Signal Analysis Project

Fast Fourier Transform (FFT)

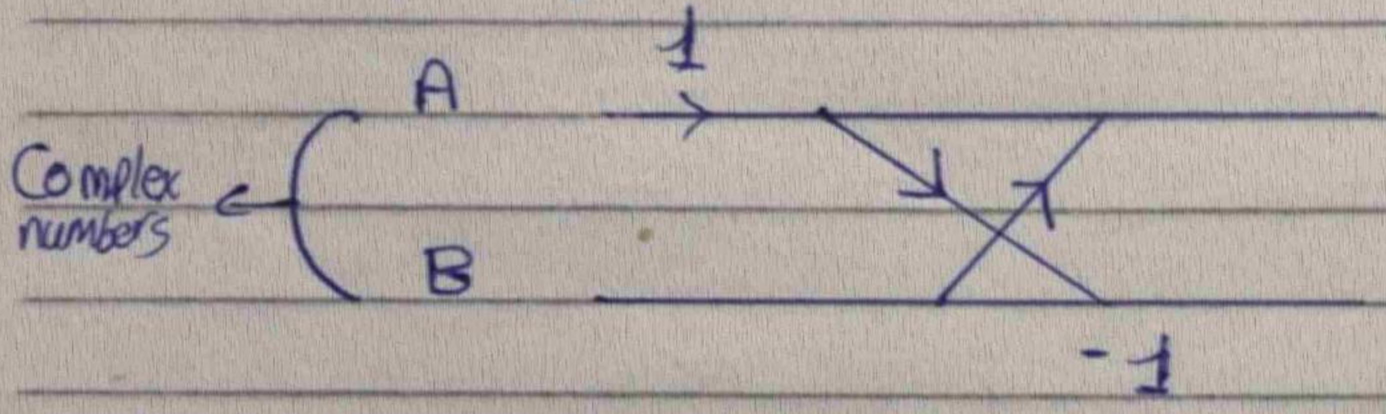
Alhussein Gamal (1200399)

a) How many multiplications are required to calculate the N points complex FS coefficients (a_k) using equation (2).

Multiplications Required = N^2

As the sum of one harmonic is from 0 ---> N-1 , and the number of harmonics are from 0 ---> N-1 , so the multiplications required are N^2.

b) Define the butterfly as a basic module in radix-2 FFT.

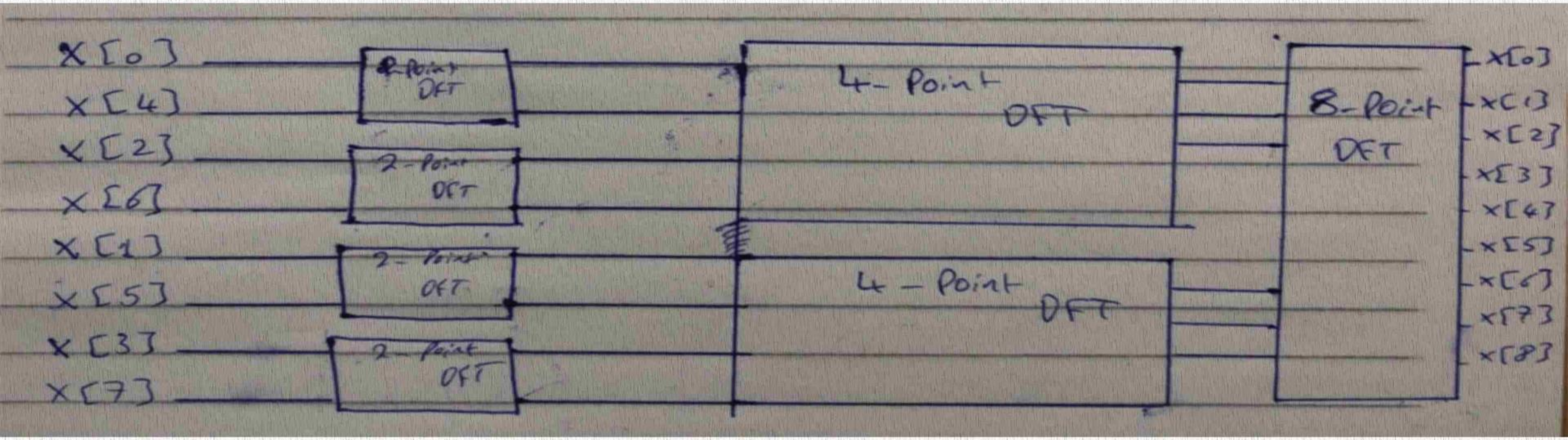


$W_N^K \longrightarrow$ phase factor

$C = A + W_N^K B$

$D = B - W_N^K B$

c) Draw the block diagram of a radix-2 8 points FFT.

