CS 405/511: Algorithm Analysis
Winter, 2013
Ternary FFT, due March 8, 2013

In lecture we discussed how to construct a $\Theta(n \lg n)$ algorithm for the discrete Fourier transform of a coefficient vector by breaking the underlying polynomial into two components with interlaced coefficients. This method is appropriate for coefficient vectors with length equal to a power of 2. You are to develop a similar algorithm for coefficient vectors of length equal to a power of 3. Padding the input to the next power of 2 would trivially generate such an algorithm, but you are *not* to use that expedient for this exercise. Instead, you are to invent an approach that divides the underlying polynomial into three components with appropriately interlaced components. This extension is relatively straightforward once you understand the process that delivered the binary version. Specifically, you need to submit the following items.

1. A short explanation of your ternary approach, including a discussion as to why the algorithm is $\Theta(n \lg n)$.
2. An observation that enables a slight modification to produce the reverse transform. This discovery is again very similar to that of the binary case.
3. Two programs: the ternary FFT and its inverse (reverse) transform.
4. A run showing the operation on the data in file ternary81.txt. The output should follow the example below, which processes a vector of length 27, available from file ternary27.txt.

```
>> a
a =
   83.0829
   58.5264
   54.9724
   91.7194
   28.5839
   75.7200
   75.3729
   38.0446
   56.7822
    7.5854
    5.3950
   53.0798
   77.9167
   93.4011
   12.9906
   56.8824
   46.9391
    1.1902
   33.7123
   16.2182
   79.4285
   31.1215
   52.8533
   16.5649
   60.1982
   26.2971
   65.4079

>> b = ternaryFFT(a, 1);

>> c = reverseTernaryFFT(b);
```

My algorithm accepts a coefficient column vector and a mode parameter. Mode 1 generates the FFT; mode -1 generates the reverse transform, except for the final division by n, the vector length. Consequently, reverseTernaryFFT simply calls ternaryFFT(**b**, -1) and divides the result by the length of **b**. Although **a** is a real vector, both **b** and **c** are complex. Of course, **c** should have zero imaginary part because it should reproduce **a**. The utility printParallel prints the rows of the various vectors for easy comparison. For a correct implementation, we verify that **a** = real(**c**). The second two columns constitute the real and imaginary parts of the transform.

```
>> printParallel([a real(b) imag(b) real(c) imag(c)]);

   83.0829  1299.9869     0.0000    83.0829     0.0000
   58.5264   120.2144   103.6418    58.5264     0.0000
   54.9724   115.1924    57.4766    54.9724     0.0000
   91.7194  -127.6611    78.2311    91.7194     0.0000
   28.5839   193.2861   -80.8503    28.5839     0.0000
   75.7200    36.4226    60.5759    75.7200    -0.0000
   75.3729  -103.0484   -68.4275    75.3729     0.0000
   38.0446   104.1139    66.6526    38.0446     0.0000
   56.7822   -23.6266  -105.4736    56.7822    -0.0000
    7.5854   126.3941   -43.1954     7.5854     0.0000
    5.3950   115.3098   -63.8910     5.3950     0.0000
   53.0798    56.7067   175.0255    53.0798     0.0000
   77.9167    14.0347  -125.0840    77.9167    -0.0000
   93.4011  -155.7129   -41.5878    93.4011    -0.0000
   12.9906  -155.7129    41.5878    12.9906     0.0000
   56.8824    14.0347   125.0840    56.8824    -0.0000
   46.9391    56.7067  -175.0255    46.9391    -0.0000
    1.1902   115.3098    63.8910     1.1902     0.0000
   33.7123   126.3941    43.1954    33.7123     0.0000
   16.2182   -23.6266   105.4736    16.2182     0.0000
   79.4285   104.1139   -66.6526    79.4285    -0.0000
   31.1215  -103.0484    68.4275    31.1215     0.0000
   52.8533    36.4226   -60.5759    52.8533    -0.0000
   16.5649   193.2861    80.8503    16.5649     0.0000
   60.1982  -127.6611   -78.2311    60.1982    -0.0000
   26.2971   115.1924   -57.4766    26.2971     0.0000
   65.4079   120.2144  -103.6418    65.4079    -0.0000
```