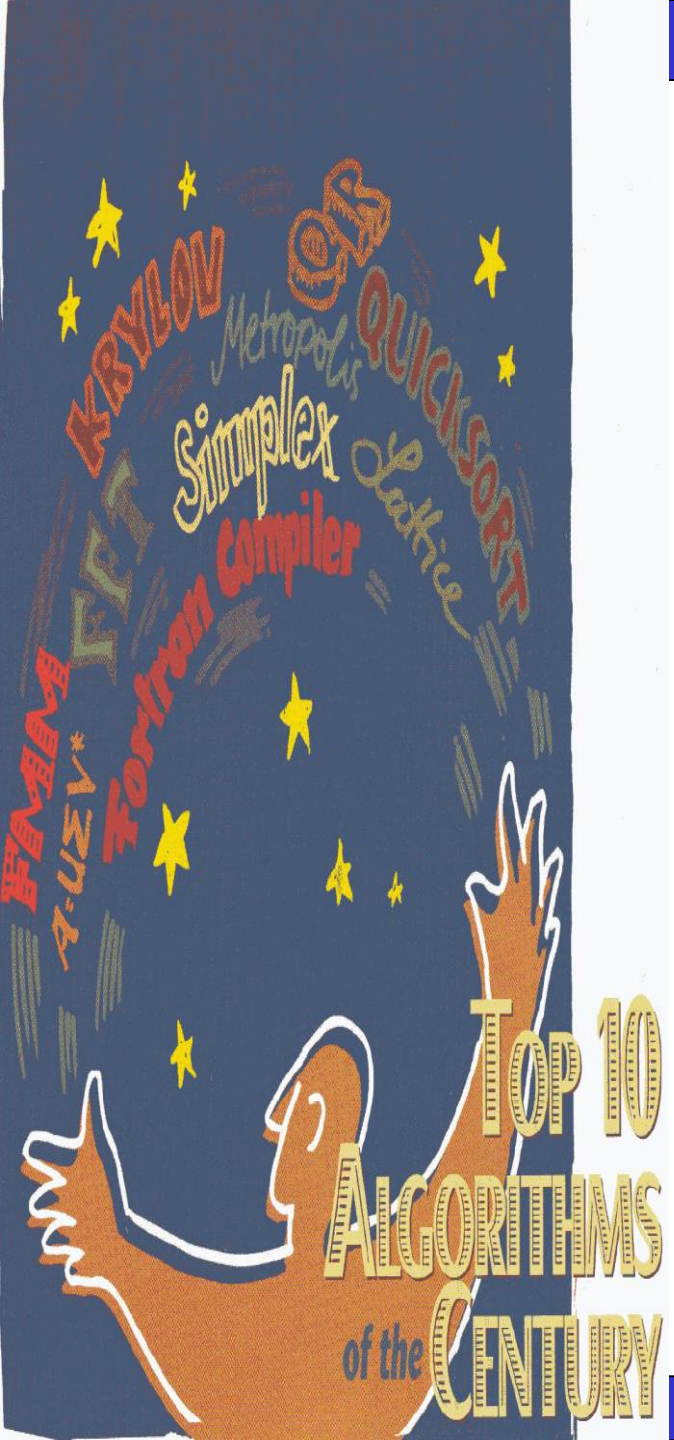


1. Metropolis Algorithm for Monte Carlo
2. Simplex Method for Linear Programming
3. Conjugate Gradients & Krylov Subspace Iteration Method for Solving Linear System
4. Matrix Factorizations (LU, Cholesky) for Solving Linear System and Least Squares
5. The Fortran Optimizing Compiler
6. QR/QZ/SVD Algos for Eigenvalue Computation
7. Quick Sort Algorithm
8. **Fast Fourier Transform (FFT)**
9. Fast Multipole Method
10. Integer Relation Detection

Efficient Software Implementations of FFT Algorithms

- **The practical applications of the DFT, such as digital signal processing, demand the utmost speed.**

- **Two Efficient Implementations of FFT Algorithms**
  - **Iterative FFT Algorithm**
  - **Butterfly Operation Algorithm**

1. An Iterative FFT Implementation

Let us assume that *n = N; Data Vector 'x = a'; DFT Vector 'X = y'*

- **Lines 10-13, RECURSIVE-FFT involves in computing the value $\omega_n^k y_k^{[1]}$ twice, change the loop to compute it only**

```
RECURSIVE-FFT(a)
 1  n ← length[a]
 2  if n = 1
 3     then return a
 4  ω_n ← e^{2πi/n}
 5  ω ← 1
 6  a^[0] ← (a_0, a_2, ..., a_{n-2})
 7  a^[1] ← (a_1, a_3, ..., a_{n-1})
 8  y^[0] ← RECURSIVE-FFT(a^[0])
 9  y^[1] ← RECURSIVE-FFT(a^[1])
10  for k ← 0 to n/2 − 1
11        do y_k ← y_k^[0] + ω y_k^[1]
12           y_{k+(n/2)} ← y_k^[0] − ω y_k^[1]
13           ω ← ω ω_n
14  return y
```

// *n* is a power of 2
// Lines 2-3 represent the basis of recursion
// Lines 6-7 defines the coefficient vectors for $a^{[0]}$ and $a^{[1]}$
// Lines 4, 5, and 13 guarantee that ω is updated properly so that whenever Lines 11-12 are executed, we have $\omega = \omega_n^k$
// Lines 8-9 perform the recursive $DFT_{n/2}$ computations.
// Lines 11-12 combine the results of the recursive $DFT_{n/2}$ computations.
// *y* is a column vector

