r=0,1,2,\cdots,N/2-1$ from odd samples, then the N DFT coefficients A_r, $r=0,1,2,\cdots,N/2-1$ and $A_{r+N/2}$, $r=0,1,2,\cdots,N/2-1$ can be obtained by one more multiplication $(W^r \cdot C_r)$ and two more addition/subtraction.

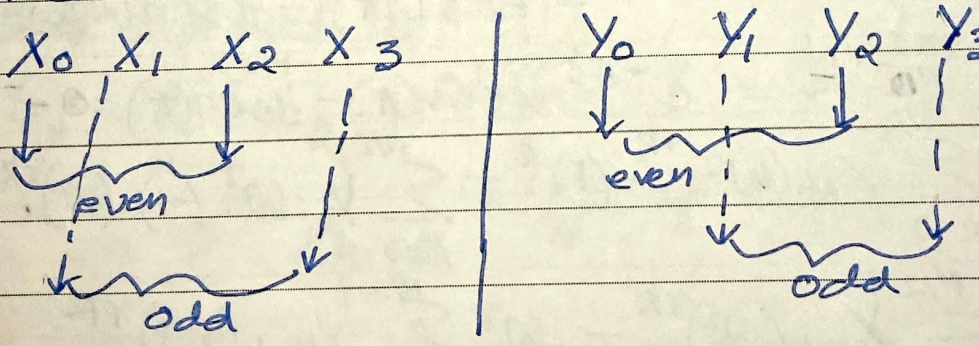
If $T(N)$ be the total number of multiplications required to evaluate N DFT coefficients, then

$$T(N) = 2T(N/2) + N/2 \quad \cdots\cdots (1)$$ where

$N/2$ = # multiplications,

because we have to compute $W^r C_r$, $r=0,1,2\cdots,N/2-1$.

We can likewise evaluate B_r and C_r coefficients again by dividing each of the respective sample set in even and odd-numbered samples:



Since N is a power of 2, $(N=2^k)$, we can recursively follow this technique until the eve/odd numbered sample sets contain only one element each. Note that there is only one simple sample f_0, its DFT coefficient is, by definition $A_0 = f_0 \cdot e^{\frac{-2\pi i}{1} \cdot 0} = f_0$

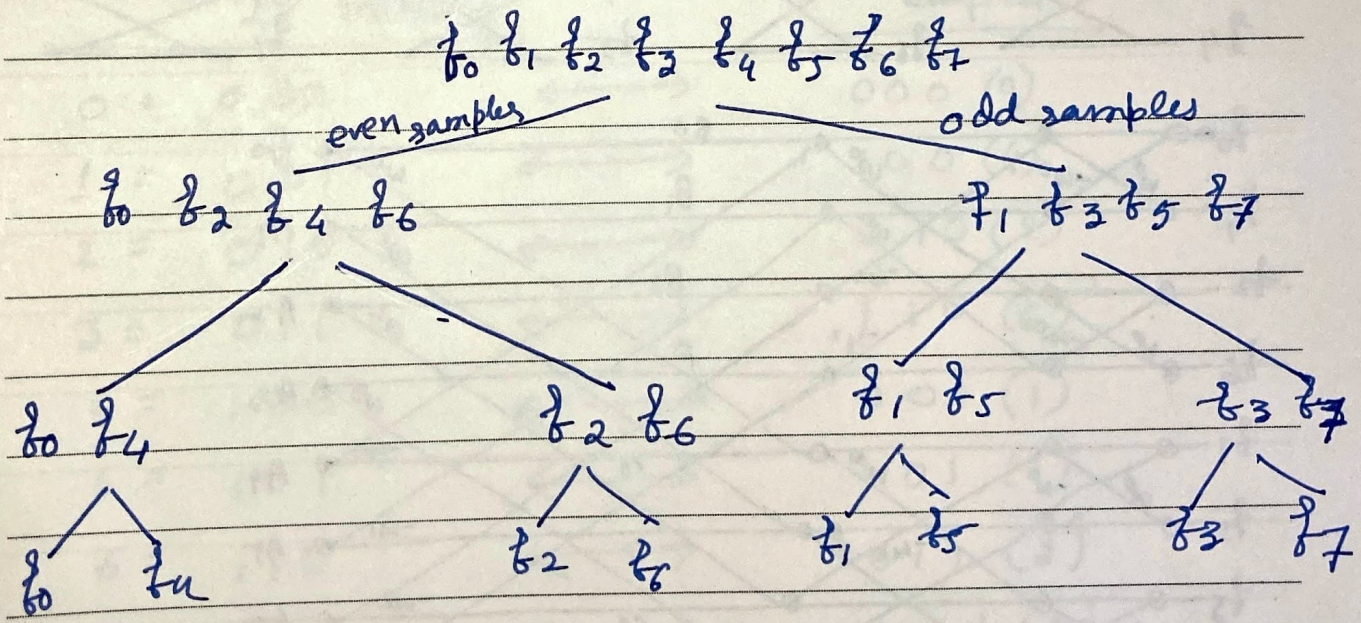
So, we can write

$$T(N) = 2T(N/2) + N/2$$
$$= 2\left[2T(N/2^2) + N/2^2\right] + N/2$$
$$= 2^2 T(N/2^2) + 2 \cdot N/2$$
$$= 2^2\left[2T(N/2^3) + N/2^3\right] + 2 \cdot N/2$$
$$= 2^3 T(N/2^3) + 3 \cdot N/2$$

