# Question 4

This script implements the double transform algorithm to apply the discrete Fourier transform to two real, N-point sequences using one complex N-point transform.

In [ ]:
```
import numpy as np

from scipy.fft import fft, ifft
```

## Part A: Equal length sequences

In [ ]:
```
# Define the vectors
x = np.array([1, 2, 4, 4, 5, 3, 7, 8])
y = np.array([1, 5, 3, 1, 3, 5, 3, 7])

print("DFT of x:", fft(x))
print("DFT of y:", fft(y))
```

```
DFT of x: [34.         -0.j          -1.87867966+6.53553391j -5.         +7.j
 -6.12132034+0.53553391j  0.         -0.j          -6.12132034-0.53553391j
 -5.         -7.j          -1.87867966-6.53553391j]
DFT of y: [28.         -0.j           2.24264069+4.24264069j -2.         -2.j
 -6.24264069+4.24264069j -8.         -0.j          -6.24264069-4.24264069j
 -2.         +2.j           2.24264069-4.24264069j]
```

In [ ]:
```
# Combine x and y into a single complex vector and apply the FFT
Z = fft(np.array([a+b*1j for a, b in zip(x, y)]))

print("DFT of z:", Z)
```

```
DFT of z: [ 34.         +28.j           -6.12132034 +8.77817459j
  -3.          +5.j          -10.36396103 -5.70710678j
   0.          -8.j           -1.87867966 -6.77817459j
  -7.          -9.j            2.36396103 -4.29289322j]
```

In [ ]:
```
# Extract the odd and even components of the real and imaginary parts of Z

def ev(H: np.array) -> np.array:
    """
    Returns the even component of the given sequence `H`.
    """
    H_minus = np.concatenate([H[:1], H[-1:0:-1]])
    return 0.5 * (H + H_minus)

def od(H: np.array) -> np.array:
    """
    Returns the odd component of the given sequence `H`.
    """
    H_minus = np.concatenate([H[:1], H[-1:0:-1]])
    return 0.5 * (H - H_minus)

print(f"{ev(np.real(Z)) = }")
print(f"{od(np.imag(Z)) = }")
print(f"{od(np.real(Z)) = }")
print(f"{ev(np.imag(Z)) = }")
```

```
ev(np.real(Z)) = array([34.        , -1.87867966, -5.        , -6.12132034,  0.
,
       -6.12132034, -5.        , -1.87867966])
od(np.imag(Z)) = array([ 0.        ,  6.53553391,  7.        ,  0.53553391,  0.
,
       -0.53553391, -7.        , -6.53553391])
od(np.real(Z)) = array([ 0.        , -4.24264069,  2.        , -4.24264069,  0.
,
        4.24264069, -2.        ,  4.24264069])
ev(np.imag(Z)) = array([28.        ,  2.24264069, -2.        , -6.24264069, -8.
,
       -6.24264069, -2.        ,  2.24264069])
```

In [ ]:
```
# Finally, reconstruct the DFTs of x and y
X = ev(np.real(Z)) + 1j * od(np.imag(Z))
Y = ev(np.imag(Z)) - 1j * od(np.real(Z))

print(f"{X = }")
print(f"{Y = }")
```

```
X = array([34.        +0.j        , -1.87867966+6.53553391j,
       -5.        +7.j        , -6.12132034+0.53553391j,
        0.        +0.j        , -6.12132034-0.53553391j,
       -5.        -7.j        , -1.87867966-6.53553391j])
Y = array([28.        +0.j        ,  2.24264069+4.24264069j,
       -2.        -2.j        , -6.24264069+4.24264069j,
       -8.        +0.j        , -6.24264069-4.24264069j,
       -2.        +2.j        ,  2.24264069-4.24264069j])
```

In [ ]:
```
# Inverse Fourier transform X and Y to prove that they are correct
print(f"{ifft(X) = }")
print(f"{ifft(Y) = }")
```

```
ifft(X) = array([1.+0.j, 2.+0.j, 4.+0.j, 4.+0.j, 5.+0.j, 3.+0.j, 7.+0.j, 8.+0.j])
ifft(Y) = array([1.+0.j, 5.+0.j, 3.+0.j, 1.+0.j, 3.+0.j, 5.+0.j, 3.+0.j, 7.+0.j])
```