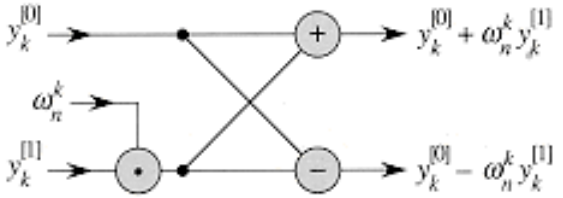
Butterfly Operation:
adding and subtracting *t* from $y_k^{[0]}$

```
for k ← 0 to n/2 − 1
    do t ← ω y_k^[1]
       y_k ← y_k^[0] + t
       y_{k+(n/2)} ← y_k^[0] − t
       ω ← ω ω_n
```

1. An Iterative FFT Implementation (Contd…)

Let us assume that *n = N; Data Vector 'x = a'; DFT Vector 'X = y'*

```
RECURSIVE-FFT(a)
 1  n ← length[a]
 2  if n = 1
 3      then return a
 4  ω_n ← e^{2πi/n}
 5  ω ← 1
 6  a^[0] ← (a_0, a_2, ..., a_{n-2})
 7  a^[1] ← (a_1, a_3, ..., a_{n-1})
 8  y^[0] ← RECURSIVE-FFT(a^[0])
 9  y^[1] ← RECURSIVE-FFT(a^[1])
10  for k ← 0 to n/2 − 1
11        do y_k ← y_k^[0] + ω y_k^[1]
12           y_{k+(n/2)} ← y_k^[0] − ω y_k^[1]
13           ω ← ω ω_n
14  return y
```



y_k

$(k = 0,1,\cdots,n/2-1)$

$y_{k+n/2}$

Butterfly Operation: adding and subtracting *t* from $y_k^{[0]}$

```
for k ← 0 to n/2 − 1
    do t ← ω y_k^[1]
       y_k ← y_k^[0] + t
       y_{k+(n/2)} ← y_k^[0] − t
       ω ← ω ω_n
```

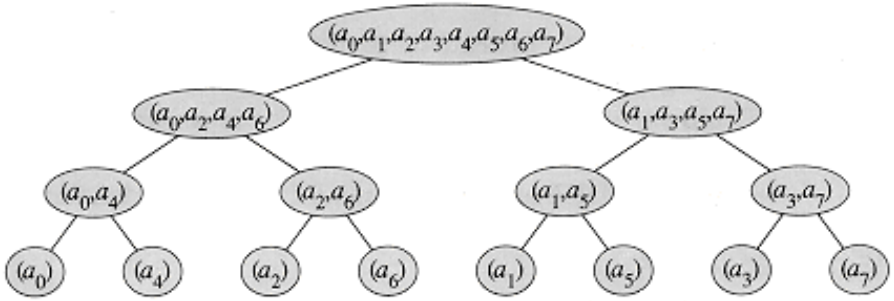
1. An Iterative FFT Implementation (Contd…)

Let us assume that *n = N; Data Vector 'x = a'; DFT Vector 'X = y'*

- **Make the FFT Algorithm Iterative rather than Recursive in Structure.**

```
RECURSIVE-FFT(a)
 1  n ← length[a]
 2  if n = 1
 3      then return a
 4  ω_n ← e^{2πi/n}
 5  ω ← 1
 6  a^[0] ← (a_0, a_2, ..., a_{n-2})
 7  a^[1] ← (a_1, a_3, ..., a_{n-1})
 8  y^[0] ← RECURSIVE-FFT(a^[0])
 9  y^[1] ← RECURSIVE-FFT(a^[1])
10  for k ← 0 to n/2 − 1
11        do y_k ← y_k^[0] + ω y_k^[1]
12           y_{k+(n/2)} ← y_k^[0] − ω y_k^[1]
13           ω ← ω ω_n
14  return y
```

Recursive calls of RECURSIVE-FFT in a Tree Structure.




```
for k ← 0 to n/2 − 1
    do t ← ω y_k^[1]
       y_k ← y_k^[0] + t
       y_{k+(n/2)} ← y_k^[0] − t
       ω ← ω ω_n
```

1. An Iterative FFT Implementation (Contd…)

Let us assume that *n = N; Data Vector 'x = a'; DFT Vector 'X = y'*

- **Iterative FFT Algorithm**

Arrange the elements of *a* into the order in which they appear in the leaves,

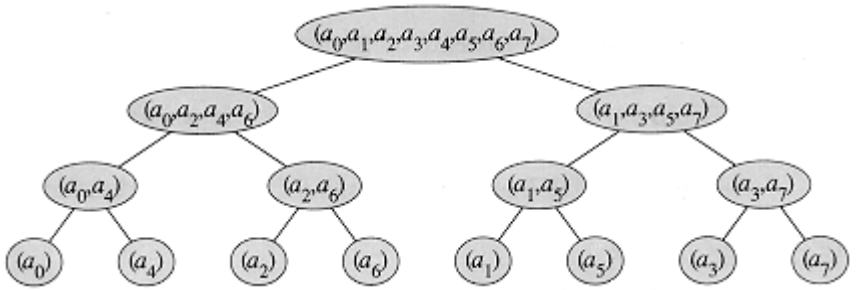
1) compute the DFT of each pair using one butterfly operation, replace the pair with its DFT, the vector then holds (*n*/2)th 2-element DFT's;

2) take the (*n*/2)th DFT's in pairs, compute the DFT of the four vector elements, 2 butterfly operations, the new vector holds (*n*/4)th

4-element DFT's;

3) continue in this manner.