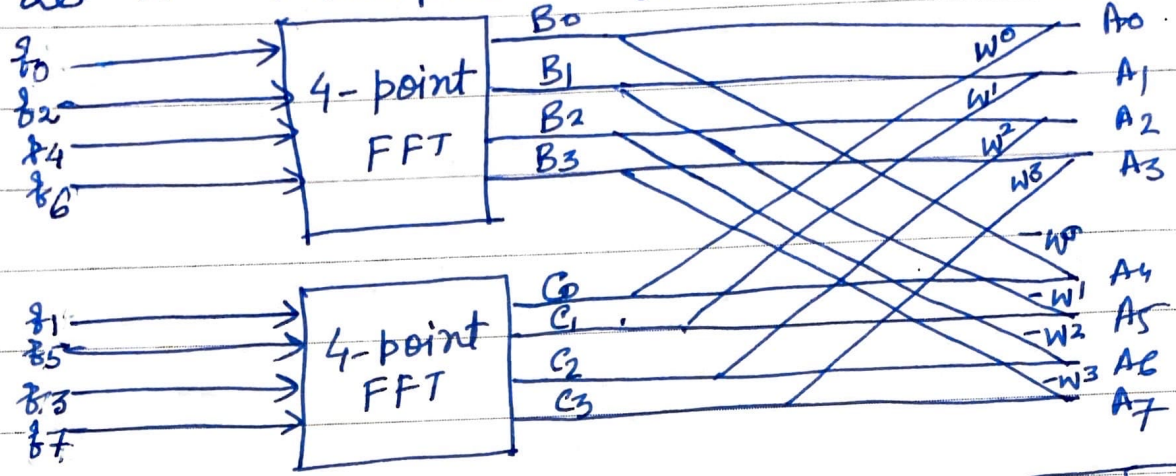
Let k steps are there and $N = 2^k$.

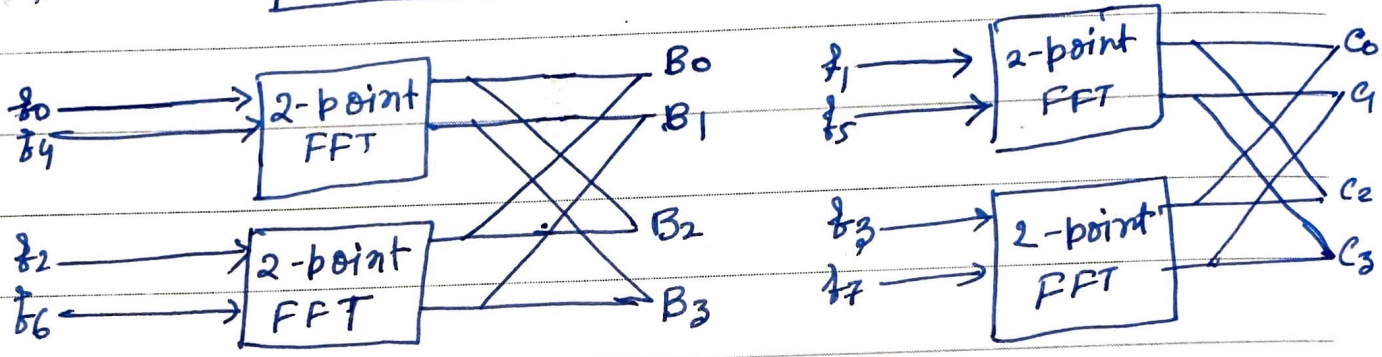
$$\therefore T(N) = 2^k T(N/2^k) + k \cdot N/2$$
$$= N\,T(1) + N\log N/2$$
$$= \frac{N\log N}{2} \quad \text{as } T(1) = 0$$

