EXPERIMENT NO 6

NAME: Vishal Shashikant Salvi **UID: 2019230069**
CLASS: TE COMPS **BATCH: C**

Aim: To implement Fast Fourier Transform and Inverse Fast Fourier Transform.

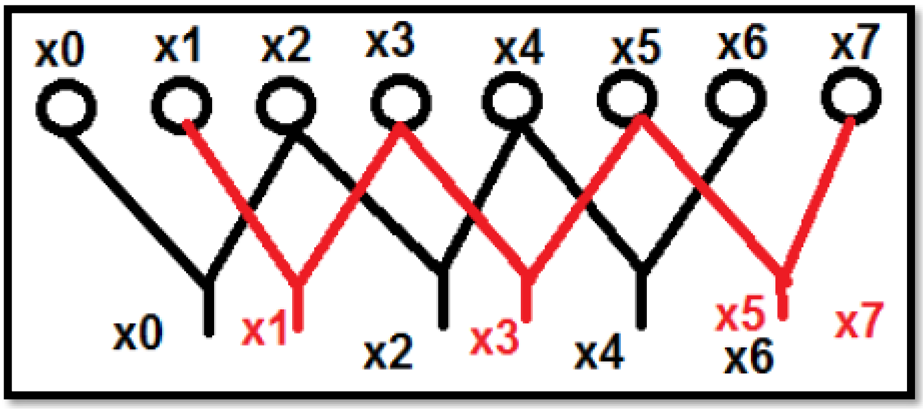
Theory:

In earlier DFT methods, we have seen that the computational part is too long. We want to reduce that. This can be done through FFT or fast Fourier transform. So, we can say FFT is nothing but computation of discrete Fourier transform in an algorithmic format, where the computational part will be reduced.

The main advantage of having FFT is that through it, we can design the FIR filters. Mathematically, the FFT can be written as follows;

$$\boxed{x[K] = \sum_{n=0}^{N-1} x[n] W_N^{nk}}$$

Let us take an example to understand it better. We have considered eight points named from x0tox7x0tox7. We will choose the even terms in one group and the odd terms in the other. Diagrammatic view of the above said has been shown below –



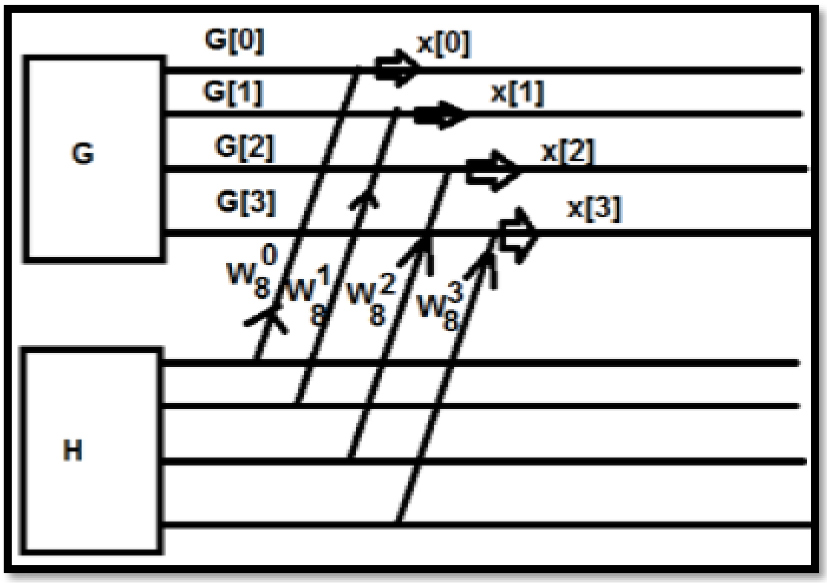
Here, points x_0, x_2, x_4 and x_6 have been grouped into one category and similarly, points x_1, x_3, x_5 and x_7 has been put into another category. Now, we can further make them in a group of two and can proceed with the computation. Now, let us see how these breaking into further two is helping in computation.

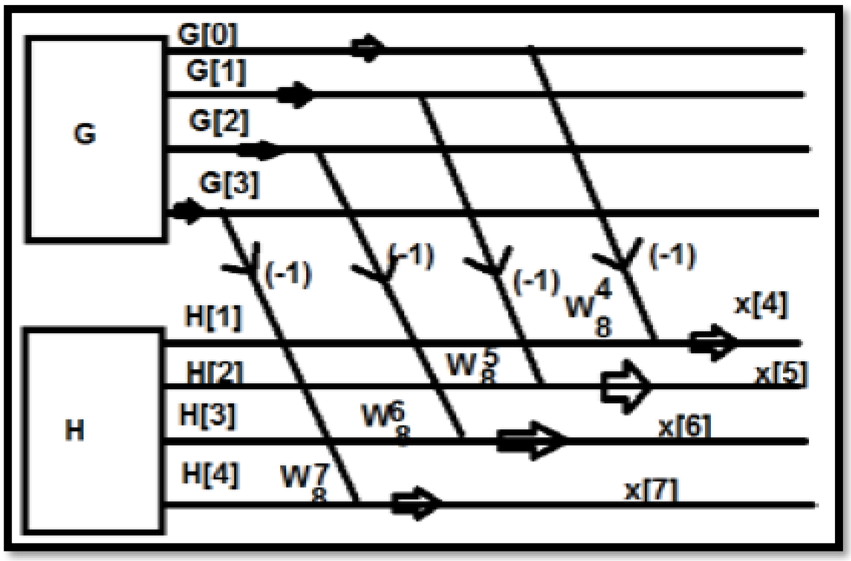
$$x[k] = \sum_{r=0}^{\frac{N}{2}-1} x[2r]W_N^{2rk} + \sum_{r=0}^{\frac{N}{2}-1} x[2r+1]W_N^{(2r+1)k}$$

$$= \sum_{r=0}^{\frac{N}{2}-1} x[2r]W_{N/2}^{rk} + \sum_{r=0}^{\frac{N}{2}-1} x[2r+1]W_{N/2}^{rk} \times W_N^k$$

$$= G[k] + H[k] \times W_N^k$$

Initially, we took an eight-point sequence, but later we broke that one into two parts G[k] and H[k]. G[k] stands for the even part whereas H[k] stands for the odd part. If we want to realize it through a diagram, then it can be shown as below −





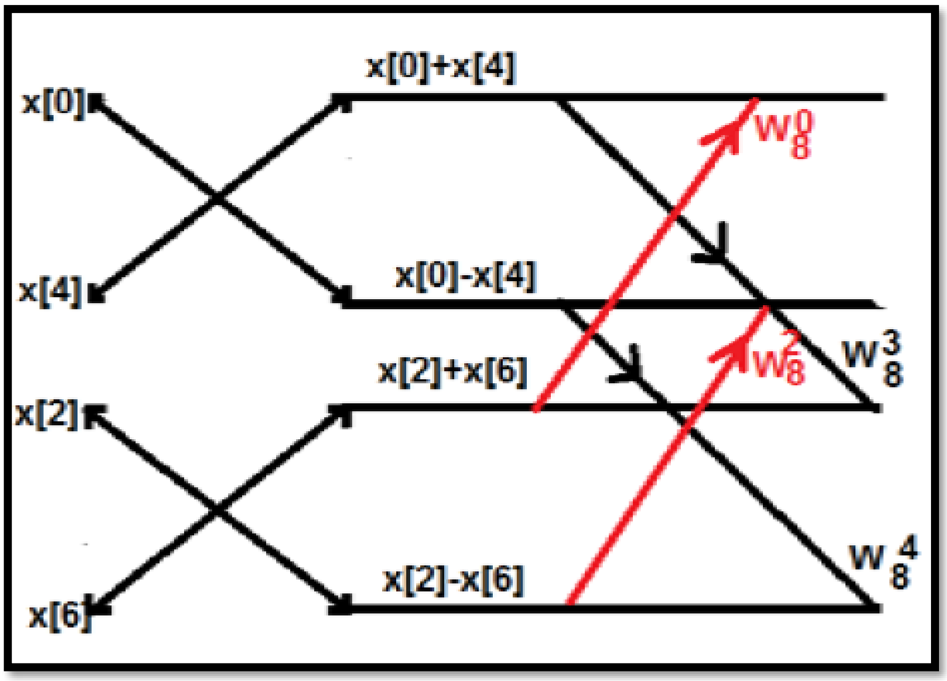
From the above figure, we can see that

$$\boxed{\begin{array}{l} W_8^4 = -1 \\ W_8^5 = -W_8^1 \\ W_8^6 = -W_8^2 \\ W_8^7 = -W_8^3 \end{array}}$$