Recursive calls of RECURSIVE-FFT in a Tree Structure.



for $k \leftarrow 0$ **to** $n/2 - 1$
  do $t \leftarrow \omega\, y_k^{[1]}$
  $y_k \leftarrow y_k^{[0]} + t$
  $y_{k+(n/2)} \leftarrow y_k^{[0]} - t$
  $\omega \leftarrow \omega\, \omega_n$

1. An Iterative FFT Implementation (Contd…)

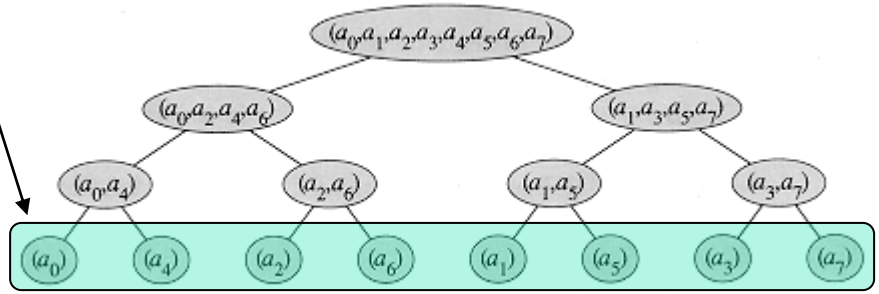
Let us assume that *n = N; Data Vector 'x = a'; DFT Vector 'X = y'*

- **Iterative FFT Algorithm**

initial $A[0..n-1]$

```
ITERATIVE-FFT (a)
1  BIT-REVERSE-COPY (a, A)
2  n←length[a]  // n is a power of 2.
3  for s←1 to lg n
4      do m←2^s
5         ω_m←e^(2πi/m)
6         for k←0 to n-1 by m
7             do ω←1
8                for j←0 to m/2 -1
9                    do t←ωA[k+j+m/2]
10                      u←A[k+j]
11                      A[k+j]←u+t
12                      A[k+j+m/2]←u-t
13                      ω←ω . ω_m
```

Recursive calls of RECURSIVE-FFT in a Tree Structure.



$s=1$	$s=2$	$s=3$
$m=2^s=2$	$m=4$	$m=8$
$k=0, 2,.., n-1$	$k=0, 4,.., n-1$	$k=0, 8,.., n-1$
$j=0$	$j=0, 1$	$j=0, 1, 2, 3$
$A[0]$ $A[1]$	$A[0]$ $A[2]$	$A[0]$ $A[4]$
$A[2]$ $A[3]$	$A[1]$ $A[3]$	$A[1]$ $A[5]$
…	…	…

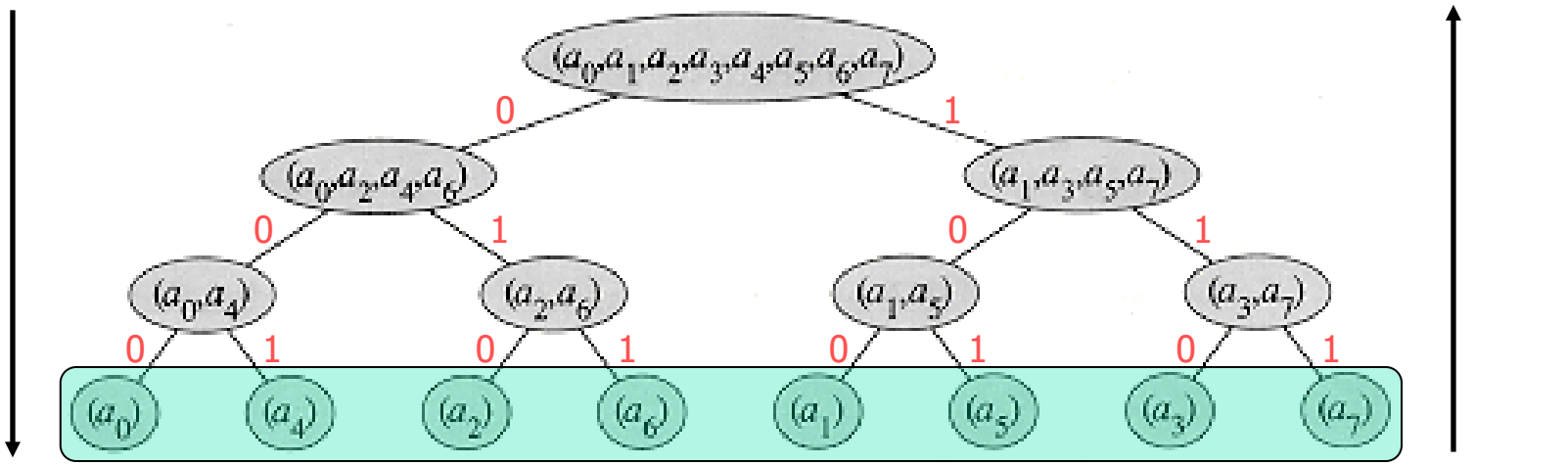
1. An Iterative FFT Implementation (Contd…)

Let us assume that *n = N; Data Vector 'x = a'; DFT Vector 'X = y'*

- **Bit-Reversal Permutation**

ITERATIVE-FFT (*a*)
1 BIT-REVERSE-COPY (*a, A*)
…

Original order



Bit-Reversal

0	1	2	3	4	5	6	7
000	001	010	011	100	101	110	111
000	100	010	110	001	101	011	111
0	4	2	6	1	5	3	7