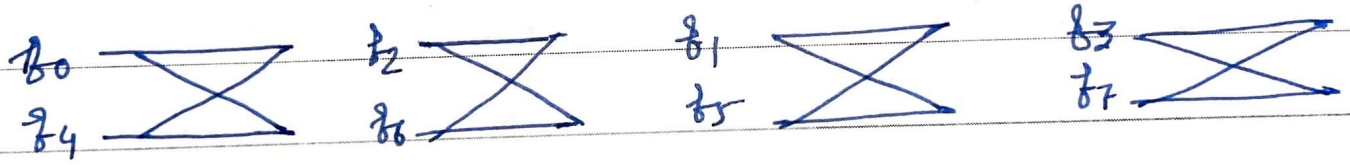
$$\therefore T(N) = O(N\log N)$$

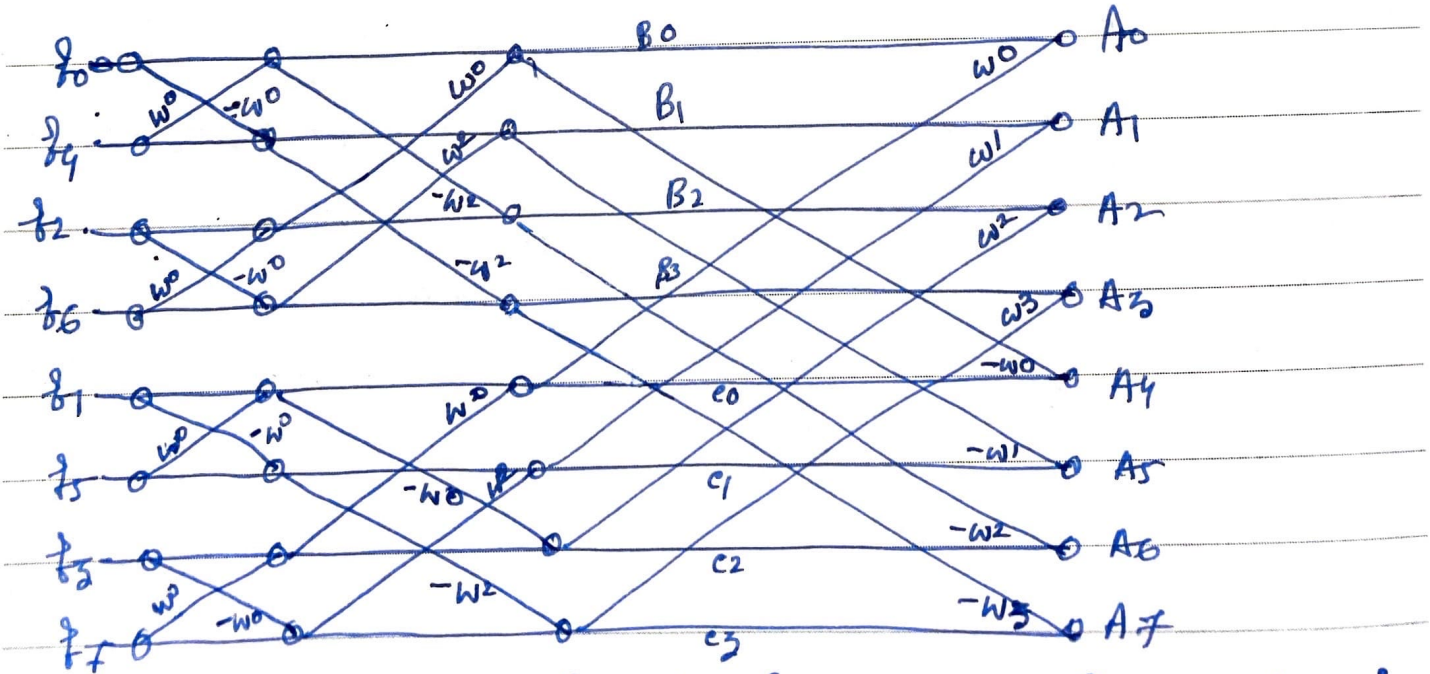


Let $N=8$, samples are $f_0, f_1, f_2, f_3, f_4, f_5, f_6, f_7$

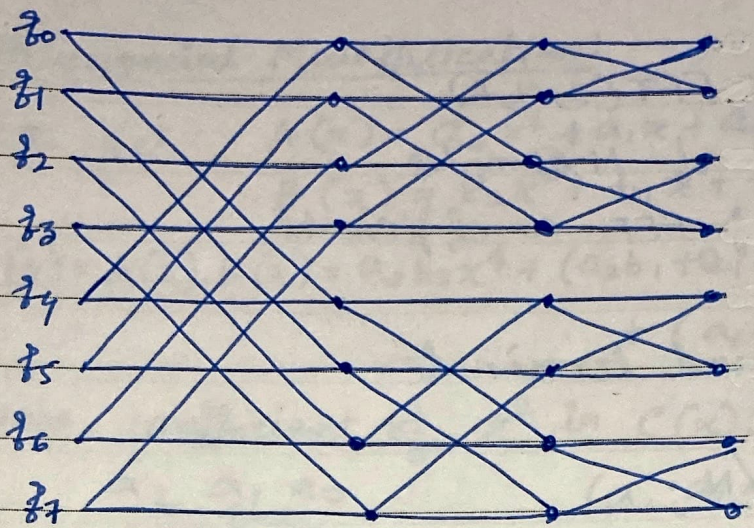








Signal Flow Graph (SFG) for Decimation in time (Bit reversed order of Time-samples.



