Example trace of FFT algorithm:
P has coefficients 0 1 2 3, while Q has coefficients 10 11 12 13
Each complex number is printed as #c(real-part complex-part)
The FFT algorithm has three arguments
FFT(int size, int power, Complex[] coefficients), where size is the number of terms, power is the power of omega, and coefficients are the coefficients of the polynomial (if used as FFT, or the values if used FFT-1).
Here is the first run evaluating P:

```
0[1]: (FFT 8 1 (0 1 2 3 0 0 0 0))
  1[1]: (FFT 4 2 (0 2 0 0))
    2[1]: (FFT 2 4 (0 0))
      3[1]: (FFT 1 8 (0))
      3[1]: returned (0)
      3[1]: (FFT 1 8 (0))
      3[1]: returned (0)
    2[1]: returned (#c(0.0d0 0.0d0) #c(0.0d0 0.0d0))
    2[1]: (FFT 2 4 (2 0))
      3[1]: (FFT 1 8 (2))
      3[1]: returned (2)
      3[1]: (FFT 1 8 (0))
      3[1]: returned (0)
    2[1]: returned (#c(2.0d0 0.0d0) #c(2.0d0 0.0d0))
  1[1]: returned
        (#c(2.0d0 0.0d0) #c(1.2246063538223773d-16 2.0d0) #c(-2.0d0 0.0d0)  #c(-1.2246063538223773d-16 -2.0d0))
  1[1]: (FFT 4 2 (1 3 0 0))
    2[1]: (FFT 2 4 (1 0))
      3[1]: (FFT 1 8 (1))
      3[1]: returned (1)
      3[1]: (FFT 1 8 (0))
      3[1]: returned (0)
    2[1]: returned (#c(1.0d0 0.0d0) #c(1.0d0 0.0d0))
    2[1]: (FFT 2 4 (3 0))
      3[1]: (FFT 1 8 (3))
      3[1]: returned (3)
      3[1]: (FFT 1 8 (0))
      3[1]: returned (0)
    2[1]: returned (#c(3.0d0 0.0d0) #c(3.0d0 0.0d0))
  1[1]: returned
        (#c(4.0d0 0.0d0) #c(1.0000000000000002d0 3.0d0) #c(-2.0d0 0.0d0)  #c(0.9999999999999998d0 -3.0d0))
 0[1]: returned
       (#c(6.0d0 0.0d0) #c(-1.4142135623730945d0 4.82842712474619d0)
        #c(-2.0d0 -2.0d0) #c(1.4142135623730951d0 0.8284271247461898d0)
        #c(-2.0d0 0.0d0) #c(1.414213562373095d0 -0.8284271247461903d0)
        #c(-1.9999999999999998d0 2.0d0)
        #c(-1.4142135623730956d0 -4.82842712474619d0))
(#c(6.0d0 0.0d0) #c(-1.4142135623730945d0 4.82842712474619d0) #c(-2.0d0 -2.0d0)
 #c(1.4142135623730951d0 0.8284271247461898d0) #c(-2.0d0 0.0d0)
 #c(1.414213562373095d0 -0.8284271247461903d0) #c(-1.9999999999999998d0 2.0d0)
 #c(-1.4142135623730956d0 -4.82842712474619d0))
```

Note that it returns 8 complex numbers, where a complex number is represented as
#C(real-part imag-part)

Next we evaluate the Q polynomial:

```
0[1]: (FFT 8 1 (10 11 12 13 0 0 0 0))
  1[1]: (FFT 4 2 (10 12 0 0))
    2[1]: (FFT 2 4 (10 0))
      3[1]: (FFT 1 8 (10))
      3[1]: returned (10)
      3[1]: (FFT 1 8 (0))
      3[1]: returned (0)
    2[1]: returned (#c(10.0d0 0.0d0) #c(10.0d0 0.0d0))
    2[1]: (FFT 2 4 (12 0))
      3[1]: (FFT 1 8 (12))
      3[1]: returned (12)
      3[1]: (FFT 1 8 (0))
      3[1]: returned (0)
    2[1]: returned (#c(12.0d0 0.0d0) #c(12.0d0 0.0d0))
  1[1]: returned
          (#c(22.0d0 0.0d0) #c(10.0d0 12.0d0) #c(-2.0d0 0.0d0) #c(10.0d0 -12.0d0))
  1[1]: (FFT 4 2 (11 13 0 0))
    2[1]: (FFT 2 4 (11 0))
      3[1]: (FFT 1 8 (11))
      3[1]: returned (11)
      3[1]: (FFT 1 8 (0))
      3[1]: returned (0)
    2[1]: returned (#c(11.0d0 0.0d0) #c(11.0d0 0.0d0))
    2[1]: (FFT 2 4 (13 0))
      3[1]: (FFT 1 8 (13))
      3[1]: returned (13)
      3[1]: (FFT 1 8 (0))
      3[1]: returned (0)
    2[1]: returned (#c(13.0d0 0.0d0) #c(13.0d0 0.0d0))
  1[1]: returned
          (#c(24.0d0 0.0d0) #c(11.0d0 13.0d0) #c(-2.0d0 0.0d0) #c(11.0d0 -13.0d0))
 0[1]: returned
        (#c(46.0d0 0.0d0) #c(8.585786437626906d0 28.970562748477143d0)
         #c(-2.0d0 -2.0d0) #c(11.414213562373098d0 4.970562748477143d0)
         #c(-2.0d0 0.0d0) #c(11.414213562373094d0 -4.970562748477143d0)
         #c(-1.9999999999999998d0 2.0d0)
         #c(8.585786437626902d0 -28.970562748477143d0))
(#c(46.0d0 0.0d0) #c(8.585786437626906d0 28.970562748477143d0) #c(-2.0d0 -2.0d0)
 #c(11.414213562373098d0 4.970562748477143d0) #c(-2.0d0 0.0d0)
 #c(11.414213562373094d0 -4.970562748477143d0) #c(-1.9999999999999998d0 2.0d0)
 #c(8.585786437626902d0 -28.970562748477143d0))
```

We multiply item by item, the two results, then call fft again, using a pre-computed array of powers of 1/w (with the imaginary part negated)

```
0[1]: (FFT 8 1
        (#c(276.0d0 0.0d0) #c(-152.02438661763952d0 0.485281374238582d0)
         #c(0.0d0 8.0d0) #c(12.024386617639516d0 16.485281374238575d0)
         #c(4.0d0 0.0d0) #c(12.024386617639507d0 -16.485281374238575d0)
         #c(-8.881784197001252d-16 -7.999999999999999d0)
         #c(-152.02438661763952d0 -0.48528137423853934d0)))
  1[1]: (FFT 4 2
          (#c(276.0d0 0.0d0) #c(0.0d0 8.0d0) #c(4.0d0 0.0d0)
           #c(-8.881784197001252d-16 -7.999999999999999d0)))
    2[1]: (FFT 2 4 (#c(276.0d0 0.0d0) #c(4.0d0 0.0d0)))
      3[1]: (FFT 1 8 (#c(276.0d0 0.0d0)))
      3[1]: returned (#c(276.0d0 0.0d0))
      3[1]: (FFT 1 8 (#c(4.0d0 0.0d0)))
      3[1]: returned (#c(4.0d0 0.0d0))
    2[1]: returned (#c(280.0d0 0.0d0) #c(272.0d0 0.0d0))
    2[1]: (FFT 2 4
            (#c(0.0d0 8.0d0) #c(-8.881784197001252d-16 -7.999999999999999d0)))
      3[1]: (FFT 1 8 (#c(0.0d0 8.0d0)))
      3[1]: returned (#c(0.0d0 8.0d0))
      3[1]: (FFT 1 8 (#c(-8.881784197001252d-16 -7.999999999999999d0)))
      3[1]: returned (#c(-8.881784197001252d-16 -7.999999999999999d0))
    2[1]: returned
            (#c(-8.881784197001252d-16 8.881784197001252d-16)
             #c(8.881784197001252d-16 16.0d0))
  1[1]: returned
          (#c(280.0d0 8.881784197001252d-16) #c(288.0d0 9.150666335777657d-17)
           #c(280.0d0 -8.881784197001252d-16) #c(256.0d0 -9.150666335777657d-17))
  1[1]: (FFT 4 2
          (#c(-152.02438661763952d0 0.485281374238582d0)
           #c(12.024386617639516d0 16.485281374238575d0)
           #c(12.024386617639507d0 -16.485281374238575d0)
           #c(-152.02438661763952d0 -0.48528137423853934d0)))
    2[1]: (FFT 2 4
            (#c(-152.02438661763952d0 0.485281374238582d0)
             #c(12.024386617639507d0 -16.485281374238575d0)))
      3[1]: (FFT 1 8 (#c(-152.02438661763952d0 0.485281374238582d0)))
      3[1]: returned (#c(-152.02438661763952d0 0.485281374238582d0))
      3[1]: (FFT 1 8 (#c(12.024386617639507d0 -16.485281374238575d0)))
      3[1]: returned (#c(12.024386617639507d0 -16.485281374238575d0))
    2[1]: returned
            (#c(-140.0d0 -15.999999999999993d0)
             #c(-164.04877323527904d0 16.970562748477157d0))
    2[1]: (FFT 2 4
            (#c(12.024386617639516d0 16.485281374238575d0)
             #c(-152.02438661763952d0 -0.48528137423853934d0)))
      3[1]: (FFT 1 8 (#c(12.024386617639516d0 16.485281374238575d0)))
      3[1]: returned (#c(12.024386617639516d0 16.485281374238575d0))
      3[1]: (FFT 1 8 (#c(-152.02438661763952d0 -0.48528137423853934d0)))
      3[1]: returned (#c(-152.02438661763952d0 -0.48528137423853934d0))
    2[1]: returned
            (#c(-140.0d0 16.000000000000036d0)
             #c(164.04877323527904d0 16.970562748477114d0))
  1[1]: returned
```

```
     (#c(-280.0d0 4.263256414560601d-14)
      #c(-147.0782104868019d0 -147.07821048680188d0)
      #c(0.0d0 -32.00000000000003d0)
      #c(-181.01933598375618d0 181.0193359837562d0))
 0[1]: returned
    (#c(0.0d0 4.3520742565306136d-14) #c(80.0d0 9.150666335777657d-17)
     #c(247.99999999999997d0 -2.8475485858159304d-15)
     #c(512.0d0 -9.150666335777657d-17) #c(560.0d0 -4.1744385725905886d-14)
     #c(496.0d0 9.150666335777657d-17) #c(312.0d0 1.07119174641568d-15)
     #c(0.0d0 -9.150666335777657d-17))
```

Note that with the inverse FFT we must divide by n
Looking at the real parts of these numbers, we get the same result!

```
FFT RESULT=(0.0d0 10.0d0 30.999999999999996d0 64.0d0 70.0d0 62.0d0 39.0d0 0.0d0)
SCHOOL algorithm  RESULT=(0 10 31 64 70 62 39 0)
```