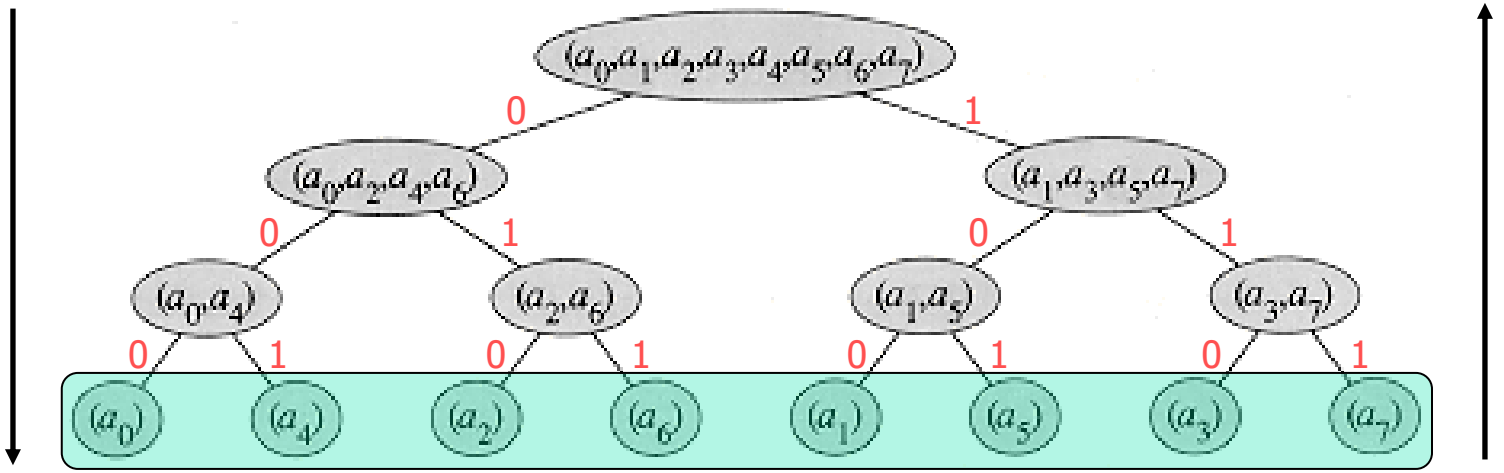
1. An Iterative FFT Implementation (Contd…)

Let us assume that *n = N; Data Vector 'x = a'; DFT Vector 'X = y'*

- **Bit-Reversal Permutation**

```
ITERATIVE-FFT (a)
1  BIT-REVERSE-COPY (a, A)
…
```

Original order



Bit-reversal

0	1	2	3	4	5	6	7
000	001	010	011	100	101	110	111
000	100	010	110	001	101	011	111
0	4	2	6	1	5	3	7

```
BIT-REVERSE-COPY(a, A)  //Θ(nlgn)
1  n←length[a]
2  for k←0 to n–1                 //Θ(n)
3       do A[rev(k)]←a_k          //Θ(lgn)
```

1. An Iterative FFT Implementation (Contd…)

Let us assume that *n = N; Data Vector 'x = a'; DFT Vector 'X = y'*

- **Iterative FFT Algorithm**

```
ITERATIVE-FFT (a)
1  BIT-REVERSE-COPY (a, A)                    //Θ(nlgn)
2  n←length[a]  // n is a power of 2.
3  for s←1 to lg n                            // lgn
4      do m←2^s
5        ω_m←e^(2πi/m)
6        for k←0 to n-1 by m                  // n/m=n/2^s
7            do ω←1
8                for j←0 to m/2 -1            // m/2=2^s/2=2^(s-1)
9                    do t←ωA[k+j+m/2]
10                      u←A[k+j]
11                      A[k+j]←u+t
12                      A[k+j+m/2]←u-t
13                     ω←ω · ω_m
```

$$L(n) = \sum_{s=1}^{\lg n} \frac{n}{2^s} 2^{s-1}$$

$$= \sum_{s=1}^{\lg n} \frac{n}{2}$$

$$= \Theta(n \lg n)$$

2. A Butterfly Operation

$$\text{DFT}, \quad \begin{pmatrix} y_0 \\ y_1 \\ \vdots \\ y_{n-1} \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega_n^1 & \omega_n^2 & \cdots & \omega_n^{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega_n^{(n-1)\cdot 1} & \omega_n^{(n-1)\cdot 2} & \cdots & \omega_n^{(n-1)\cdot(n-1)} \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_{n-1} \end{pmatrix}, \quad \Theta(n^2)$$

$$\text{FFT}, \quad \Theta(n \lg n) \quad ?$$

$$\text{FFT}, \quad \begin{pmatrix} y_0 \\ y_4 \\ y_2 \\ y_6 \\ y_1 \\ y_5 \\ y_3 \\ y_7 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega_n^{4\cdot 1} & \omega_n^{4\cdot 2} & \cdots & \omega_n^{4\cdot(n-1)} \\ 1 & \omega_n^{2\cdot 1} & \omega_n^{2\cdot 2} & \cdots & \omega_n^{2\cdot(n-1)} \\ 1 & \omega_n^{6\cdot 1} & \omega_n^{6\cdot 2} & \cdots & \omega_n^{6\cdot(n-1)} \\ 1 & \omega_n^{1\cdot 1} & \omega_n^{1\cdot 2} & \cdots & \omega_n^{1\cdot(n-1)} \\ 1 & \omega_n^{5\cdot 1} & \omega_n^{5\cdot 2} & \cdots & \omega_n^{5\cdot(n-1)} \\ 1 & \omega_n^{3\cdot 1} & \omega_n^{3\cdot 2} & \cdots & \omega_n^{3\cdot(n-1)} \\ 1 & \omega_n^{7\cdot 1} & \omega_n^{7\cdot 2} & \cdots & \omega_n^{7\cdot(n-1)} \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \\ a_6 \\ a_7 \end{pmatrix} = W_1 W_2 W_3 a, \quad \Theta(n \lg n)$$