Decimation in Frequency

0 1 2 3 4 5 6 7

↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓

f0 f4 f2 f6 f1 f5 f3 f7

Original Position in memory		Data Value shifted
0 = 000	⟷	000 (0)
1 = 001	⟷	100 (4)
2 = 010	⟷	010 (2)
3 = 011	⟷	110 (6)
4 = 100	⟷	001 (1)
5 = 101	⟷	101 (5)
6 = 110	⟷	011 (3)
7 = 111	⟷	111 (7)

Bit reversed order (BRO)

TATA

Procedure Recursive_FFT(S, N, A)

Input: N, Array S of N samples

Output: Array A of FFT co-efficients

if $N = 1$ then

Set $A \leftarrow S$ and terminate.

else

{

$W \leftarrow e^{-2\pi j/N}$;

$p \leftarrow 1$;

$X \leftarrow (S_0, S_2, \cdots, S_{N-2})$;

$Y \leftarrow (S_1, S_3, \cdots, S_{N-1})$;

Recursive_FFT(X, N/2, B);

Recursive_FFT(Y, N/2, C);

for $r = 0$ to $(N/2 - 1)$ do

{

term $\leftarrow p.C[r]$;

$A[r] \leftarrow B[r] + term$;

$A[r + \frac{N}{2}] \leftarrow B[r] - term$;

$p \leftarrow p.W$;

}

}

Polynomial Multiplication:

Ex. Let $A(x) = a_2x^2 + a_1x + a_0$

$B(x) = b_2x^2 + b_1x + b_0$

$C(x) = A(x).B(x) = a_2b_2x^4 + (a_2b_1 + a_1b_2)x^3 + (a_2b_0 + a_1b_1 + a_0b_2)x^2 + (a_1b_0 + a_0b_1)x + a_0b_0$

Now, coefficient of x^2 in $C(x)$: $a_2b_0 + a_1b_1 + a_0b_2$

$a_2\ a_1\ a_0$
$b_2\ b_1\ b_0$

$(a_2\ a_1\ a_0) \circ (b_2\ b_1\ b_0)$ ← convolution operation

Coefficient of x^3 in $C(x)$:

$0 \leftarrow a_3\ a_2\ a_1\ a_0$ $\quad a_3 = 0$ in $A(x)$ as there is no term x^3.

$b_0\ b_1\ b_2\ b_3$ $\quad$ and $b_3 = 0$ in $B(x)$ for same reason.

$\downarrow 0$

$a_2b_1 + a_1b_2$

coefficient of x^4 in $C(x)$:

$0 \leftarrow a_4\ a_3\ a_2\ a_1\ a_0$ $\quad a_3, a_4, b_3$ & b_4 are zero.

$b_0\ b_1\ b_2\ b_3^{\to 0}\ b_4^{\to 0}$

a_2b_2

Similarly we can compute the coefficients of x^1 & x^0 in $C(x)$. In general, to find the coefficient of x^i in $C(x)$ we compute as:

$(a_i\ a_{i-1}\ a_{i-2} \cdots a_1\ a_0) \circ (b_i\ b_{i-1}\ b_{i-2} \cdots b_1\ b_0)$

↓ convolution

$a_i\ a_{i-1}\ a_{i-2} \cdots a_1\ a_0$

$b_0\ b_1\ b_2 \cdots b_{i-1}\ b_i$

$a_ib_0 + a_{i-1}b_1 + a_{i-2}b_2 + \cdots + a_1b_{i-1} + a_0b_i$

Note: $a_i = b_i = 0$ for $n < i \leq 2n$

Because, $A(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_0$

and $B(x) = b_n x^n + b_{n-1} x^{n-1} + \cdots + b_0$

The number of product terms in computing the coefficient c_i of x^i in $C(x)$ is $(i+1)$ for $0 \leq i \leq n$ and $(2n-i+1)$ for $n < i \leq 2n$. So the total number of multiplications involved in computing all the coefficients of the product polynomial $C(x)$ is:

$C(x) = c_{2n} x^{2n} + c_{2n-1} x^{2n-1} + \cdots + c_1 x + c_0$

is $= \underbrace{1 + 2 + 3 + \cdots + n + (n+1)} + \underbrace{n + (n-1) + \cdots + 2 + 1}$

$= \frac{(n+1)(n+2)}{2} + \frac{n(n+1)}{2}$

$= \frac{(n+1)}{2}(n+2+n) = (n+1)(n+1) = (n+1)^2 = O(n^2)$

Observation 1:

