FFT Lecture – Homework problem

1) In class we examined the FFT algorithm for computing the DFT. As a reminder, we saw that if the number of samples N is a power of 2, i.e. $N = 2^{\nu}$, then we could develop a decimation-in-time FFT algorithm that gave the result

$$X(k) = X_e(k) + W_N^k X_o(k) \qquad \text{(Eq 1)}$$

where W_N^k is as defined in class, $X_e(k)$ is an N/2 DFT of the even samples of x(n), and $X_o(k)$ is an N/2 –point DFT of the odd samples of x(n). Because we subdivide by factors of 2, this algorithm is called a *radix-2* FFT.

While less common, it's possible to define FFT algorithms that are not power-of-2 based. Assume that you have signals where N is a power of 3, i.e. $N = 3^{\nu}$ (for example, we might want to take an 81-point DFT, so ν=4). Here, we might like to split the data by thirds to find a *radix-3* FFT algorithm.

Following a similar logic to that outlined in slides 6-7 of today's PPT slides (screenshots from the van Veen lecture), derive an equation similar to Eq 1 that lets you find *X(k)* in terms of three N/3 - point DFTs.

2) For this problem, we'll assume we are using a radix-2 FFT such as that discussed in class.

One disadvantage of the FFT is that if we want to compute the DFT of a signal whose length is not a power of two, we need to add zeros at the end of the signal (called 'zero-padding') until the signal is a power of 2 long. Thus to compute an N=1100 point DFT, we first add (2048-1100) additional zeros and then perform a 2048-point FFT. Clearly, the fact that we are computing the result at extra frequencies creates extra computational cost.

Based on results from lecture, roughly estimate the speedup for using a 2048 point FFT to compute an N=1100 point DFT.