FAST FOURIER TRANSFORM

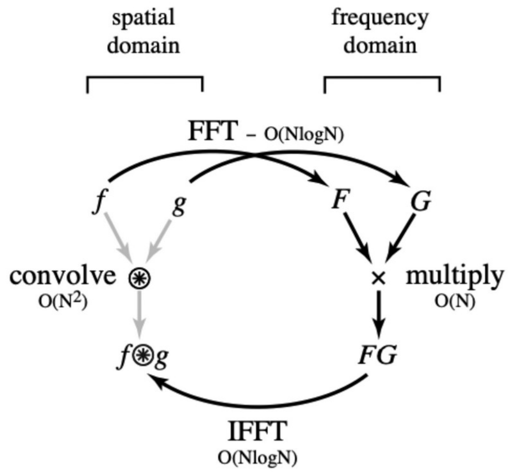
PhD in «Mathematics, Physics and application for engineering», 38° cycle
Michele Di Giovanni

FAST FOURIER TRANSFORM – THE REASON BEHIND

The Fourier transform is widely used in several engineering—related applications.
Those applications are often implemented through Digital computers, which means that the analysis is done by the **DFT.**

The widespread use of the Fourier Transform led to the need to find **a practical, and fast way, to compute the DFT's coefficients**.

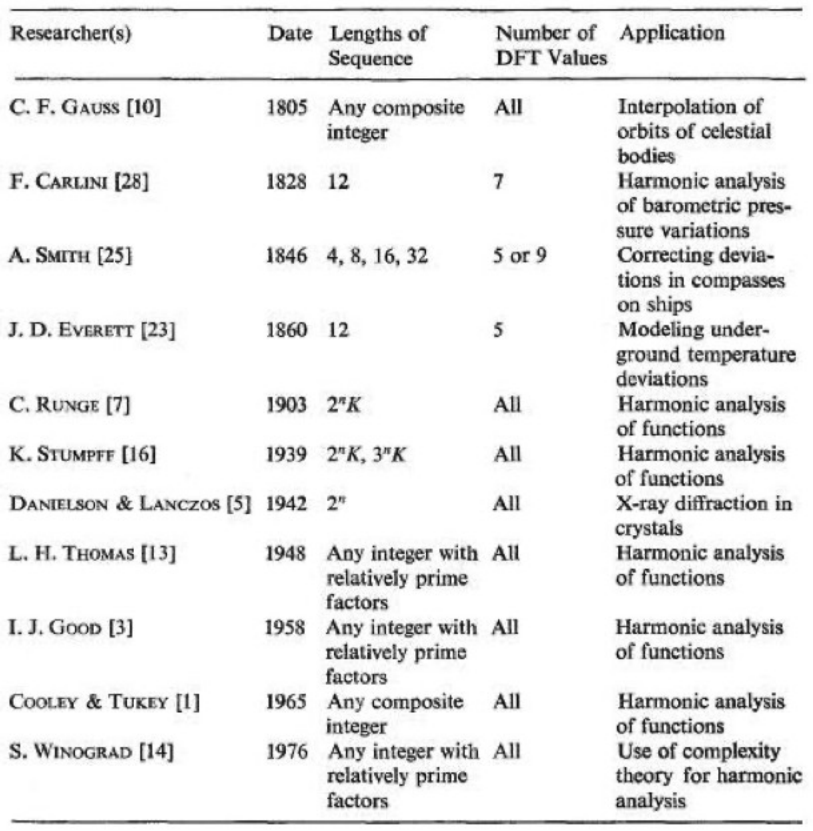
Why should we want to compute the DFT? A practical example: **convolution and filtering**



FAST FOURIER TRANSFORM – HISTORICAL INTRODUCTION

Between 1805 and 1965 several scientist developed efficient methods to calculate the DFT, applied in different fields of applied math.

Cooley and ***Turkey*** developed an algorithm in the context of time-series analysis, that is called **Fast Fourier Transform**



Researcher(s)	Date	Lengths of Sequence	Number of DFT Values	Application
C. F. Gauss [10]	1805	Any composite integer	All	Interpolation of orbits of celestial bodies
F. Carlini [28]	1828	12	7	Harmonic analysis of barometric pressure variations
A. Smith [25]	1846	4, 8, 16, 32	5 or 9	Correcting deviations in compasses on ships
J. D. Everett [23]	1860	12	5	Modeling underground temperature deviations
C. Runge [7]	1903	2^nK	All	Harmonic analysis of functions
K. Stumpff [16]	1939	2^nK, 3^nK	All	Harmonic analysis of functions
Danielson & Lanczos [5]	1942	2^n	All	X-ray diffraction in crystals
L. H. Thomas [13]	1948	Any integer with relatively prime factors	All	Harmonic analysis of functions
I. J. Good [3]	1958	Any integer with relatively prime factors	All	Harmonic analysis of functions
Cooley & Tukey [1]	1965	Any composite integer	All	Harmonic analysis of functions
S. Winograd [14]	1976	Any integer with relatively prime factors	All	Use of complexity theory for harmonic analysis

FAST FOURIER TRANSFORM – SIGNAL REPRESENTATION

A periodic signal of periodicity N is representable as a N-elements vector.