Now, let's think about the coefficients $a_0, a_1, a_2, \cdots, a_n$ of $A(x)$. If we compute the FFT of these $(n+1)$ coefficient values, then we would get the FFT coefficients as

$$A_r = \sum_{k=0}^{n} a_k W^{rk} \quad \text{where } W = e^{-2\pi j/(n+1)}$$

$r = 0, 1, \cdots, n+1$

$$\therefore A_0 = \sum_{k=0}^{n} a_k \cdot W^{0.k} = a_0 + a_1 + a_2 + \cdots + a_n = A(x)$$

at $x = 1 = W^0$

$$\therefore A_1 = \sum_{k=0}^{n} a_k \cdot W^{1.k} = a_0 + a_1 W + a_2 W^2 + \cdots + a_n W^n = A(x)$$

at $x = W^1$

Likewise, $A_2 = \sum_{k=0}^{n} a_k \cdot W^{2.k} = a_0 + a_1 W^2 + a_2 W^4 + \cdots + a_n W^{2n} =$

value of $A(x)$ at $x = W^2$

That is, A_i = value of $A(x)$ at $x = w^i$

Observation 2:

If we compute the IFFT of $A_0, A_1, \ldots, A_n$, then we would get the coefficient $a_0, a_1, a_2, \ldots, a_n$ of the polynomial $A(x)$ back again. Specifically, if we could know the FFT coefficients $c_0, c_1, c_2, \ldots, c_{2n}$ of the polynomial coefficients $c_0, c_1, \ldots, c_{2n}$ of the product polynomial $C(x)$, then taking IFFT of $c_0, c_1, \ldots, c_{2n}$, we can get the coefficients $c_0, c_1, \ldots, c_{2n}$ of $C(x)$.

Observation 3:

Given the two polynomials $A(x)$ and $B(x)$, the value of the product polynomial $C(x) = A(x) \cdot B(x)$ at any value of $x = \alpha$, say, can be obtained by multiplying the value $A(\alpha)$ with $B(\alpha)$, $C(\alpha) = A(\alpha) \cdot B(\alpha)$. That is, if we evaluate the values of both $A(x)$ and $B(x)$ at $x = \alpha$.

Since c_0 = value of $C(x)$ at $x = w^0$, then c_0 is equal to product of the values of $A(x)$ and $B(x)$ at $x = w^0$, i.e., A_0 and B_0. $\Rightarrow c_0 = A_0 \cdot B_0$.

Similarly, c_1 = value of $C(x)$ at $x = w^1$

= product of values of $A(x)$ & $B(x)$ at $x = w^1$

$= A_1 \cdot B_1$

$\therefore c_2 = A_2 \cdot B_2$

$\vdots$

$c_i = A_i \cdot B_i$

Note that A_i and B_i are obtained from the FFT values

on $(a_0, a_1, \cdots, a_n)$ and $(b_0, b_1, \cdots, b_n)$, respectively. But C_i needs to be evaluated for all i, $0 \le i \le 2n$. Hence, to compute the FFT values of coefficients of $A(x)$ and $B(x)$, we have to append some zeros to the above sets of coefficient values, i.e., we find FFT values of $(a_0, a_1, a_2, \cdots, a_n, \underbrace{0, 0, \cdots, 0}_{n-1})$ and $(b_0, b_1, b_2, \cdots, b_n, \underbrace{0, 0, \cdots, 0}_{n-1})$.

In fact, FFT is defined for the number of samples as a power of 2. So, we append as many zeros as may be necessary to make the total number of coefficient values as the next power of 2.

Ex: $A(x) = a_4x^4 + a_3x^3 + a_2x^2 + a_1x + a_0$

$B(x) = b_4x^4 + b_3x^3 + b_2x^2 + b_1x + b_0$

$\therefore C(x) = c_8x^8 + c_7x^7 + c_6x^6 + c_5x^5 + c_4x^4 + c_3x^3 + c_2x^2 + c_1x + c_0$

Total number of coefficients in $C(x) = 9$. We make it 16 to have all the coefficient values from $c_0, c_1, c_2, \cdots$ to c_{15}. So, we append 11 0's after a_4 and 11 0's after b_4 so that the number of coefficients $A(x)$ and $B(x)$ each is equal to 16.

For $A(x) \Rightarrow (a_0, a_1, a_2, a_3, a_4, \underbrace{0, 0, \cdots, 0}_{11})$

For $B(x) \Rightarrow (b_0, b_1, b_2, b_3, b_4, \underbrace{0, 0, \cdots, 0}_{11})$

Algorithm for polynomial multiplication using FFT/IFFT:

Input: Coefficients $a_0, a_1, \cdots, a_n$ of $A(x)$

coefficients $b_0, b_1, \cdots, b_n$ of $B(x)$

Output: coefficients $c_0, c_1, c_2, \cdots, c_{2n}$ of $C(x) = A(x) \cdot B(x)$

Step 1: Append $2^k - (n+1)$ zeros after a_n to get the set $(a_0, a_1, \cdots, a_n, 0, 0, \cdots, 0)$ where 2^k is the least power of 2 greater than or equal to $2n$.

