3a_test1

April 1, 2019

```
In [1]: from cmath import exp, pi

        def fft(x):
            N = len(x)
            if N <= 1: return x
            even_part = fft(x[0::2])
            odd_part =  fft(x[1::2])
            T= [exp(-2j*pi*k/N)*odd_part[k] for k in range(N//2)]
            return [even_part[k] + T[k] for k in range(N//2)] + \
                   [even_part[k] - T[k] for k in range(N//2)]
```

```
In [2]: from numpy import array
```

```
In [3]: import numpy as np
```

```
In [4]: a = array([0, 0.7071, 1, 0.7071, 0, -0.7071, -1, -0.7071])
```

0.1 Now lets see what output is using our FFT function

```
In [5]: array(fft(a))
```

```
Out[5]: array([ 0.00000000e+00+0.00000000e+00j,  1.22464680e-16-3.99998082e+00j,
                0.00000000e+00+0.00000000e+00j,  9.95799250e-17+1.91800920e-05j,
                0.00000000e+00+0.00000000e+00j,  1.22464680e-16-1.91800920e-05j,
                0.00000000e+00+0.00000000e+00j, -3.44509285e-16+3.99998082e+00j])
```

0.2 Now lets see what output is using Numpy's FFT

```
In [6]: np.fft.fft(a)
```

```
Out[6]: array([0.+0.00000000e+00j, 0.-3.99998082e+00j, 0.+0.00000000e+00j,
               0.+1.91800920e-05j, 0.+0.00000000e+00j, 0.-1.91800920e-05j,
               0.-0.00000000e+00j, 0.+3.99998082e+00j])
```

0.3 Now lets check if our FFT function is element wise equal (within tolerance) to Numpy's FFT

```
In [7]: np.allclose(array(fft(a)), np.fft.fft(a))
```

```
Out[7]: True
```

0.4 Now lets see what is the difference between our FFT output and Numpy's FFT

```
In [8]: array(fft(a)) - np.fft.fft(a)
```

```
Out[8]: array([ 0.00000000e+00+0.00000000e+00j,  1.22464680e-16+0.00000000e+00j,
                0.00000000e+00+0.00000000e+00j,  9.95799250e-17+2.22044605e-16j,
                0.00000000e+00+0.00000000e+00j,  1.22464680e-16+0.00000000e+00j,
                0.00000000e+00+0.00000000e+00j, -3.44509285e-16+0.00000000e+00j])
```

0.5 Hurray!!! , The both outputs are equal (almost, difference is in around 10ˆ-17)