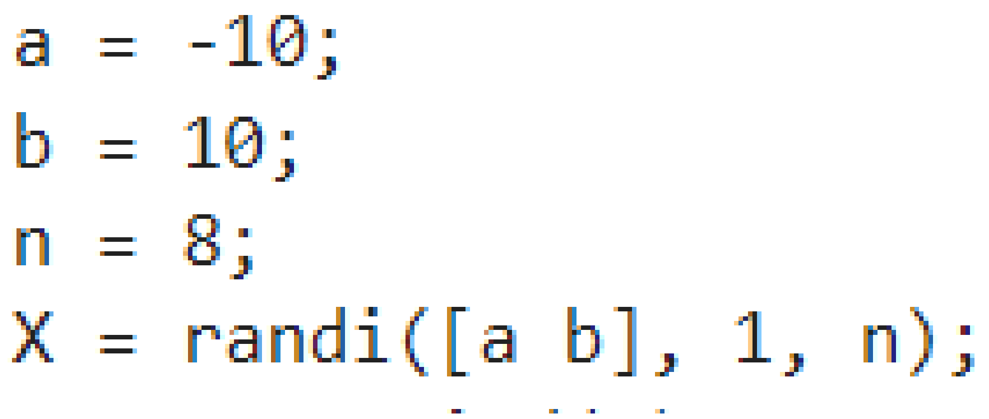


d) Why N should be in the form of $N = 2^L$ for radix-2 FFT algorithms.

Since Radix-2 FFT works by dividing the signal into even and odd samples, all of which are splited into pairs of points, then calculating the DFT for each pair. After that, each two pairs are combined to calculate the DFT of a 4-point signal. The process continues until the DFT of the full signal spectrum is calculated. Since we start at 2 and double each time, this gives the condition on N = 2 ^ L.
This form of solving the problem by breaking it into sub-problems reduces the quadratic complexity into a complexity of nlog(n).

E) i)

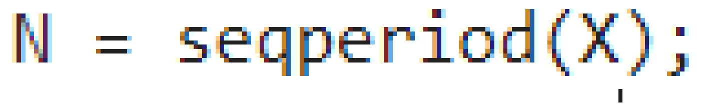
- Firstly, we generate a random signal of the required number of samples **n** with values in the specified range **[a,b].**
- The signal has a period N.



```
a = -10;
b = 10;
n = 8;
X = randi([a b], 1, n);
```

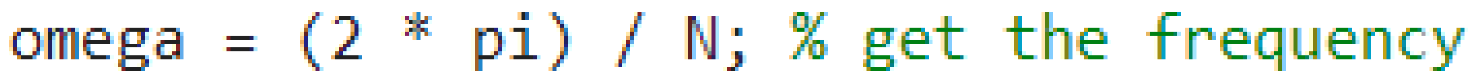
Period and Frequency Calculation

- We ensure the period using the matlab command seqperiod(X)



```
N = seqperiod(X);
```

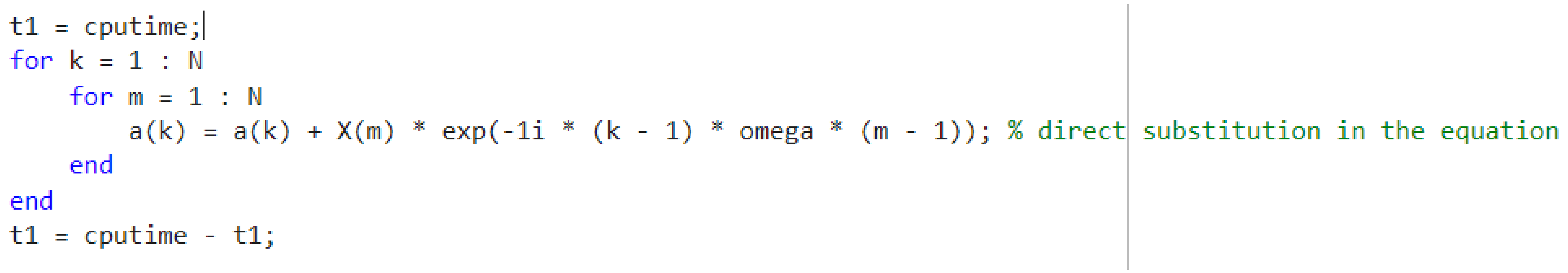
- There are other ways to ensure that the randomly generated signal will have a period N; like for example generating the random signal as a permutation of numbers. That was not required though.
- The Frequency is calculated



```
omega = (2 * pi) / N; % get the frequency
```

