Assignment 7 Description:

Write the FFT and inverse FFT function, test the code and then time it.

If you have implemented the algorithm correctly, a simple way to confirm it is working is to show that

P = FFT(FFT(P, W1), W2)/n, where P is the polynomial, W1 is the omega array (primitive roots of unity) and W2 is the inverse array (primitive roots of unity where the imaginary part is negated) and n is the number of terms in P.  You are showing that when you take the FFT then the inverse FFT you get the same answer returned.

The following file prints the inputs and outputs of the FFT function as it evaluates a simple polynomial and may be useful when debugging.  This is the code that is executed to produce the output in the file

n = 8

p = [0, 1, 2, 3, 4, 5, 6, 7]

print("Forward FFT")

sol=fft([complex(p[i],0) for i in range(0,n)], getV(n, +1), n)

print("Inverse FFT")

back=[s/8 for s in fft(sol, getV(n, -1), n)]

print("Answer")

print(back)

[FFT run.txtPreview the document](https://usu.instructure.com/courses/565660/files/77116972/download?wrap=1)

Notice that the original P values are

[0j, (1+0j), (2+0j), (3+0j), (4+0j), (5+0j), (6+0j), (7+0j)]

and the answer back is:

[1.1102230246251565e-16j, (1-1.6653345369377348e-16j), (2+3.2162452993532727e-16j), (3.0000000000000004+1.6653345369377348e-16j), (4-1.1102230246251565e-16j), (5-1.6653345369377348e-16j), (6-3.2162452993532727e-16j), (7+1.6653345369377348e-16j)]

These results are the same within 15 significant digits.

Once it is working correctly, generate a graph of problem size (n from 128 to how far you can go, for n is a power of two) vs. run time. Plot the results on a log-log graph. You should see a nearly straight line because the run time is O(n log n)