Step 2: Append $2^k - (n+1)$ zeros after b_n to get the the set $(b_0, b_1, b_2, \cdots, b_n, 0, 0, \cdots, 0)$ where 2^k is the least power of 2 greater than or equal to $2n$.

Step 3: Find FFT coefficients A_r's of $(a_0, a_1, \cdots, a_n, 0, 0, \cdots)$ $0 \leq i \leq 2^k - 1$.

Step 4: Find FFT coefficients B_r's of $(b_0, b_1, \cdots, b_n, 0, 0, \cdots, 0)$ $0 \leq i \leq 2^k - 1$.

Step 5: For each r, $0 \leq r \leq 2^k - 1$, compute the product $C_r = A_r \cdot B_r$

Step 6: Compute the IFFT values of $(C_0, C_1, \cdots, C_{2^k - 1})$ which will be the coefficients $c_0, c_1, c_2, \cdots, c_{2^k - 1}$ of $C(x)$.

Analysis:

Step 1 and Step 2 take constant time.

Step 3 takes $O(2^k \cdot k) = O(n \log n)$ time

Step 4 takes $O(n \log n)$ time.

Step 5 takes $O(2^k) = O(n)$ time

Step 6 takes $O(n \log n)$ time

So, the algorithm for polynomial multiplication takes $O(n \log n)$ time in constrast to $O(n^2)$ time by straightforward multiplication coefficients of $A(x)$ and $B(x)$.

TATA

Integer Exponentiation:

How to evaluate x^n, where x is some real number, n is an integer ≥ 0?

$$x^n = \underbrace{x \cdot x \cdot x \cdot \ldots \cdot x}_{(n-1) \text{ times multiplication.}}$$

Can we do it faster?

That means, in less than $(n-1)$ multiplications?

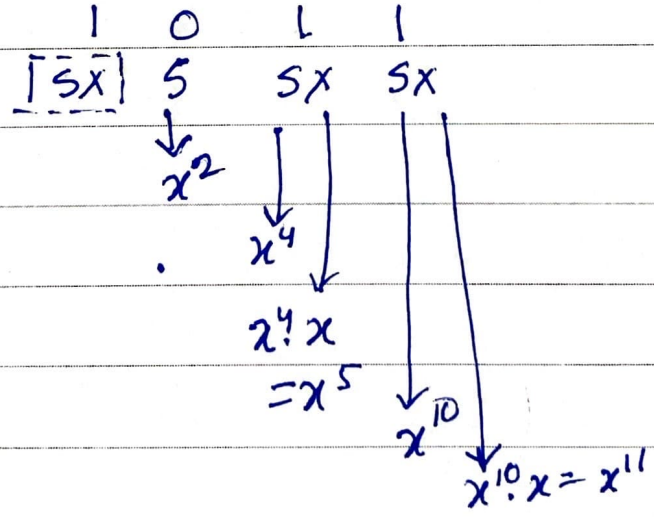
Let n be represented in binary number system, with the leftmost (most significant) bit as 1.

Ex: $n = 11 \Rightarrow$ 1 0 1 1 [Binary Representation of 11]

↓ ↓ ↓ ↓

SX S SX SX

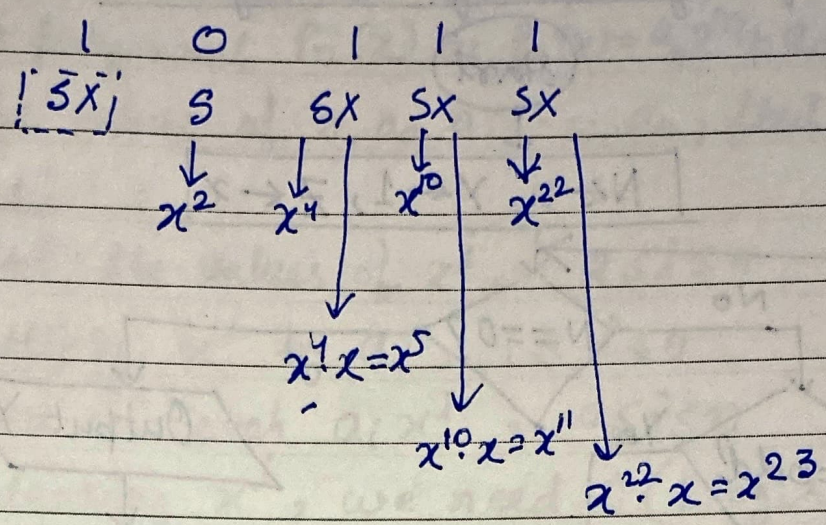
Write S below each 0 and SX below each 1; Discard the leftmost SX.