$W = W_1 W_2 W_3$

2. A Butterfly Operation (Contd…)

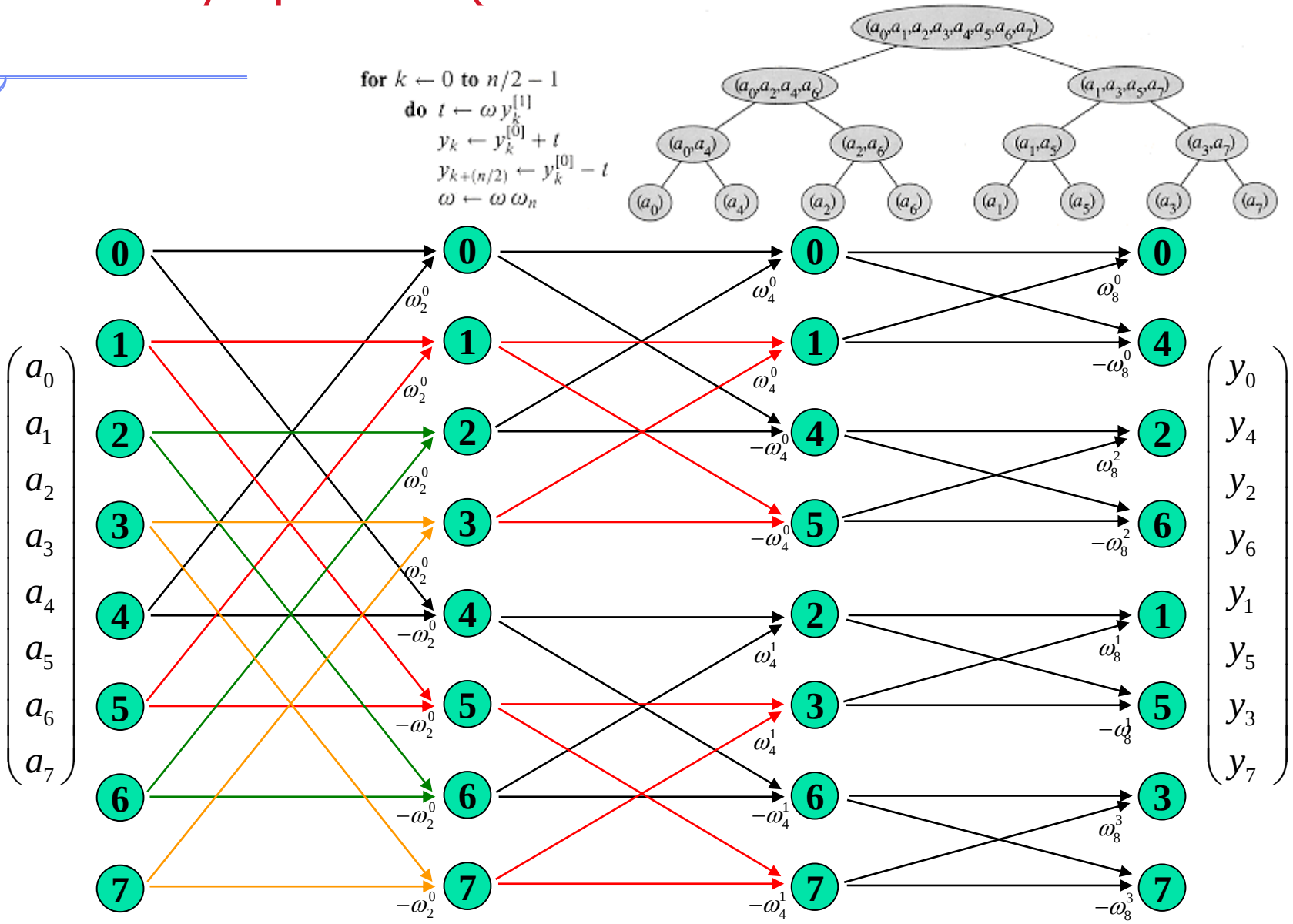
$$\omega = e^{2\pi i/8}$$

$$\begin{pmatrix} y_0 \\ y_4 \\ y_2 \\ y_6 \\ y_1 \\ y_5 \\ y_3 \\ y_7 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega_n^{4\cdot1} & \omega_n^{4\cdot2} & \cdots & \omega_n^{4\cdot(n-1)} \\ 1 & \omega_n^{2\cdot1} & \omega_n^{2\cdot2} & \cdots & \omega_n^{2\cdot(n-1)} \\ 1 & \omega_n^{6\cdot1} & \omega_n^{6\cdot2} & \cdots & \omega_n^{6\cdot(n-1)} \\ 1 & \omega_n^{1\cdot1} & \omega_n^{1\cdot2} & \cdots & \omega_n^{1\cdot(n-1)} \\ 1 & \omega_n^{5\cdot1} & \omega_n^{5\cdot2} & \cdots & \omega_n^{5\cdot(n-1)} \\ 1 & \omega_n^{3\cdot1} & \omega_n^{3\cdot2} & \cdots & \omega_n^{3\cdot(n-1)} \\ 1 & \omega_n^{7\cdot1} & \omega_n^{7\cdot2} & \cdots & \omega_n^{7\cdot(n-1)} \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \\ a_6 \\ a_7 \end{pmatrix}$$