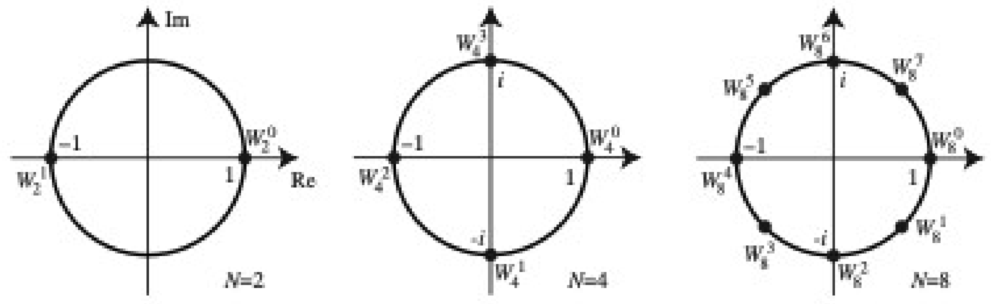
=   for and

By mean of the DFT, x can be represented through N coefficients

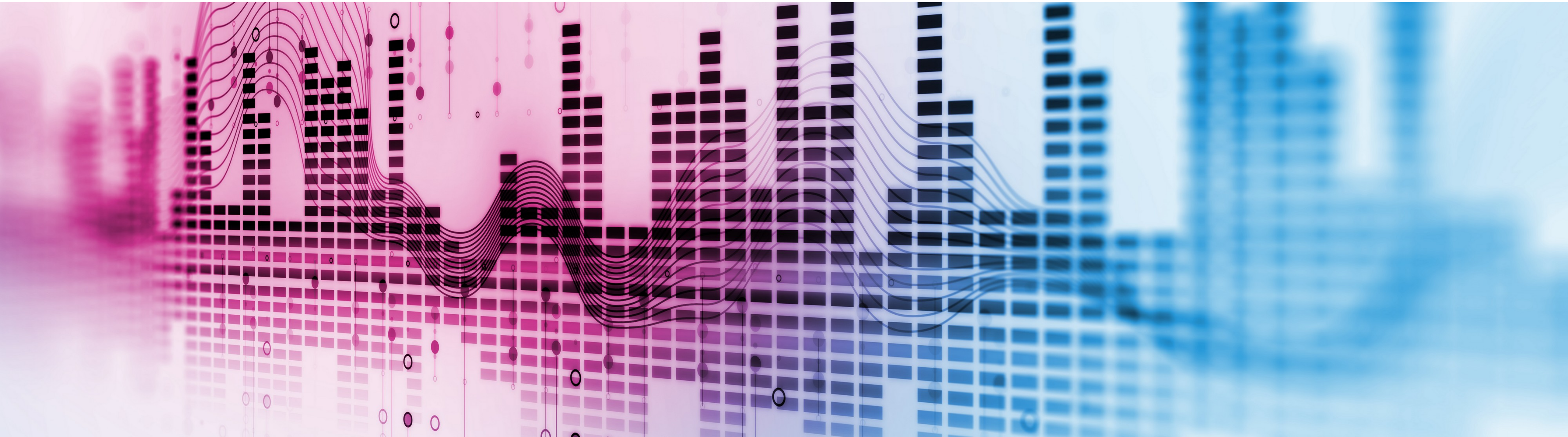
That is more commonly written as

where

$$\begin{pmatrix} A_0 \\ A_1 \\ A_2 \\ A_3 \\ A_4 \\ A_5 \\ A_6 \\ A_7 \end{pmatrix} = \begin{pmatrix} W^0 & W^0 & W^0 & W^0 & W^0 & W^0 & W^0 & W^0 \\ W^0 & W^1 & W^2 & W^3 & W^4 & W^5 & W^6 & W^7 \\ W^0 & W^2 & W^4 & W^6 & W^0 & W^2 & W^4 & W^6 \\ W^0 & W^3 & W^6 & W^1 & W^4 & W^7 & W^2 & W^5 \\ W^0 & W^4 & W^0 & W^4 & W^0 & W^4 & W^0 & W^4 \\ W^0 & W^5 & W^2 & W^7 & W^4 & W^1 & W^6 & W^3 \\ W^0 & W^6 & W^4 & W^2 & W^0 & W^6 & W^4 & W^2 \\ W^0 & W^7 & W^6 & W^5 & W^4 & W^3 & W^2 & W^1 \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \\ a_6 \\ a_7 \end{pmatrix}$$



FFT: ALGORITHM



FAST FOURIER TRANSFORM – DFT COMPUTATION

Two-point DFT (N=2)

$W_2 = e^{-i\pi} = -1$, and

$$A_k = \sum_{n=0}^{1} (-1)^{kn} a_n = (-1)^{k \cdot 0} a_0 + (-1)^{k \cdot 1} a_1 = a_0 + (-1)^k a_1$$

so

$$A_0 = a_0 + a_1$$
$$A_1 = a_0 - a_1$$

Four-point DFT (N=4)

$W_4 = e^{-i\pi/2} = -i$, and

$$A_k = \sum_{n=0}^{3} (-i)^{kn} a_n = a_0 + (-i)^k a_1 + (-i)^{2k} a_2 + (-i)^{3k} a_3 = a_0 + (-i)^k a_1 + (-1)^k a_2 + i^k a_3$$

so

$$A_0 = a_0 + a_1 + a_2 + a_3$$
$$A_1 = a_0 - ia_1 - a_2 + ia_3$$
$$A_2 = a_0 - a_1 + a_2 - a_3$$
$$A_3 = a_0 + ia_1 - a_2 - ia_3$$

This can also be written as a matrix multiply:

$$\begin{bmatrix} A_0 \\ A_1 \\ A_2 \\ A_3 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -i & -1 & i \\ 1 & -1 & 1 & -1 \\ 1 & i & -1 & -i \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix}$$

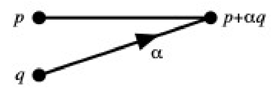
To compute the N coefficients, operations are needed.
So, the computational complexity for the DFT is

To compute A quickly, we can pre-compute common subexpressions:

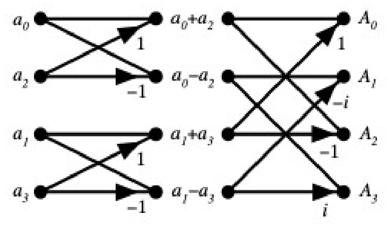
$$A_0 = (a_0 + a_2) + (a_1 + a_3)$$
$$A_1 = (a_0 - a_2) - i(a_1 - a_3)$$
$$A_2 = (a_0 + a_2) - (a_1 + a_3)$$
$$A_3 = (a_0 - a_2) + i(a_1 - a_3)$$

This saves a lot of adds. (Note that each add and multiply here is a complex (not real) operation.)

If we use the following diagram for a complex multiply and add:



then we can diagram the 4-point DFT like so:

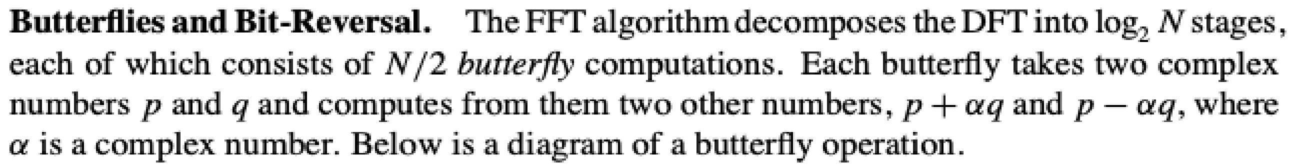


If we carry on to $N = 8$, $N = 16$, and other power-of-two discrete Fourier transforms, we get...

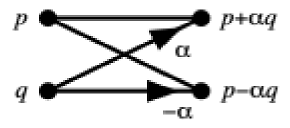
FAST FOURIER TRANSFORM

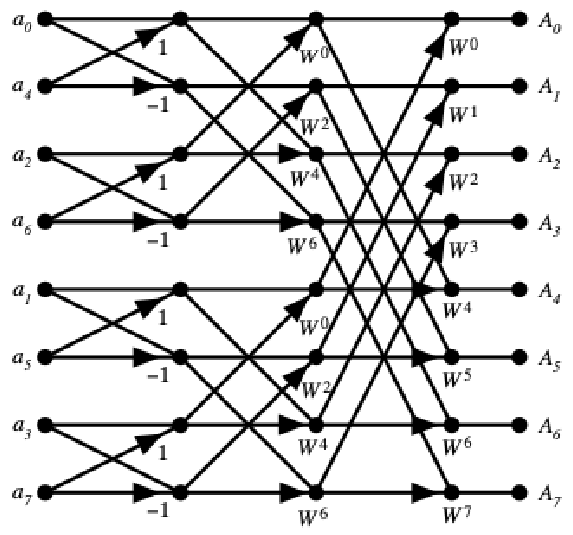
The FFT Algorithm defines a fast way to compute the DFT.

Given a N point scenario - with an N that is a power of two: N = 8,16,32 - **the FFT allows a fast computation based on operations.**



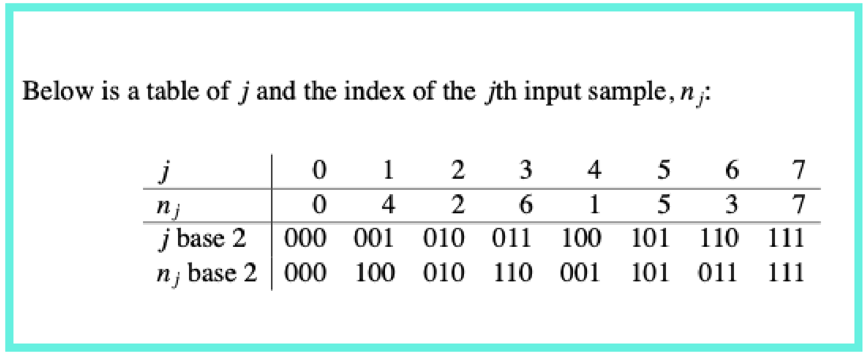
Butterflies and Bit-Reversal. The FFT algorithm decomposes the DFT into $\log_2 N$ stages, each of which consists of $N/2$ *butterfly* computations. Each butterfly takes two complex numbers p and q and computes from them two other numbers, $p + \alpha q$ and $p - \alpha q$, where α is a complex number. Below is a diagram of a butterfly operation.





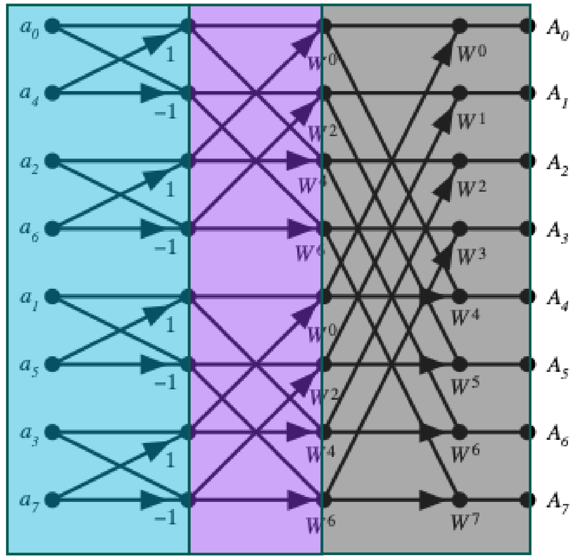
N = 8
# Stages = 3
# butterfly per stages = 4

FAST FOURIER TRANSFORM



Below is a table of j and the index of the jth input sample, n_j:

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



The butterfly coefficients can be seen to have exponents in arithmetic sequence modulo N.

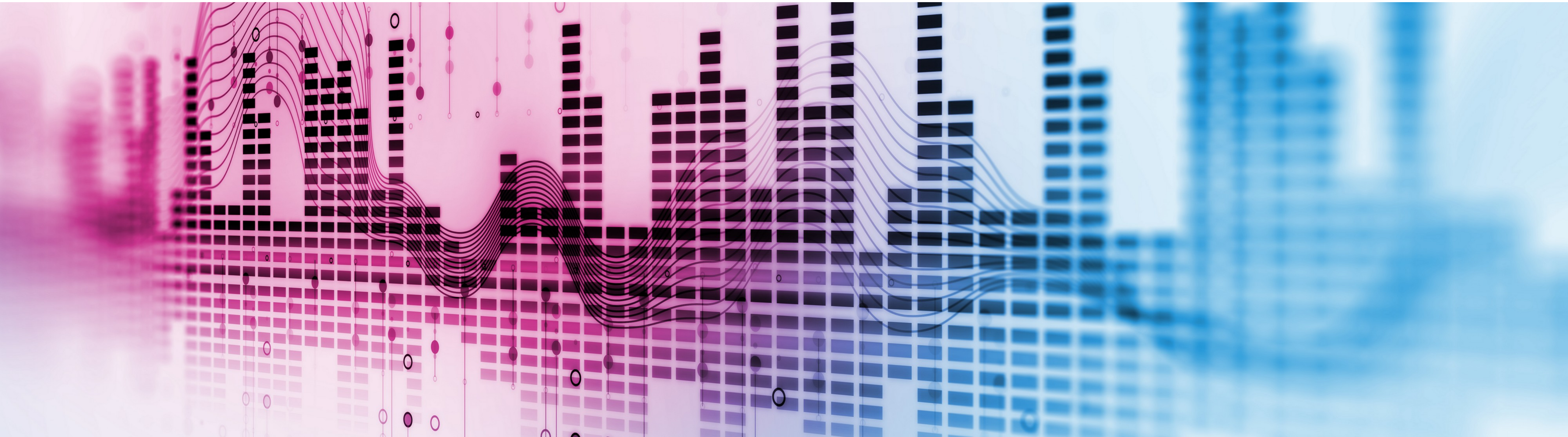
Starting from the last stage, the coefficient are

On the previous stage we can find that is the sequence 0,2,4,6,8,10,12 modulo 8.

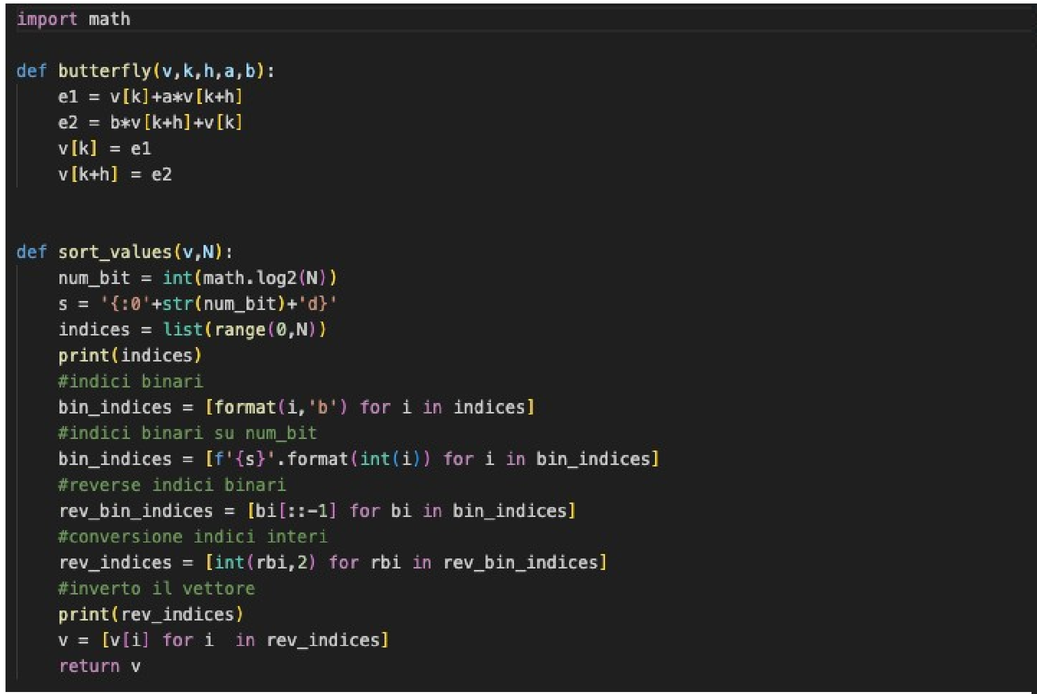
The first coefficients are 1,-1,1,-1,1,-1,..., -1 that are equivalent to the exponent sequence 0,4,8,12,16,20,24,28 modulo 8.

The height of the X's in the diagram can ben seen as 1,2,4,... in successive stages, and the butterflies can be isolated in the first stage, then clustered into overlapping groups of 2 in the second stage, 4 in the 3d stage and so on.

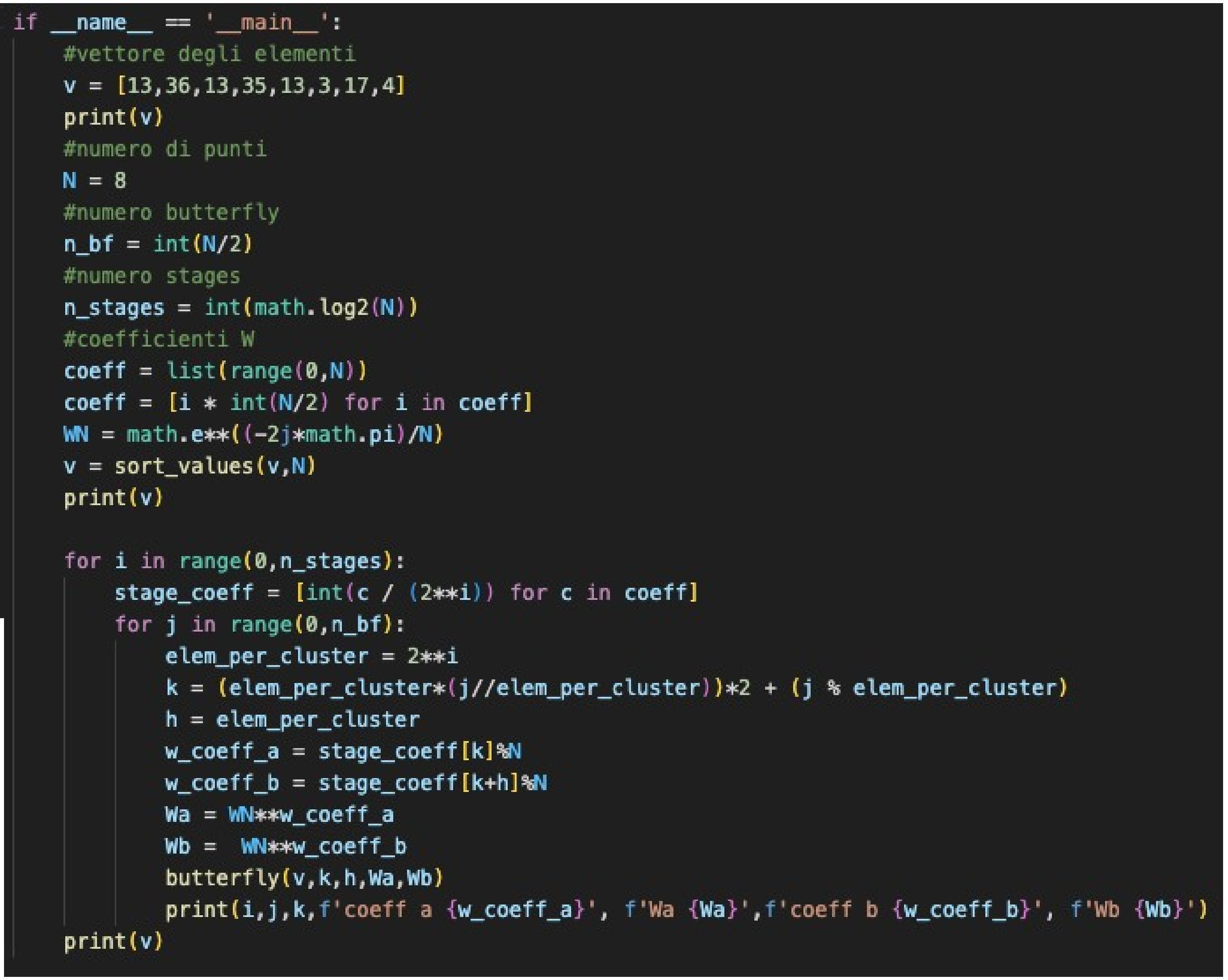
IMPLEMENTATION OF THE ALGORITHM



FAST FOURIER TRANSFORM - IMPLEMENTATION

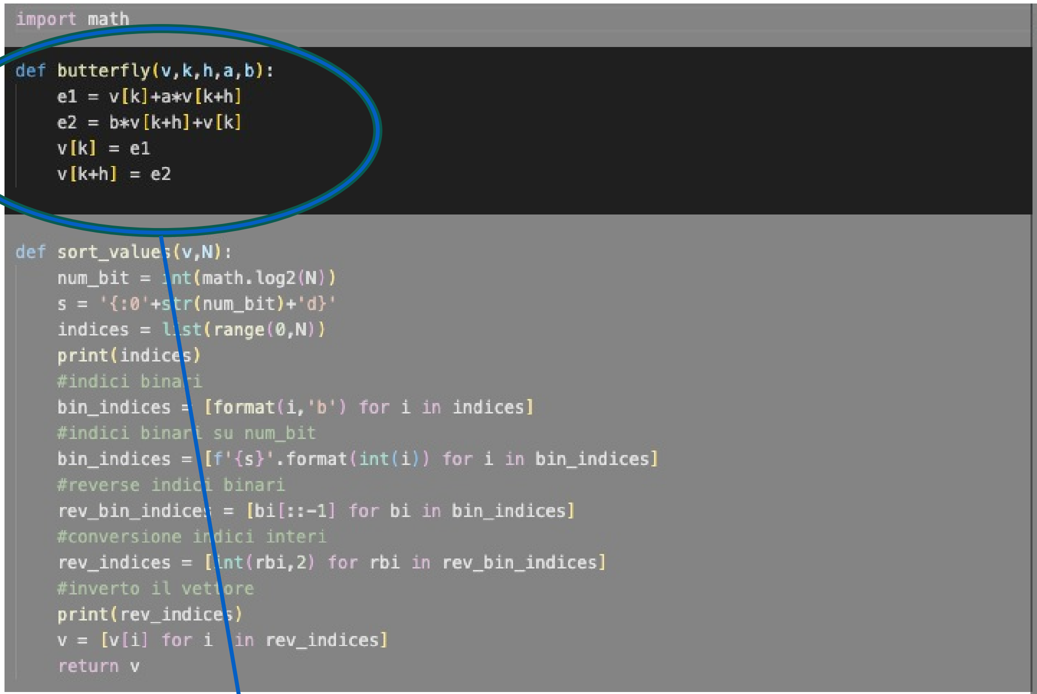


```python
import math

def butterfly(v,k,h,a,b):
    e1 = v[k]+a*v[k+h]
    e2 = b*v[k+h]+v[k]
    v[k] = e1
    v[k+h] = e2


def sort_values(v,N):
    num_bit = int(math.log2(N))
    s = '{:0'+str(num_bit)+'d}'
    indices = list(range(0,N))
    print(indices)
    #indici binari
    bin_indices = [format(i,'b') for i in indices]
    #indici binari su num_bit
    bin_indices = [f'{s}'.format(int(i)) for i in bin_indices]
    #reverse indici binari
    rev_bin_indices = [bi[::-1] for bi in bin_indices]
    #conversione indici interi
    rev_indices = [int(rbi,2) for rbi in rev_bin_indices]
    #inverto il vettore
    print(rev_indices)
    v = [v[i] for i  in rev_indices]
    return v
```