S → square the current value

X → multiply the value by x

Ex: $n = 23$



Left to right scan of the binary representation of n.
Total number of multiplication = number of bits to represent n in binary, i.e.: $\lfloor \log_2 n \rfloor + 1$

$\#\ S = \lfloor \log_2 n \rfloor + 1$

$\#\ X = r$, say (total # 1 in binary representation of n)

So, the total number of multiplications

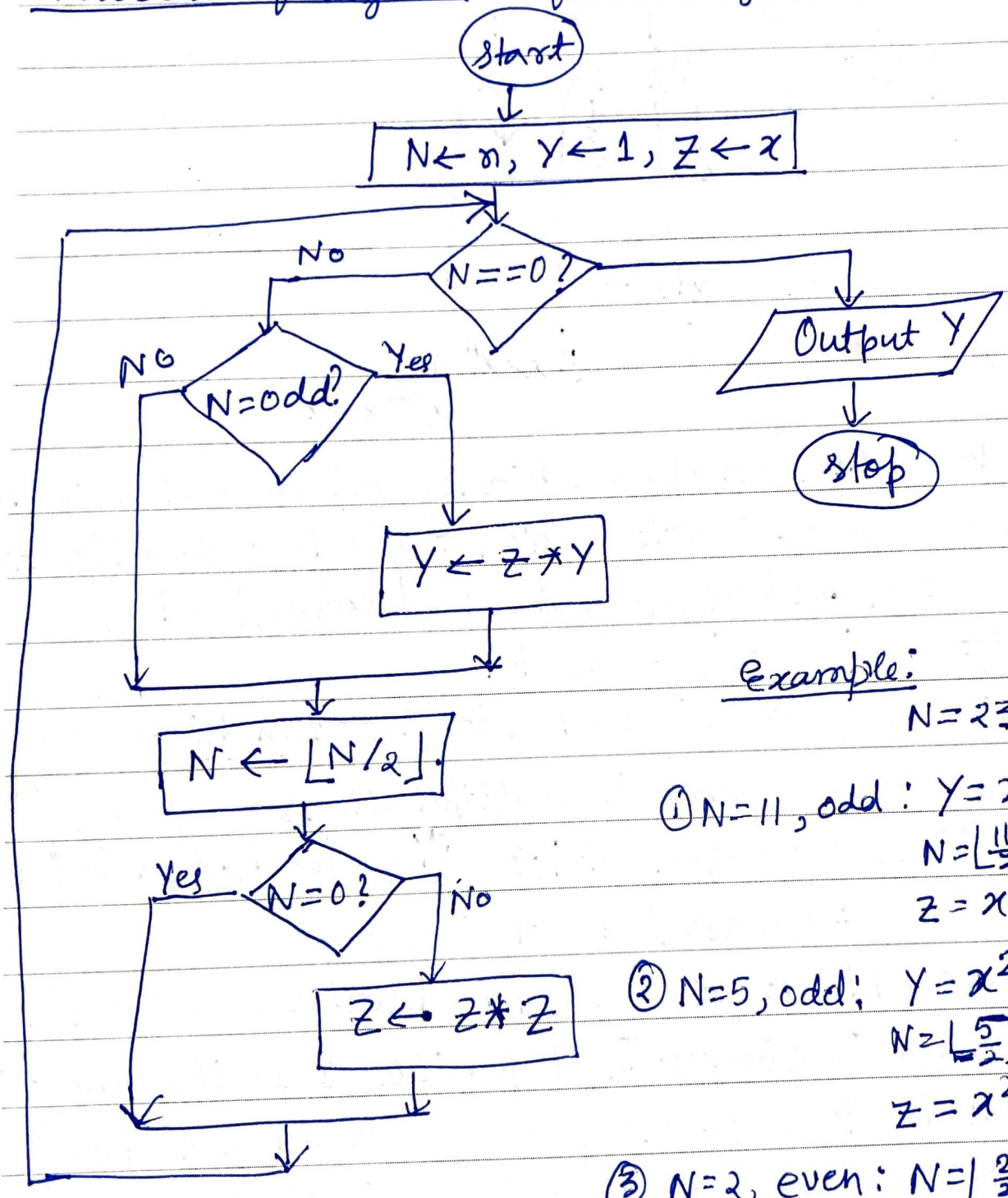
$T(n) = \lfloor \log_2 n \rfloor + 1 + r - 2$ [as first SX is ignored]

$= \lfloor \log_2 n \rfloor + r - 1$

However, left to right scan is possible only when n is represented in binary. Is it possible to write an algorithm without first converting n into its binary representation? Algorithm below can do it (by equivalently scanning the binary form of n) from right to left.