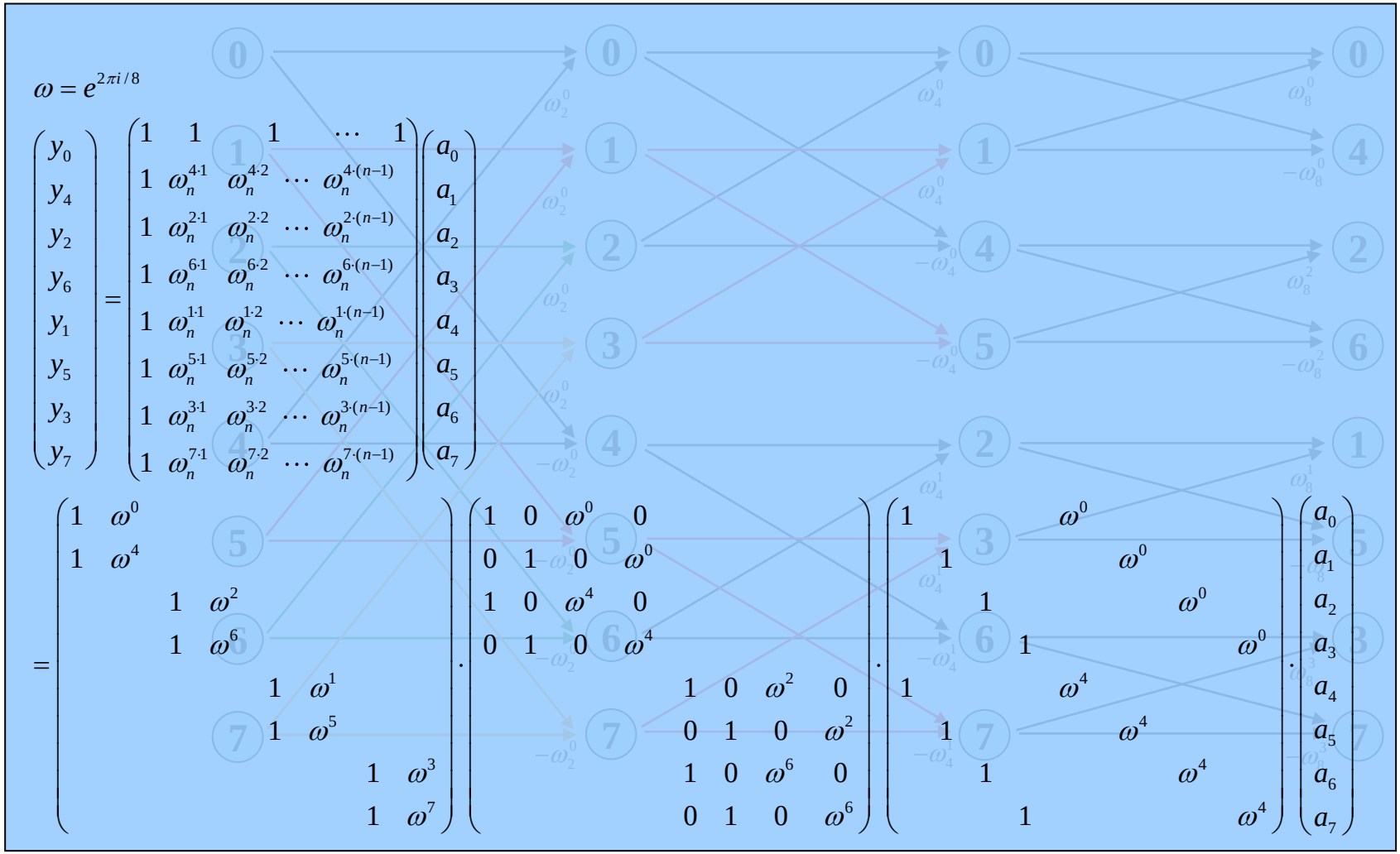
$$= \begin{pmatrix} 1 & \omega^0 & & & & & & \\ 1 & \omega^4 & & & & & & \\ & & 1 & \omega^2 & & & & \\ & & 1 & \omega^6 & & & & \\ & & & & 1 & \omega^1 & & \\ & & & & 1 & \omega^5 & & \\ & & & & & & 1 & \omega^3 \\ & & & & & & 1 & \omega^7 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & \omega^0 & 0 & & & & \\ 0 & 1 & 0 & \omega^0 & & & & \\ 1 & 0 & \omega^4 & 0 & & & & \\ 0 & 1 & 0 & \omega^4 & & & & \\ & & & & 1 & 0 & \omega^2 & 0 \\ & & & & 0 & 1 & 0 & \omega^2 \\ & & & & 1 & 0 & \omega^6 & 0 \\ & & & & 0 & 1 & 0 & \omega^6 \end{pmatrix} \cdot \begin{pmatrix} 1 & & & & \omega^0 & & & \\ & 1 & & & & \omega^0 & & \\ & & 1 & & & & \omega^0 & \\ & & & 1 & & & & \omega^0 \\ 1 & & & & \omega^4 & & & \\ & 1 & & & & \omega^4 & & \\ & & 1 & & & & \omega^4 & \\ & & & 1 & & & & \omega^4 \end{pmatrix} \cdot \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \\ a_6 \\ a_7 \end{pmatrix}$$