Similarly, the final values can be written as follows –

$$\boxed{\begin{array}{l} G[0] - H[0] = x[4] \\ G[1] - W_8^1 H[1] = x[5] \\ G[2] - W_8^2 H[2] = x[6] \\ G[1] - W_8^3 H[3] = x[7] \end{array}}$$

The above one is a periodic series. The disadvantage of this system is that K cannot be broken beyond 4 point. Now Let us break down the above into further. We will get the structures something like this

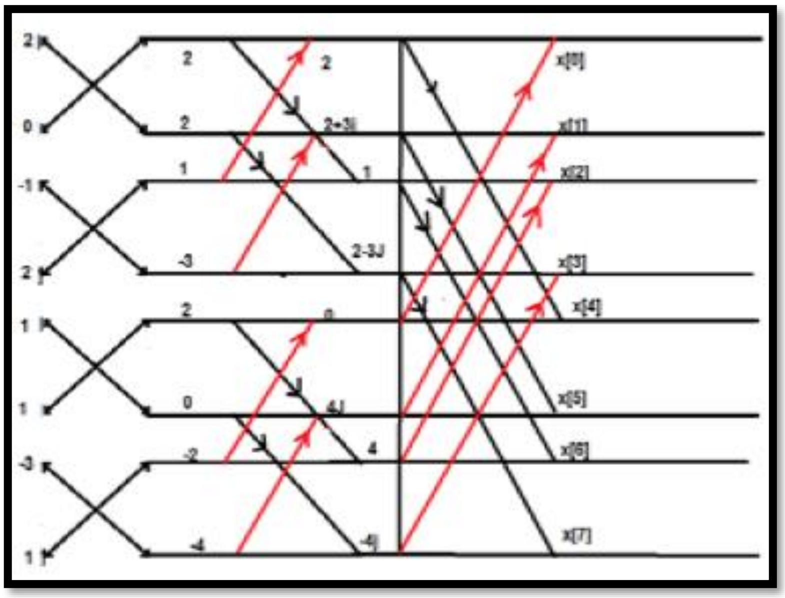


Example

Consider the sequence x[n]={ 2,1,-1,-3,0,1,2,1}. Calculate the FFT.

Solution − The given sequence is x[n]={ 2,1,-1,-3,0,1,2,1}

Arrange the terms as shown below;



IFFT:

IFFT is a fast algorithm to perform inverse (or backward) Fourier transform (IDFT), which undoes the process of DFT. IDFT of a sequence $\{F_n\}$ that can be defined as:

$$\boxed{x_i = \frac{1}{N}\sum_{n=0}^{N-1} F_n e^{\frac{2\pi j}{N} ni}}$$

If an IFFT is performed on a complex FFT result computed by Origin, this will in principle transform the FFT result back to its original data set. However, this is true only when all of the following requirements are met:

- The **Spectrum Type** in FFT is two-sided.
- The **Window** option is *Rectangle* for both IFFT and FFT.
- The **Factor** option is the same for IFFT and FFT.
- If the **Shift** checkbox is selected in FFT, the **Undo Shift Input Data** checkbox must be selected in IFFT. Conversely, if the **Shift** checkbox is cleared in FFT, the **Undo Shift Input Data** checkbox should not be selected in IFFT.

To use IFFT:

1. Make a workbook or a graph active.
2. Select Analysis: Signal Processing: FFT: IFFT from the Origin menu.

Why do we use IFFT?

The Fourier transform is used to convert the signals from time domain to frequency domain and the inverse Fourier transform is used to convert the signal back from the frequency domain to the time domain. The Fourier transform is a powerful tool to analyse the signals and construct them to and from their frequency components. If the signal is discrete in time that is sampled, one uses the discrete Fourier transform to convert them to the discrete frequency form DFT, and vice versa, the inverse discrete transform IDFT is used to back convert the discrete frequency form into the discrete time form.

To reduce the mathematical operations used in the calculation of DFT and IDFT one uses the fast Fourier transform algorithm FFT and IFFT which corresponds to DFT and IDFT, respectively.

In transmitters using OFDM as a multicarrier modulation technology, the OFDM symbol is constructed in the frequency domain by mapping the input bits on the I- and Q- components of the QAM symbols and then ordering them in a sequence with specific length according to the number of subcarriers in the OFDM symbol. That is by the mapping and ordering process, one constructs the frequency components of the OFDM symbol. To transmit them, the signal must be represented in time domain. This is accomplished by the inverse fast Fourier transform IFFT.