TATA

Flowchart of right to left binary scan:



Example:

$N = 23, Y = 1, Z = x$

① $N = 11$, odd : $Y = x \cdot 1 = x$

$N = \lfloor \frac{11}{2} \rfloor = 5$

$Z = x \cdot x = x^2$

② $N = 5$, odd; $Y = x^2 \cdot x = x^3$

$N = \lfloor \frac{5}{2} \rfloor = 2$

$Z = x^2 \cdot x^2 = x^4$

③ $N = 2$, even : $N = \lfloor \frac{2}{2} \rfloor = 1$

$Z = x^4 \cdot x^4 = x^8$

④ $N = 1$, odd : $Y = x^8 \cdot x^3 = x^{11}$

$N = \lfloor \frac{1}{2} \rfloor = 0$

Polynomial Evaluation:

Given a polynomial $P_n(x)$ as $P_n(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0$, and the values of $x, a_0, a_1, \cdots, a_n$, find the value of $P_n(x)$

Method 1:

Step 1. Find the values of x^i, $2 \le i \le n$

Step 2. Multiply x^i by a_i, $1 \le i \le n$

Step 3. Add all such $a_i x^i$, $0 \le i \le n$

For evaluating x^i, we need $O(\lfloor \log_2 i \rfloor)$ multiplications.

Thus, to evaluate all such x^i in step 1, we need $O\left(\sum_{i=2}^{n} \lfloor \log i \rfloor\right)$ multiplications. Steps 2 needs n multiplications. Step 3 needs n additions.

$$\sum_{i=1}^{n} \lfloor \log i \rfloor = \lfloor \log 1 \rfloor + \lfloor \log 2 \rfloor + \lfloor \log 3 \rfloor + \lfloor \log 4 \rfloor + \lfloor \log 5 \rfloor + \lfloor \log 6 \rfloor + \lfloor \log 7 \rfloor + \cdots$$

$$= 0 + 1 + 1 + 2 + 2 + 2 + 2 + \cdots$$

Assume that $n = 2^k - 1$.

$$\therefore \text{Then } \sum_{i=1}^{n} \lfloor \log i \rfloor = \lfloor \log 1 \rfloor + \lfloor \log 2 \rfloor + \lfloor \log 3 \rfloor + \cdots + \lfloor \log (2^k - 1) \rfloor$$

$$= 0 + \underbrace{1 + 1}_{2} + \underbrace{2 + 2 + 2 + 2}_{4} + \underbrace{3 + 3 + \cdots + 3}_{8} +$$

$$\cdots + \underbrace{(k-1) + (k-1) + \cdots + (k-1)}_{(2^{k-1} \text{ times})}$$

$$= 1 \cdot 2^1 + 2 \cdot 2^2 + 3 \cdot 2^3 + \cdots + (k-1) \cdot 2^{k-1}$$

Let $S = 1 \cdot 2^1 + 2 \cdot 2^2 + 3 \cdot 2^3 + \cdots + (k-1) \cdot 2^{k-1}$

$$2S = \quad 1 \cdot 2^2 + 2 \cdot 2^3 + \cdots + (k-2) \cdot 2^{k-1} + (k-1) \cdot 2^k$$

$$\therefore -S = 2^1 + 2^2 + 2^3 + \cdots + 2^{k-1} - (k-1) \cdot 2^k$$

$\therefore S = (k-1)\cdot 2^k - (2^1 + 2^2 + \cdots + 2^{k-1})$

$= (k-1)\cdot 2^k - (2^k - 2) = 2^k\cdot(k-2) + 2 = O(n \log n)$

Hence method 1: needs $O(n\log n + n) = O(n \log n)$ multiplications.

Method 2:

```
Step 1: P ← a0; i ← 1; term ← 1;
Step 2: if n > 0 then
            while i ≤ n do
            {   term ← term * x;
                P ← P + term * ai;
            }
Step 3: Print P;
```

Analysis: Step 1 needs constant time. Step 2 needs two multiplications and one addition in each iteration. Hence, the above method needs 2n multiplications and n additions.

Method 3:

Rewrite the polynomial $P_n(x)$ as $P_n(x) = a_n x^n +$

$P_n(x) = a_n x^n + a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + \cdots + a_2 x^2 + a_1 x + a_0$

$= (\ldots(((a_n x + a_{n-1})x + a_{n-2})x + a_{n-3})x + \cdots + a_1)x + a_0$

Thus, there will be $(n-1)$ opening parenthesis at the begining, and closing parenthesis are inserted after adding $a_{n-1}, a_{n-2}, a_{n-3}, \ldots, a_1$ successively.

Try with $P_2(x)$:

$P_2(x) = a_2x^2 + a_1x + a_0 = (a_2x + a_1)x + a_0$

Similarly, $P_3(x) = a_3x^3 + a_2x^2 + a_1x + a_0$

$= ((a_3x + a_2)x + a_1)x + a_0$

Algorithm:

step 1: $p \leftarrow a_n$; $i \leftarrow n$;

step 2: while $i \geq 1$ do

{ $p \leftarrow p * x + a_{i-1}$;

$i \leftarrow i - 1$;

}

step 3: print p;

Altogether, n multiplications are needed as compared to $O(n \log n)$ multiplications in method 1 and $2n$ multiplications in method 2.