W_1 W_2 W_3

2. A Butterfly Operation (Contd…)



2. A Butterfly Operation (Contd…)



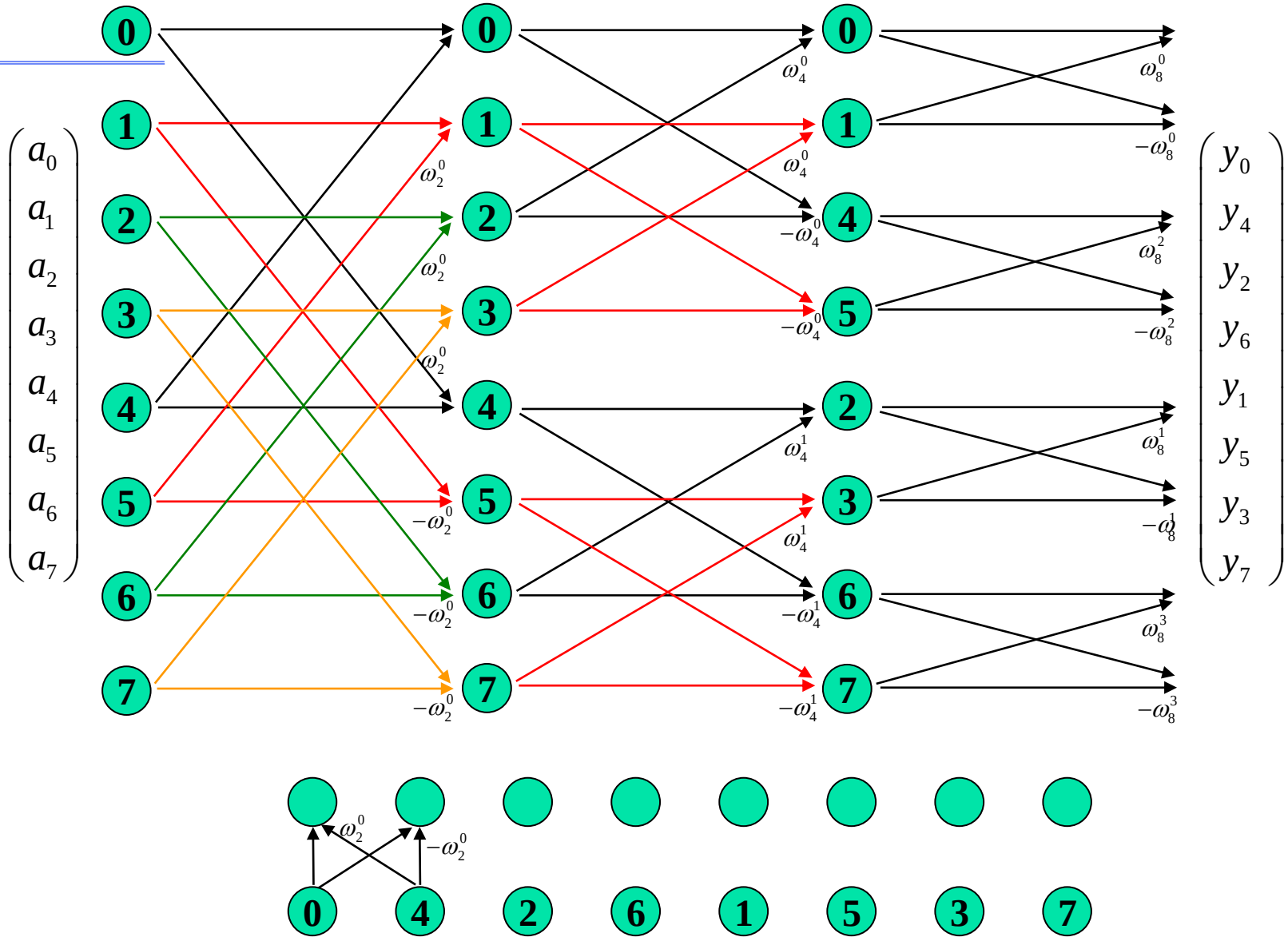
2. A Butterfly Operation (Contd…)

$$\omega = e^{2\pi i/8}$$

$$\begin{pmatrix} y_0 \\ y_4 \\ y_2 \\ y_6 \\ y_1 \\ y_5 \\ y_3 \\ y_7 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega_n^{4\cdot 1} & \omega_n^{4\cdot 2} & \cdots & \omega_n^{4\cdot(n-1)} \\ 1 & \omega_n^{2\cdot 1} & \omega_n^{2\cdot 2} & \cdots & \omega_n^{2\cdot(n-1)} \\ 1 & \omega_n^{6\cdot 1} & \omega_n^{6\cdot 2} & \cdots & \omega_n^{6\cdot(n-1)} \\ 1 & \omega_n^{1\cdot 1} & \omega_n^{1\cdot 2} & \cdots & \omega_n^{1\cdot(n-1)} \\ 1 & \omega_n^{5\cdot 1} & \omega_n^{5\cdot 2} & \cdots & \omega_n^{5\cdot(n-1)} \\ 1 & \omega_n^{3\cdot 1} & \omega_n^{3\cdot 2} & \cdots & \omega_n^{3\cdot(n-1)} \\ 1 & \omega_n^{7\cdot 1} & \omega_n^{7\cdot 2} & \cdots & \omega_n^{7\cdot(n-1)} \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \\ a_6 \\ a_7 \end{pmatrix}$$

$$= \begin{pmatrix} 1 & \omega^0 & & & & & & \\ 1 & \omega^4 & & & & & & \\ & & 1 & \omega^2 & & & & \\ & & 1 & \omega^6 & & & & \\ & & & & 1 & \omega^1 & & \\ & & & & 1 & \omega^5 & & \\ & & & & & & 1 & \omega^3 \\ & & & & & & 1 & \omega^7 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & \omega^0 & 0 & & & & \\ 0 & 1 & 0 & \omega^0 & & & & \\ 1 & 0 & \omega^4 & 0 & & & & \\ 0 & 1 & 0 & \omega^4 & & & & \\ & & & & 1 & 0 & \omega^2 & 0 \\ & & & & 0 & 1 & 0 & \omega^2 \\ & & & & 1 & 0 & \omega^6 & 0 \\ & & & & 0 & 1 & 0 & \omega^6 \end{pmatrix} \cdot \begin{pmatrix} 1 & & & & \omega^0 & & & \\ & 1 & & & & \omega^0 & & \\ & & 1 & & & & \omega^0 & \\ & & & 1 & & & & \omega^0 \\ 1 & & & & \omega^4 & & & \\ & 1 & & & & \omega^4 & & \\ & & 1 & & & & \omega^4 & \\ & & & 1 & & & & \omega^4 \end{pmatrix} \cdot \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \\ a_6 \\ a_7 \end{pmatrix}$$