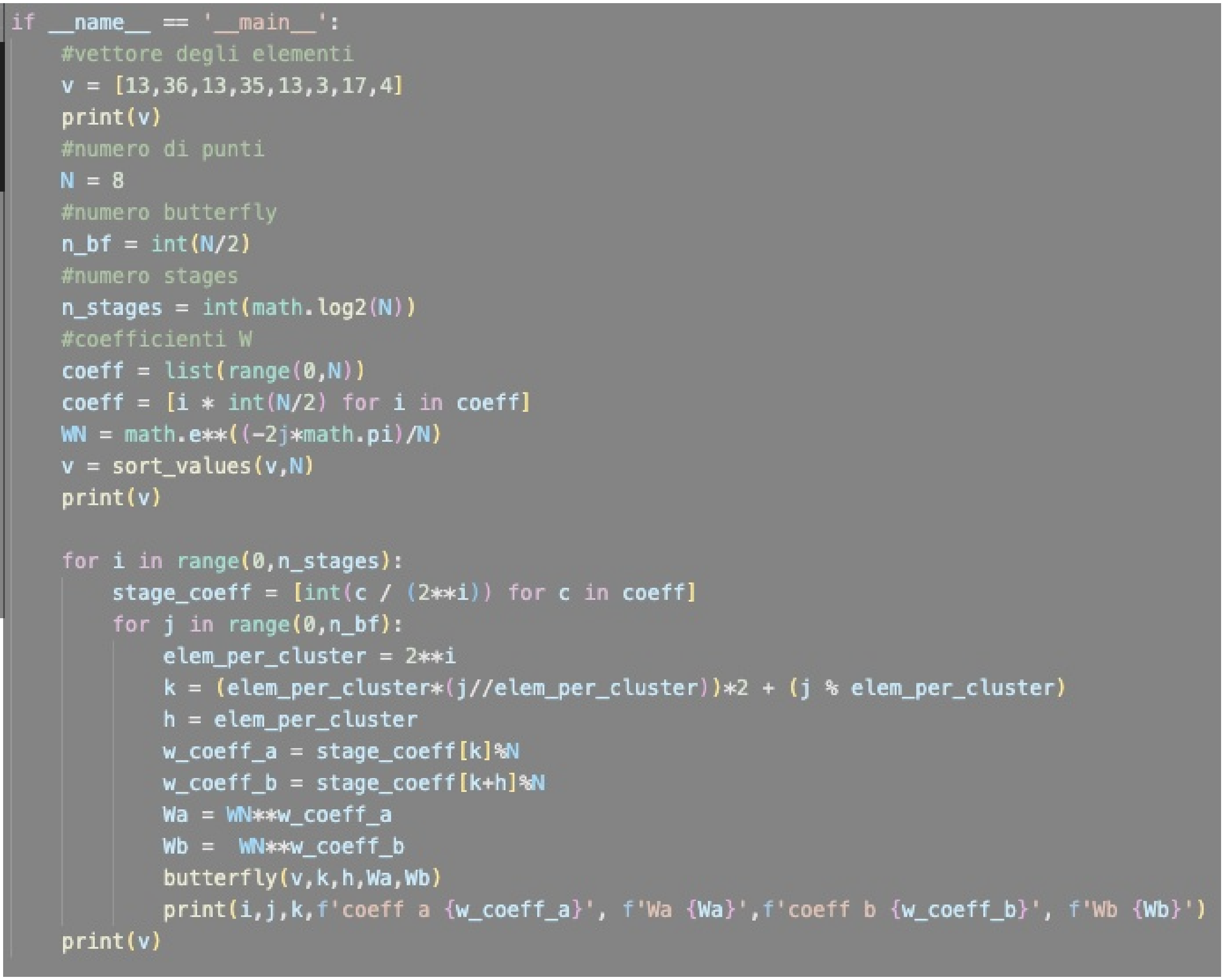


```python
if __name__ == '__main__':
    #vettore degli elementi
    v = [13,36,13,35,13,3,17,4]
    print(v)
    #numero di punti
    N = 8
    #numero butterfly
    n_bf = int(N/2)
    #numero stages
    n_stages = int(math.log2(N))
    #coefficienti W
    coeff = list(range(0,N))
    coeff = [i * int(N/2) for i in coeff]
    WN = math.e**((-2j*math.pi)/N)
    v = sort_values(v,N)
    print(v)

    for i in range(0,n_stages):
        stage_coeff = [int(c / (2**i)) for c in coeff]
        for j in range(0,n_bf):
            elem_per_cluster = 2**i
            k = (elem_per_cluster*(j//elem_per_cluster))*2 + (j % elem_per_cluster)
            h = elem_per_cluster
            w_coeff_a = stage_coeff[k]%N
            w_coeff_b = stage_coeff[k+h]%N
            Wa = WN**w_coeff_a
            Wb =  WN**w_coeff_b
            butterfly(v,k,h,Wa,Wb)
            print(i,j,k,f'coeff a {w_coeff_a}', f'Wa {Wa}',f'coeff b {w_coeff_b}', f'Wb {Wb}')
    print(v)
```

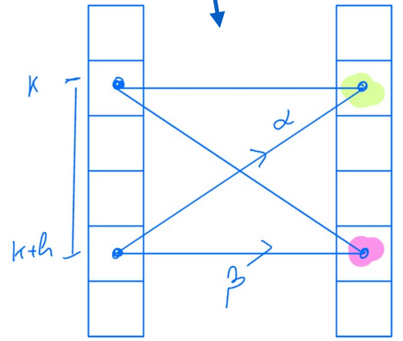
FAST FOURIER TRANSFORM - IMPLEMENTATION



```python
import math

def butterfly(v,k,h,a,b):
    e1 = v[k]+a*v[k+h]
    e2 = b*v[k+h]+v[k]
    v[k] = e1
    v[k+h] = e2

def sort_values(v,N):
    num_bit = int(math.log2(N))
    s = '{:0'+str(num_bit)+'d}'
    indices = list(range(0,N))
    print(indices)
    #indici binari
    bin_indices = [format(i,'b') for i in indices]
    #indici binari su num_bit
    bin_indices = [f'{s}'.format(int(i)) for i in bin_indices]
    #reverse indici binari
    rev_bin_indices = [bi[::-1] for bi in bin_indices]
    #conversione indici interi
    rev_indices = [int(rbi,2) for rbi in rev_bin_indices]
    #inverto il vettore
    print(rev_indices)
    v = [v[i] for i  in rev_indices]
    return v
```



```python
if __name__ == '__main__':
    #vettore degli elementi
    v = [13,36,13,35,13,3,17,4]
    print(v)
    #numero di punti
    N = 8
    #numero butterfly
    n_bf = int(N/2)
    #numero stages
    n_stages = int(math.log2(N))
    #coefficienti W
    coeff = list(range(0,N))
    coeff = [i * int(N/2) for i in coeff]
    WN = math.e**((-2j*math.pi)/N)
    v = sort_values(v,N)
    print(v)

    for i in range(0,n_stages):
        stage_coeff = [int(c / (2**i)) for c in coeff]
        for j in range(0,n_bf):
            elem_per_cluster = 2**i
            k = (elem_per_cluster*(j//elem_per_cluster))*2 + (j % elem_per_cluster)
            h = elem_per_cluster
            w_coeff_a = stage_coeff[k]%N
            w_coeff_b = stage_coeff[k+h]%N
            Wa = WN**w_coeff_a
            Wb =  WN**w_coeff_b
            butterfly(v,k,h,Wa,Wb)
            print(i,j,k,f'coeff a {w_coeff_a}', f'Wa {Wa}',f'coeff b {w_coeff_b}', f'Wb {Wb}')
    print(v)
```