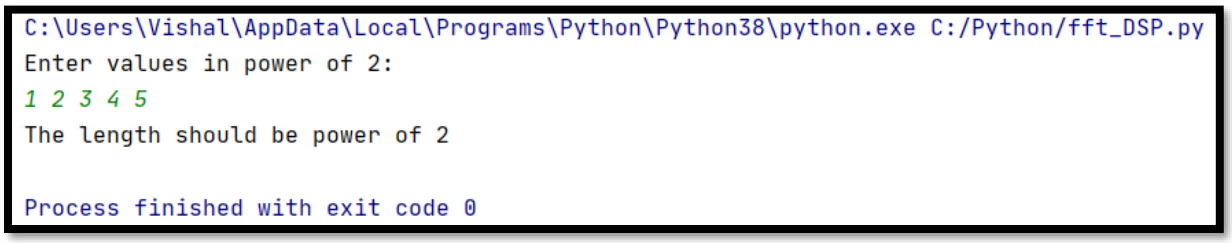
So, in summary the signal is easier synthesized in discrete frequency domain in the transmitter and to transmit it must be converted to discrete time domain by IFFT.

Code:

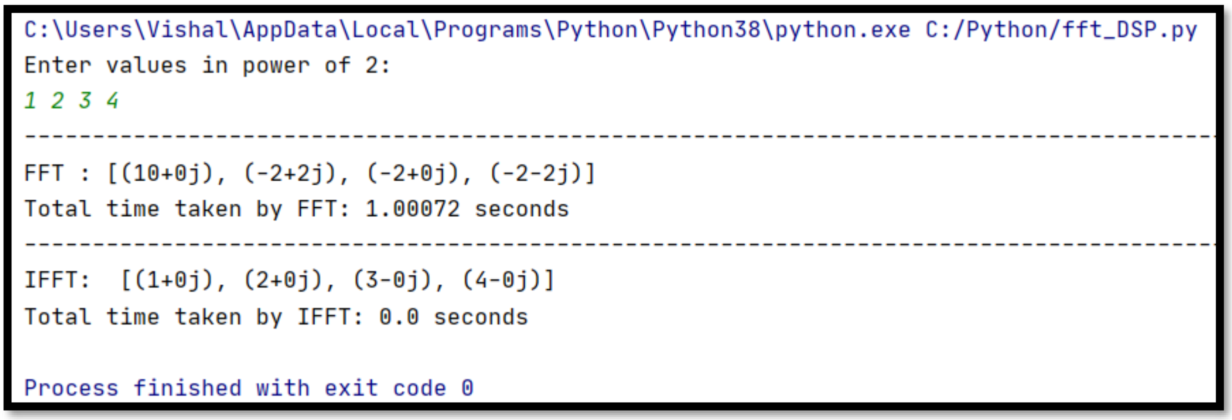
```
import numpy as np
import math
import time

# fft function
def fft(xn):
  time.sleep(1)
  N = len(xn)
  add = [0]*N
  comp = 0-1j

  for k in range(N):
    for n in range(N):
      add[k] = add[k]+xn[n]*((np.exp((math.pi*2/N)*comp))**(n*k))

  for xk in range(N):
    add[xk] = np.round(add[xk].real, 2)+np.round(add[xk].imag, 2)*1j

  return(add)

# ifft function
def ifft(x):
  n = len(x)
  w_n = np.exp(-2j*np.pi/n)
  w_n_exp = [np.power(w_n,-i) for i in range(n)]

  def ifft_rec(x):
    if len(x) <= 1:
      return x
    w_n_terms = w_n_exp[::n//len(x)]
    x_even, x_odd = ifft_rec(x[::2]),ifft_rec(x[1::2])
    res = [x_even[i%(len(x)//2)] + x_odd[i%(len(x)//2)]*w_n_terms[i] for i in range(len(x))]
    return res

  return [np.round(k/n, 2) for k in ifft_rec(x)]

# taking inputs
input_xn = list(map(int, input("Enter values in power of 2:\n").split()))

# checking length
N = len(input_xn)
if N%2>0:
  print("The length should be power of 2 ")
  exit()
print("----------------------------------------------------------------------------------------------------
-----------" )

# calculating fft time
time.sleep(1)
```

```
start_time=time.time()
yn_fft=fft(input_xn)
fft_time = time.time()
print("FFT :" ,yn_fft)
print("Total time taken by FFT:", np.round(fft_time-start_time,5),"seconds")

print("---------------------------------------------------------------------------------------------------
-----------" )

# calculating ifft time
t1=time.time()
yn_ifft=ifft(yn_fft)
t2=time.time()
print("IFFT: ",yn_ifft)
print("Total time taken by IFFT:", np.round(t2-t1,5),"seconds")
```