3. A New Butterfly Operation?



3. A New Butterfly Operation? (Contd…)

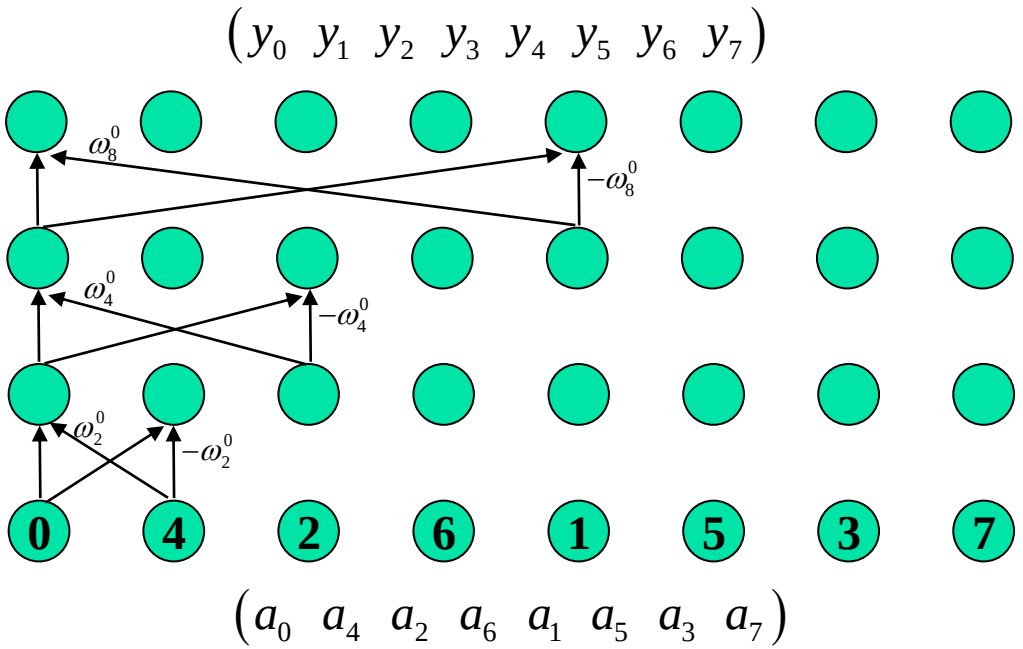
for $k \leftarrow 0$ **to** $n/2 - 1$

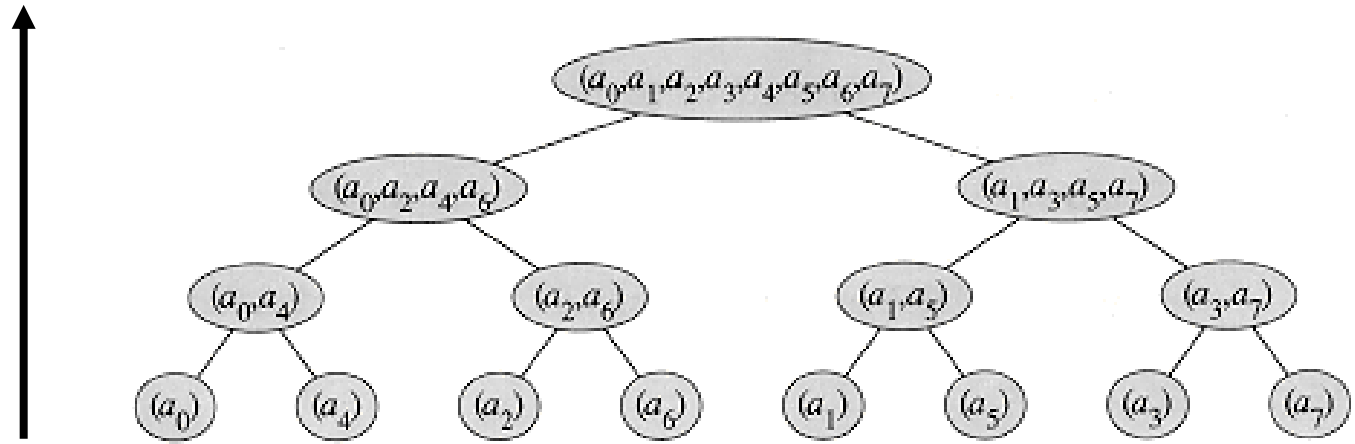
do $t \leftarrow \omega\, y_k^{[1]}$

$y_k \leftarrow y_k^{[0]} + t$

$y_{k+(n/2)} \leftarrow y_k^{[0]} - t$

$\omega \leftarrow \omega\, \omega_n$





Comments

- The FFT algorithm is very useful when computations are carried out in the frequency domain.
- FFT is much faster than a regular DFT algorithm
- FFT is more precise with less errors due to round off.
- The timed coding examples further support this claim and demonstrate how to code the algorithm.
- The Radix-2 FFT isn't the fastest but it uses a less complex addressing and twiddle factor routine
- Hence, FFT is better and more efficient than DFT.