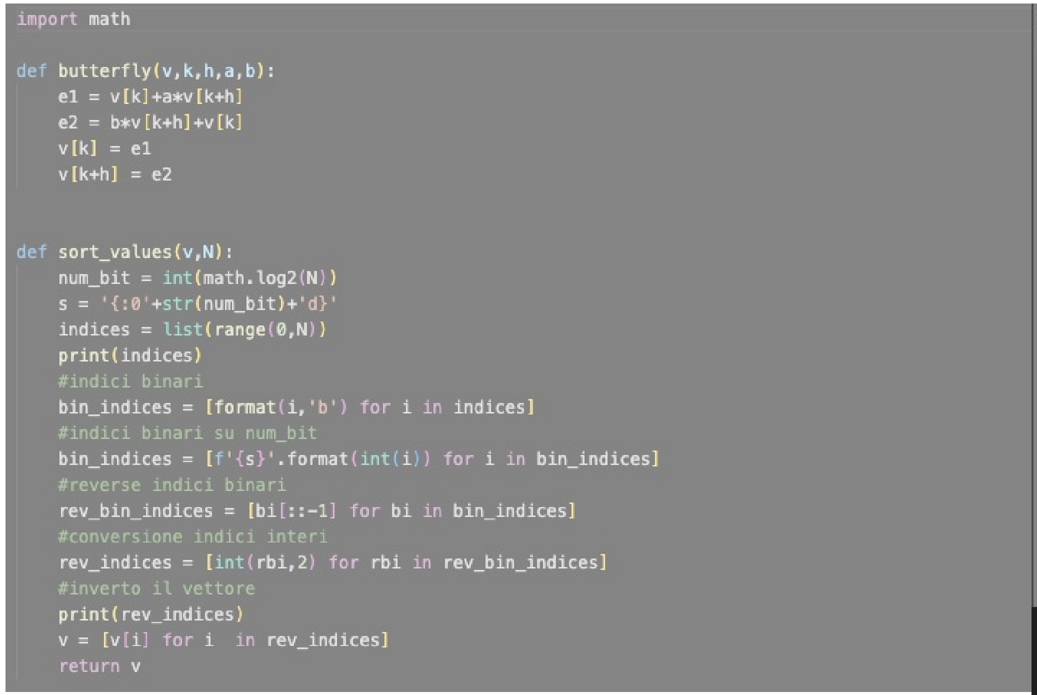


Butterfly (v, k, h, α, β)

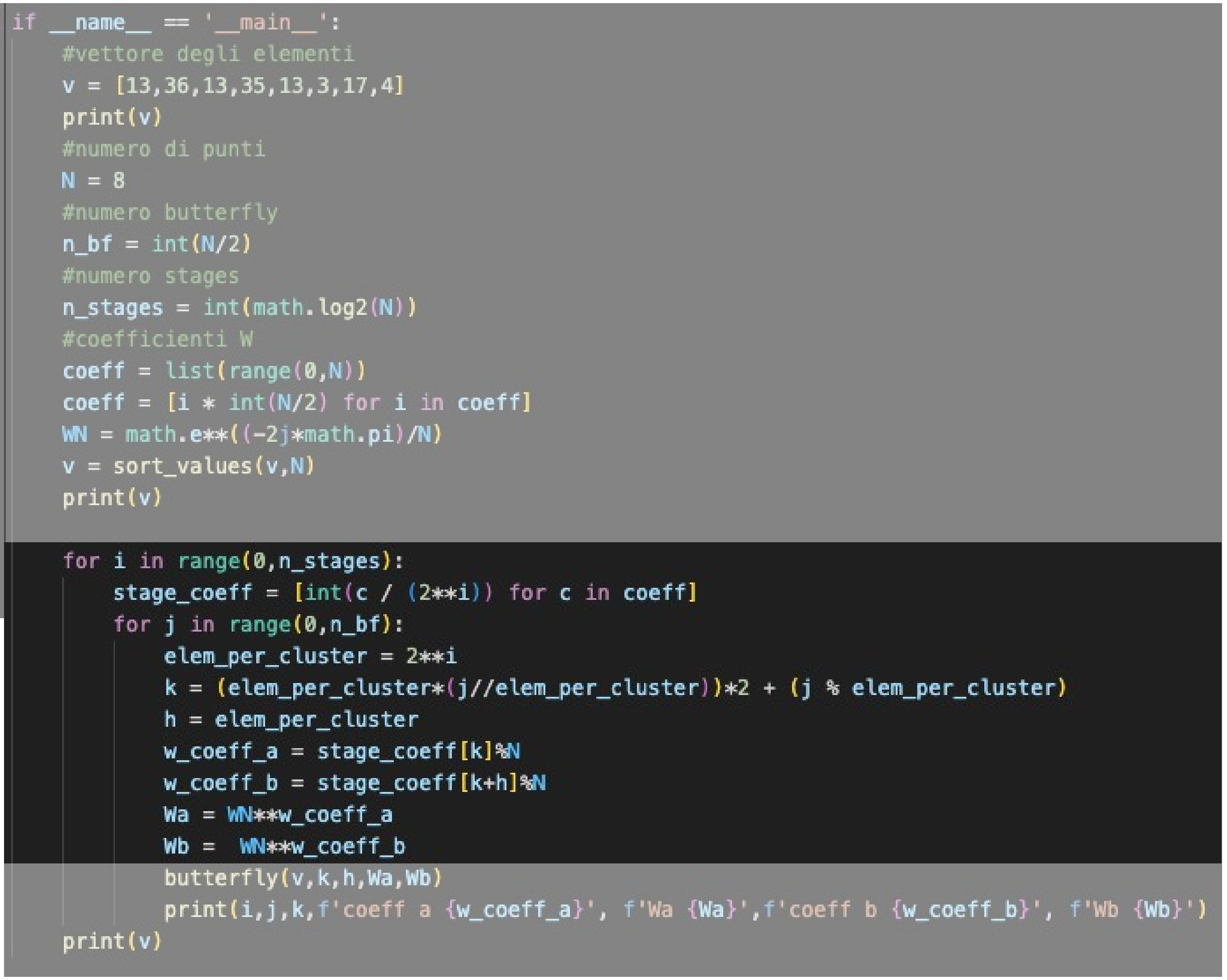
$e_1 = v[k] + \alpha\, v[k+h]$

$e_2 = v[k] + \beta\, v[k+h]$

FAST FOURIER TRANSFORM - IMPLEMENTATION

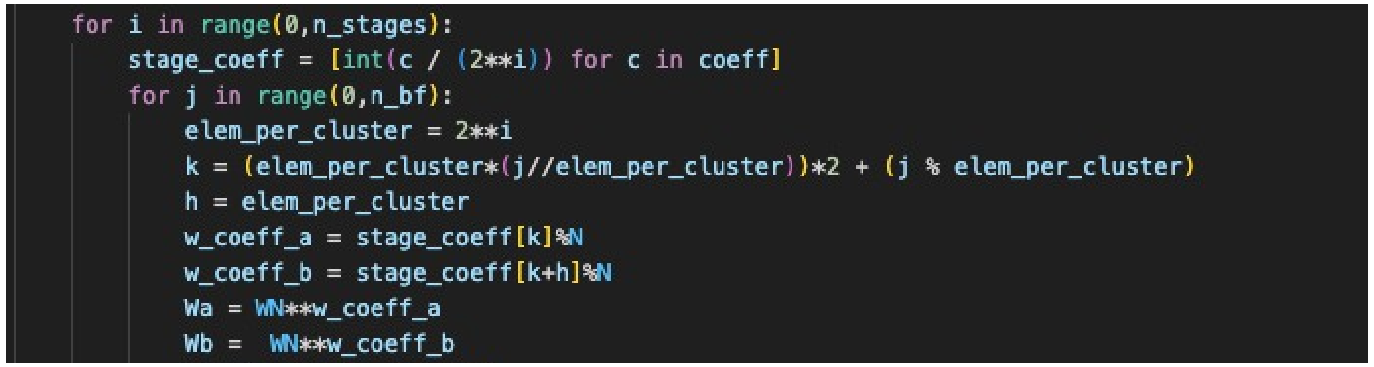


```python
import math

def butterfly(v,k,h,a,b):
    e1 = v[k]+a*v[k+h]
    e2 = b*v[k+h]+v[k]
    v[k] = e1
    v[k+h] = e2


def sort_values(v,N):
    num_bit = int(math.log2(N))
    s = '{:0'+str(num_bit)+'d}'
    indices = list(range(0,N))
    print(indices)
    #indici binari
    bin_indices = [format(i,'b') for i in indices]
    #indici binari su num_bit
    bin_indices = [f'{s}'.format(int(i)) for i in bin_indices]
    #reverse indici binari
    rev_bin_indices = [bi[::-1] for bi in bin_indices]
    #conversione indici interi
    rev_indices = [int(rbi,2) for rbi in rev_bin_indices]
    #inverto il vettore
    print(rev_indices)
    v = [v[i] for i  in rev_indices]
    return v
```