## Part B: Unequal length sequences

In [ ]:
```
# Define the vectors
x = np.array([1, 2, 4, 4, 5, 3, 7, 8]) # this is the same x vector as part (a)
y = np.array([1, 5, 3, 1, 3, 5, 3, 0]) # this y vector is already zero-padded

print("DFT of x:", fft(x))
print("DFT of y:", fft(y))
```

```
DFT of x: [34.        -0.j          -1.87867966+6.53553391j -5.        +7.j
 -6.12132034+0.53553391j  0.        -0.j          -6.12132034-0.53553391j
 -5.        -7.j          -1.87867966-6.53553391j]
DFT of y: [21.        -0.j          -2.70710678-0.70710678j -2.        -9.j
 -1.29289322-0.70710678j -1.        -0.j          -1.29289322+0.70710678j
 -2.        +9.j          -2.70710678+0.70710678j]
```

In [ ]:
```
# Combine x and y into a single complex vector and apply the FFT
Z = fft(np.array([a+b*1j for a, b in zip(x, y)]))

print("DFT of z:", Z)
```

```
DFT of z: [ 34.        +21.j          -1.17157288 +3.82842712j
   4.         +5.j          -5.41421356 -0.75735931j
   0.         -1.j          -6.82842712 -1.82842712j
 -14.         -9.j          -2.58578644 -9.24264069j]
```

In [ ]:
```
# Extract the odd and even components of the real and imaginary parts of Z
print(f"{ev(np.real(Z)) = }")
print(f"{od(np.imag(Z)) = }")
print(f"{od(np.real(Z)) = }")
print(f"{ev(np.imag(Z)) = }")
```

```
ev(np.real(Z)) = array([34.        , -1.87867966, -5.        , -6.12132034,  0.
,
       -6.12132034, -5.        , -1.87867966])
od(np.imag(Z)) = array([ 0.        ,  6.53553391,  7.        ,  0.53553391,  0.
,
       -0.53553391, -7.        , -6.53553391])
od(np.real(Z)) = array([ 0.        ,  0.70710678,  9.        ,  0.70710678,  0.
,
       -0.70710678, -9.        , -0.70710678])
ev(np.imag(Z)) = array([21.        , -2.70710678, -2.        , -1.29289322, -1.
,
       -1.29289322, -2.        , -2.70710678])
```

In [ ]:
```python
# Finally, reconstruct the DFTs of x and y
X = ev(np.real(Z)) + 1j * od(np.imag(Z))
Y = ev(np.imag(Z)) - 1j * od(np.real(Z))

print(f"{X = }")
print(f"{Y = }")
```

```
X = array([34.        +0.j        , -1.87867966+6.53553391j,
       -5.        +7.j        , -6.12132034+0.53553391j,
        0.        +0.j        , -6.12132034-0.53553391j,
       -5.        -7.j        , -1.87867966-6.53553391j])
Y = array([21.        +0.j        , -2.70710678-0.70710678j,
       -2.        -9.j        , -1.29289322-0.70710678j,
       -1.        +0.j        , -1.29289322+0.70710678j,
       -2.        +9.j        , -2.70710678+0.70710678j])
```

In [ ]:
```python
# Inverse Fourier transform X and Y to prove that they are correct
print(f"{ifft(X) = }")
print(f"{ifft(Y) = }")
```

```
ifft(X) = array([1.+0.j, 2.+0.j, 4.+0.j, 4.+0.j, 5.+0.j, 3.+0.j, 7.+0.j, 8.+0.j])
ifft(Y) = array([ 1.00000000e+00+0.j,  5.00000000e+00+0.j,  3.00000000e+00+0.j,
        1.00000000e+00+0.j,  3.00000000e+00+0.j,  5.00000000e+00+0.j,
        3.00000000e+00+0.j, -1.11022302e-16+0.j])
```