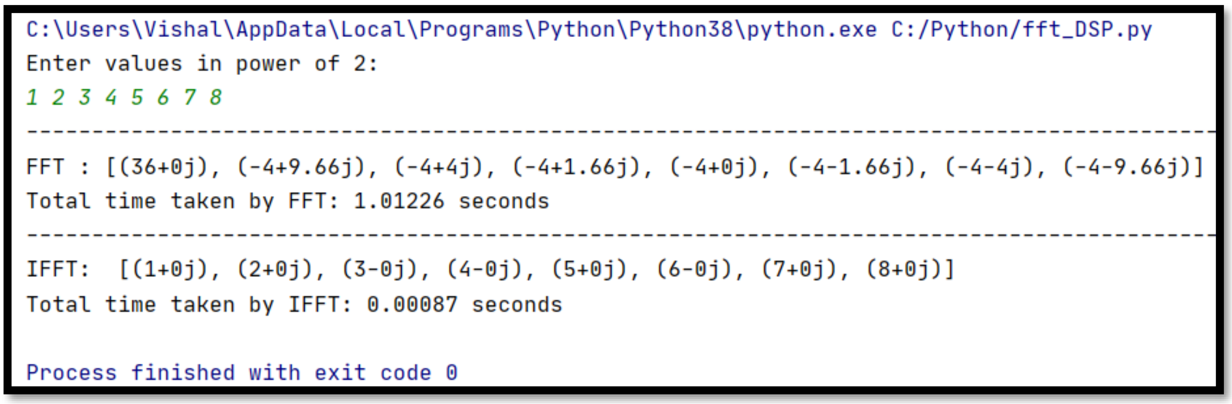
Output:



```
C:\Users\Vishal\AppData\Local\Programs\Python\Python38\python.exe C:/Python/fft_DSP.py
Enter values in power of 2:
1 2 3 4 5
The length should be power of 2

Process finished with exit code 0
```



```
C:\Users\Vishal\AppData\Local\Programs\Python\Python38\python.exe C:/Python/fft_DSP.py
Enter values in power of 2:
1 2 3 4
----------------------------------------------------------------------------------------
FFT : [(10+0j), (-2+2j), (-2+0j), (-2-2j)]
Total time taken by FFT: 1.00072 seconds
----------------------------------------------------------------------------------------
IFFT:  [(1+0j), (2+0j), (3-0j), (4-0j)]
Total time taken by IFFT: 0.0 seconds

Process finished with exit code 0
```



```
C:\Users\Vishal\AppData\Local\Programs\Python\Python38\python.exe C:/Python/fft_DSP.py
Enter values in power of 2:
1 2 3 4 5 6 7 8
----------------------------------------------------------------------------------------
FFT : [(36+0j), (-4+9.66j), (-4+4j), (-4+1.66j), (-4+0j), (-4-1.66j), (-4-4j), (-4-9.66j)]
Total time taken by FFT: 1.01226 seconds
----------------------------------------------------------------------------------------
IFFT:  [(1+0j), (2+0j), (3-0j), (4-0j), (5+0j), (6-0j), (7+0j), (8+0j)]
Total time taken by IFFT: 0.00087 seconds

Process finished with exit code 0
```

Conclusion:

Hence, I came to know how to implement Fast Fourier Transform and Inverse Fast Fourier Transform . **FFT** is a collection of algorithms for fast computation of the DFT**.** Typically the number of operations required by the FFT is on the order of *N**log*N*.

Reference:

1) **https://www.tutorialspoint.com/digital_signal_processing/dsp_fast_fourier_transform.htm**
2) **https://www.originlab.com/doc/Origin-Help/IFFT#:~:text=Fourier%20Transform%20(IFFT)-,IFFT,to%20its%20original%20data%20set.**
3) **https://www.researchgate.net/post/Why-do-we-use-IFFT**