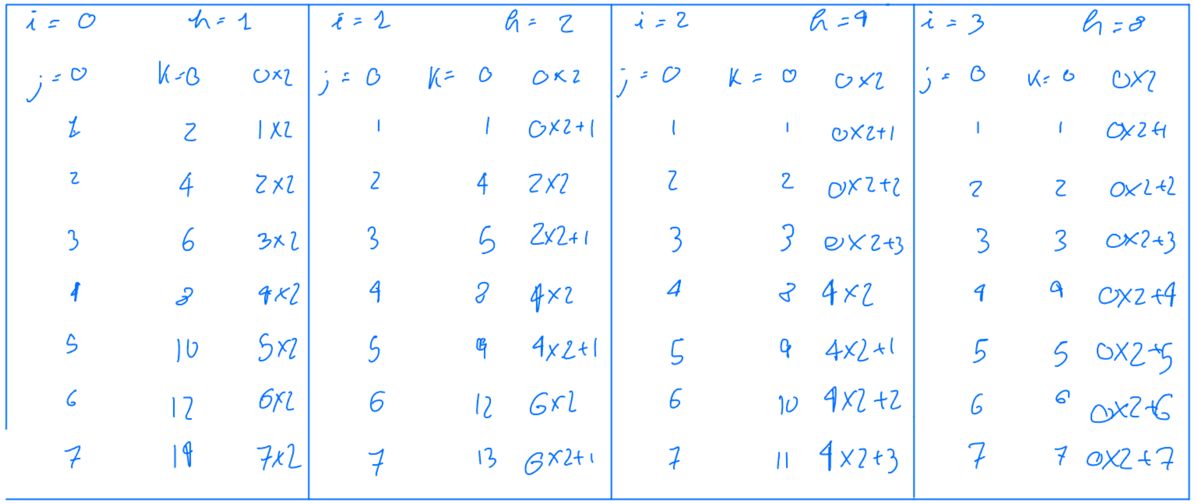


```python
if __name__ == '__main__':
    #vettore degli elementi
    v = [13,36,13,35,13,3,17,4]
    print(v)
    #numero di punti
    N = 8
    #numero butterfly
    n_bf = int(N/2)
    #numero stages
    n_stages = int(math.log2(N))
    #coefficienti W
    coeff = list(range(0,N))
    coeff = [i * int(N/2) for i in coeff]
    WN = math.e**((-2j*math.pi)/N)
    v = sort_values(v,N)
    print(v)

    for i in range(0,n_stages):
        stage_coeff = [int(c / (2**i)) for c in coeff]
        for j in range(0,n_bf):
            elem_per_cluster = 2**i
            k = (elem_per_cluster*(j//elem_per_cluster))*2 + (j % elem_per_cluster)
            h = elem_per_cluster
            w_coeff_a = stage_coeff[k]%N
            w_coeff_b = stage_coeff[k+h]%N
            Wa = WN**w_coeff_a
            Wb =  WN**w_coeff_b
            butterfly(v,k,h,Wa,Wb)
            print(i,j,k,f'coeff a {w_coeff_a}', f'Wa {Wa}',f'coeff b {w_coeff_b}', f'Wb {Wb}')
    print(v)
```