ii) Calculation using the equation

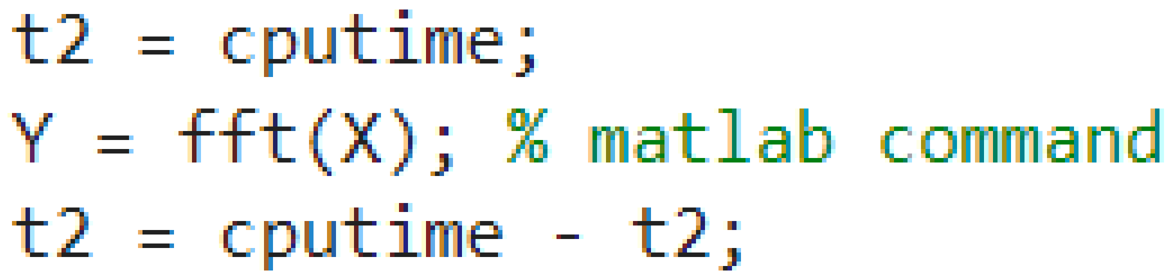
- We implement two nested loops to calculate the DFS coefficients using the equation.



```
t1 = cputime;
for k = 1 : N
    for m = 1 : N
        a(k) = a(k) + X(m) * exp(-1i * (k - 1) * omega * (m - 1)); % direct substitution in the equation
    end
end
t1 = cputime - t1;
```

- CPU time is obtained before and after the command to get the elapsed time t1.

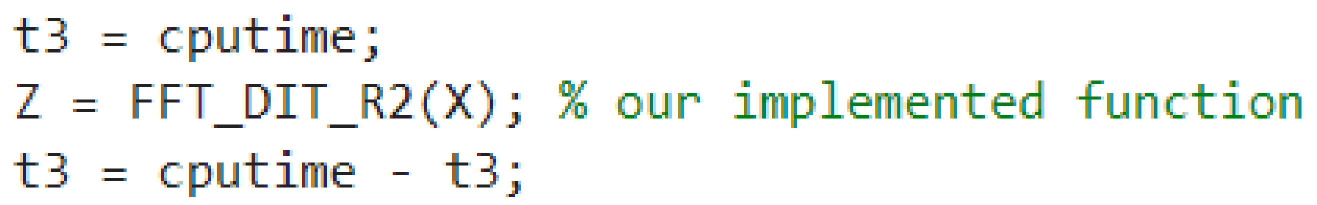
ii) Calculation using Matlab’s FFT



```
t2 = cputime;
Y = fft(X); % matlab command
t2 = cputime - t2;
```

- Computational time is calculated in the same way in time t2.

iii) Calculation using our implementation of the radix-2 DIT FFT



```
t3 = cputime;
Z = FFT_DIT_R2(X); % our implemented function
t3 = cputime - t3;
```

- Computational time is calculated in the same way in time t3.

