1. Done
2. Done
3. I have had significant issues in understanding how to multiply to receive the end polynomial, I had started this implementation the day it was due and was unable to make it to TA hours, thus I only completed a program that turns a given ‘polynomial’ into its fft set.
4. The Theta times for the different implementations should be something like the following:

Standard = theta(n^2)

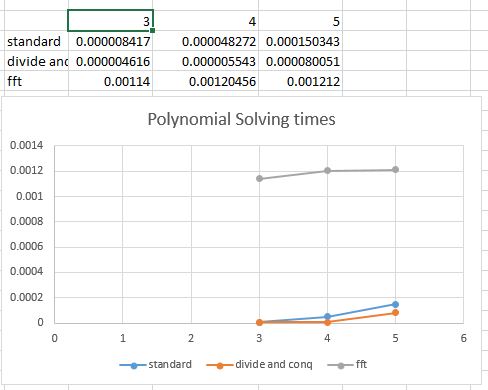
Divide and conquer = 4T(n/2) + theta(n)

Fourier =theta(nlog(n))

1. As explained, my FFT had some issues due to time constraints, my other two methods also require a lot of configuring for their setup, so I would sometimes receive really strange numbers under specific conditions (where I had not properly setup my dependencies).

Pics of results:

1. I chose a scatter plot, as it can illustrate a trend easily, and we have a trend displaying already within our program:



The FFT appears to start very slowly, but it doesn’t grow at the rate that the others appear to grow at. If this chart had continued, it would show that the FFT represents a logarithmic scale, and that the divide and conquer shows a growth rate slower than, but similar to our standard multiplication.

1. For smaller plots, it appears that divide and conquer has the advantage, but as the number of arguments grows, the less efficient its algorithm gets compared to the FFT, because of my implementation at the moment, we would need to do two runs of FFT to get vectors to compare and multiply, so i’ll give a ballpark of 15 to 20 being the moment that FFT starts becoming more and more advantageous due to its logarithmic scaling