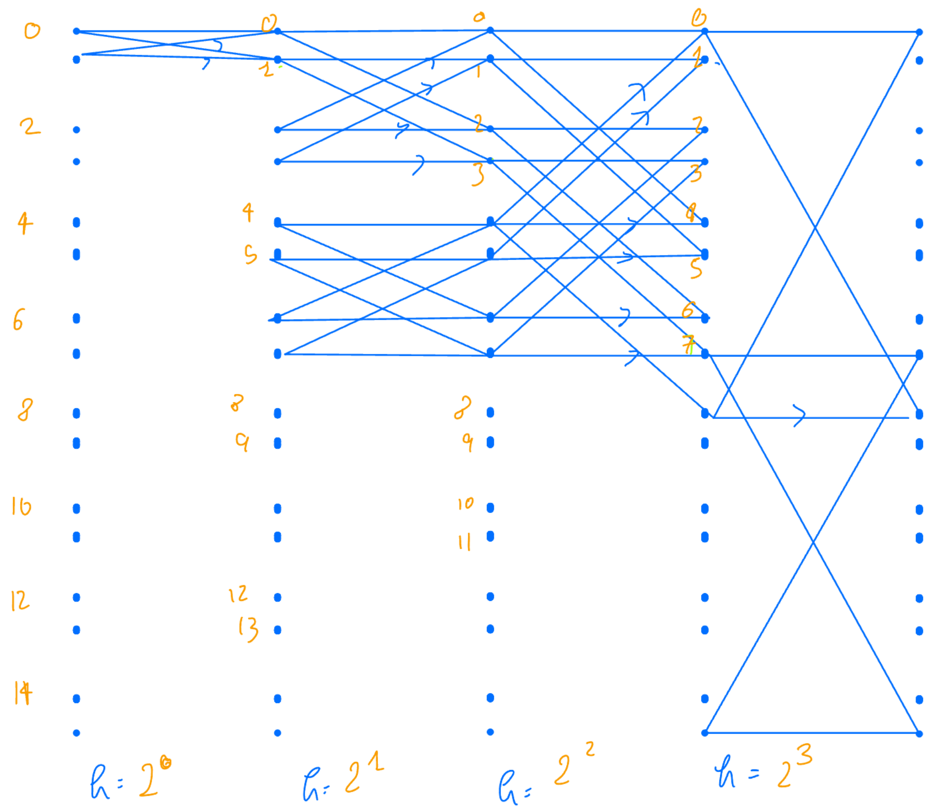
FAST FOURIER TRANSFORM - IMPLEMENTATION



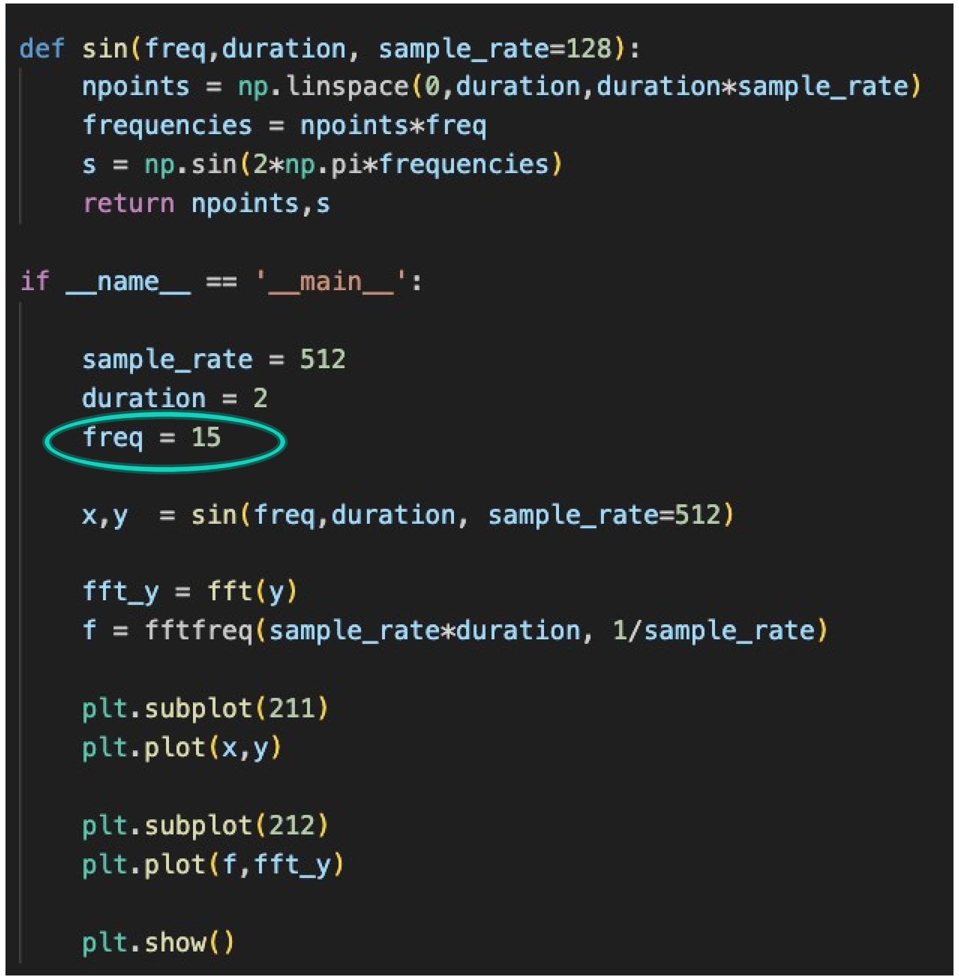
```
for i in range(0,n_stages):
    stage_coeff = [int(c / (2**i)) for c in coeff]
    for j in range(0,n_bf):
        elem_per_cluster = 2**i
        k = (elem_per_cluster*(j//elem_per_cluster))*2 + (j % elem_per_cluster)
        h = elem_per_cluster
        w_coeff_a = stage_coeff[k]%N
        w_coeff_b = stage_coeff[k+h]%N
        Wa = WN**w_coeff_a
        Wb =  WN**w_coeff_b
```



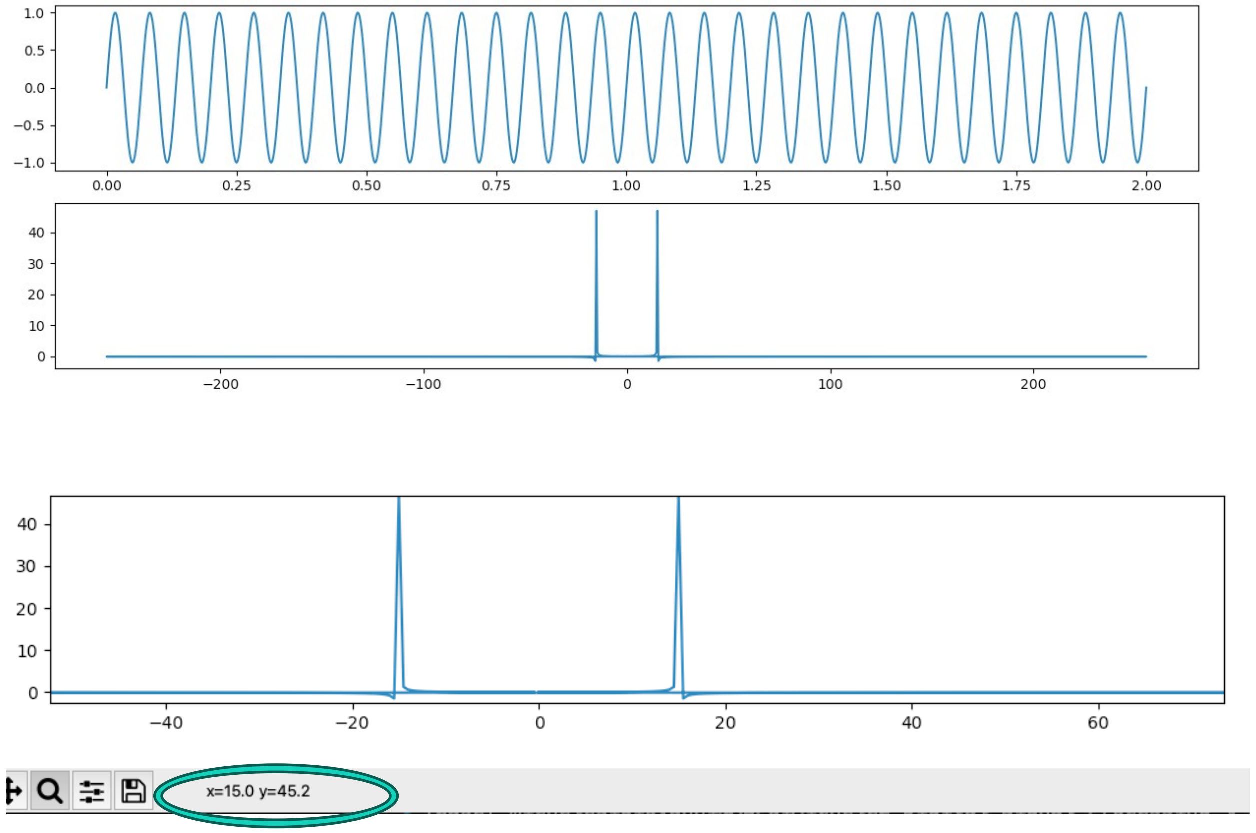
i = 0	h = 1		i = 1	h = 2		i = 2	h = 4		i = 3	h = 8	
j = 0	K = 0	0x2	j = 0	K = 0	0x2	j = 0	K = 0	0x2	j = 0	K = 0	0x2
1	2	1x2	1	1	0x2+1	1	1	0x2+1	1	1	0x2+1
2	4	2x2	2	4	2x2	2	2	0x2+2	2	2	0x2+2
3	6	3x2	3	5	2x2+1	3	3	0x2+3	3	3	0x2+3
4	8	4x2	4	8	4x2	4	8	4x2	4	4	0x2+4
5	10	5x2	5	9	4x2+1	5	9	4x2+1	5	5	0x2+5
6	12	6x2	6	12	6x2	6	10	4x2+2	6	6	0x2+6
7	14	7x2	7	13	6x2+1	7	11	4x2+3	7	7	0x2+7



FAST FOURIER TRANSFORM – TEST ON PYTHON



```
def sin(freq,duration, sample_rate=128):
    npoints = np.linspace(0,duration,duration*sample_rate)
    frequencies = npoints*freq
    s = np.sin(2*np.pi*frequencies)
    return npoints,s

if __name__ == '__main__':

    sample_rate = 512
    duration = 2
    freq = 15

    x,y  = sin(freq,duration, sample_rate=512)

    fft_y = fft(y)
    f = fftfreq(sample_rate*duration, 1/sample_rate)

    plt.subplot(211)
    plt.plot(x,y)

    plt.subplot(212)
    plt.plot(f,fft_y)

    plt.show()
```



x=15.0 y=45.2