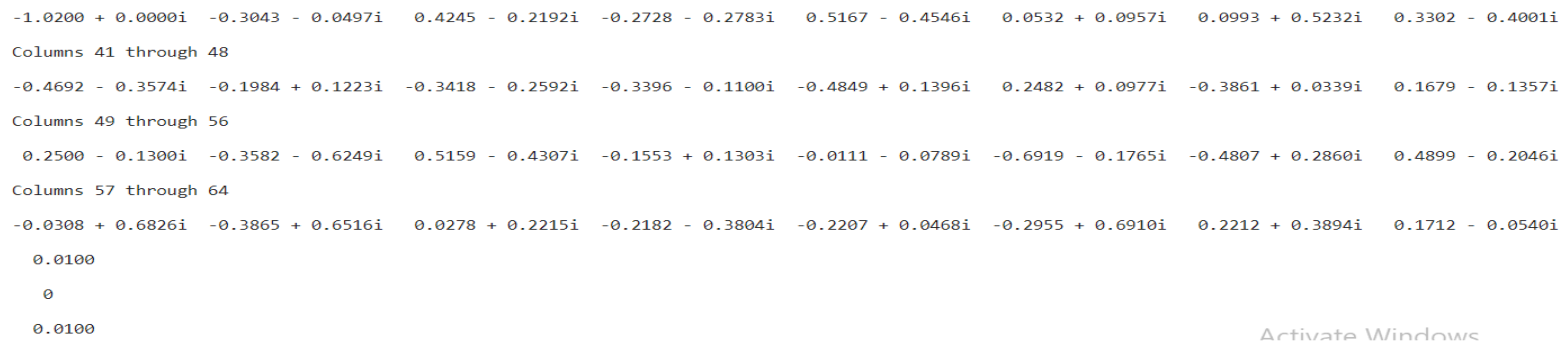
- Then the results are displayed

iv) The results for an 8 sample signal

```
     1    10    10    -7    10    10     0     6

  40.0000 + 0.0000i   0.1924 - 0.8076i   1.0000 -21.0000i -18.1924 +19.1924i   2.0000 - 0.0000i -18.1924 -19.1924i   1.0000 +21.0000i   0.1924 + 0.8076i

  40.0000 + 0.0000i   0.1924 - 0.8076i   1.0000 -21.0000i -18.1924 +19.1924i   2.0000 + 0.0000i -18.1924 -19.1924i   1.0000 +21.0000i   0.1924 + 0.8076i

  40.0000 + 0.0000i   0.1924 - 0.8076i   1.0000 -21.0000i -18.1924 +19.1924i   2.0000 + 0.0000i -18.1924 -19.1924i   1.0000 +21.0000i   0.1924 + 0.8076i

    0.0100

     0

     0
```

Algorithm	ii)	iii)	iv)
Computation time (sec)	0.0100	0.0	0.0

v) The results for a 64 sample signal

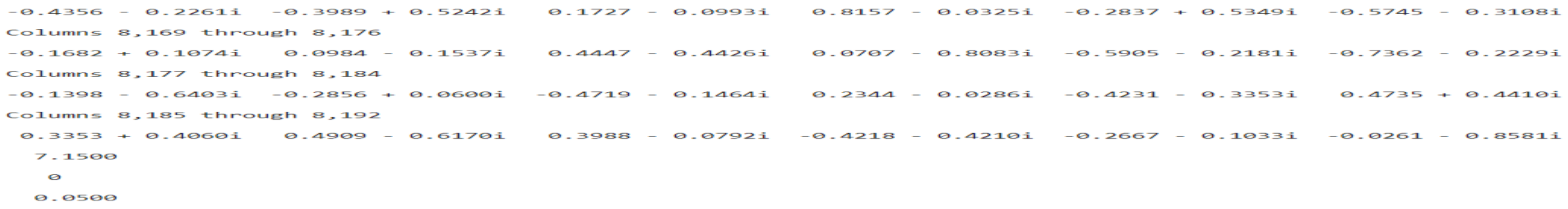


```
 -1.0200 + 0.0000i  -0.3043 - 0.0497i   0.4245 - 0.2192i  -0.2728 - 0.2783i   0.5167 - 0.4546i   0.0532 + 0.0957i   0.0993 + 0.5232i   0.3302 - 0.4001i

 Columns 41 through 48

 -0.4692 - 0.3574i  -0.1984 + 0.1223i  -0.3418 - 0.2592i  -0.3396 - 0.1100i  -0.4849 + 0.1396i   0.2482 + 0.0977i  -0.3861 + 0.0339i   0.1679 - 0.1357i

 Columns 49 through 56

  0.2500 - 0.1300i  -0.3582 - 0.6249i   0.5159 - 0.4307i  -0.1553 + 0.1303i  -0.0111 - 0.0789i  -0.6919 - 0.1765i  -0.4807 + 0.2860i   0.4899 - 0.2046i

 Columns 57 through 64

 -0.0308 + 0.6826i  -0.3865 + 0.6516i   0.0278 + 0.2215i  -0.2182 - 0.3804i  -0.2207 + 0.0468i  -0.2955 + 0.6910i   0.2212 + 0.3894i   0.1712 - 0.0540i

   0.0100

     0

   0.0100
```

Algorithm	ii)	iii)	iv)
Computation time (sec)	0.0100	0.0	0.0100

v) The results for a 8192 (2^12) sample signal

- We thought of displaying the results for a bigger sample to better show the efficiency of our implemented algorithm.



```
-0.4356 - 0.2261i  -0.3989 + 0.5242i   0.1727 - 0.0993i   0.8157 - 0.0325i  -0.2837 + 0.5349i  -0.5745 - 0.3108i
Columns 8,169 through 8,176
-0.1682 + 0.1074i   0.0984 - 0.1537i   0.4447 - 0.4426i   0.0707 - 0.8083i  -0.5905 - 0.2181i  -0.7362 - 0.2229i
Columns 8,177 through 8,184
-0.1398 - 0.6403i  -0.2856 + 0.0600i  -0.4719 - 0.1464i   0.2344 - 0.0286i  -0.4231 - 0.3353i   0.4735 + 0.4410i
Columns 8,185 through 8,192
 0.3353 + 0.4060i   0.4909 - 0.6170i   0.3988 - 0.0792i  -0.4218 - 0.4210i  -0.2667 - 0.1033i  -0.0261 - 0.8581i
  7.1500
   0
  0.0500
```

Algorithm	ii)	iii)	iv)
Computation time (sec)	7.1500	0.0	0.0500

- As n gets larger, the computational differences start showing between different algorithms. Still, though, our implementation gives an acceptable complexity.

THANK YOU