Jacob McConnell 260706620
Aleks Muaraskas 260718389

Assignment 2
ECSE 316

Design:
The design of our FFT algorithm was an implementation of the Cooley Tukey DFFT algorithm. The way it works is by splitting up the problem recursively. First the problem is split into the even and odd indexed elements of the array. Then we solve the problem recursively for the even and odd halves of the problem. Finally we solve the whole problem by adding e^(-2i*pi*k/N)*(kth element of the odd elements solution) the kth element of the even solution for k 0,1,2,..N/2 at the same time we fill out the solution for N/2 to N by subtracting e^(-2i*pi*k/N)*(kth element of the odd solution) from the kth element of the even solution. For obvious reasons this means that the algorithm only works on arrays that are of sizes that are powers of 2 so you can always divide by two till you get to a base case. You can use a base case 1 and return just the one element long list but this is not the most effiecient.
 For the inverse Discrete Fourier transform we used the property that the IDFT of X is equal to 1/N time the conjugate of the Fourier transform of the conjugate of X. The formula is shown below. This way we could use the fast FFT algorithm we had already written to solve for the inverse problem.

$$\mathcal{F}^{-1}(\mathbf{x}) = \frac{1}{N}\mathcal{F}(\mathbf{x}^*)^*$$

(https://en.wikipedia.org/wiki/Discrete_Fourier_transform#Expressing_the_inverse_DFT_in_terms_of_the_DFT)

Testing: Testing was done both on the image and on simple arrays. We compared the results of our algorithms with library functions to test their correctness. Two arrays I used for testing were the numbers 0-31 inclusive and 0-29 with two zeros at the end. This was small enough to compare examples by eye which made debugging possible.

Analysis:
The complexity of the slow DFT is O(n^2)
The complexity of the FFT is O(n*log(n))

The complexity of the slow 2d DFT is O(n^4)
The complexity of the 2d FF is O(n^2*log(n)*log(n))

Experiment: