Appendix 9B. The improvement in speed is dramatic. A dataset containing 10^6 points requires $\approx 6 \times 10^6$ multiplications rather than $\approx 10^{12}$. Because we will use this algorithm to study diffraction and other phenomena, and because coding this algorithm is nontrivial, we have provided an implementation of the FFT in the Open Source Physics numerics package. We can use this `FFT` class to transform between time and frequency or position and wavenumber. The `FFTApp` program shows how the `FFT` class is used.

Listing 9.7 The `FFTApp` program computes the fast Fourier transform of a function and displays the coefficients.

```
package org.opensourcephysics.sip.ch09;
import java.text.DecimalFormat;
import org.opensourcephysics.controls.*;
import org.opensourcephysics.numerics.FFT;

public class FFTApp extends AbstractCalculation {
   public void calculate() {
      // output format
      DecimalFormat decimal = new DecimalFormat("0.0000");
      // number of Fourier coefficients
      int N = 8;
      // array that will be transformed
      double[] z = new double[2*N];
      // FFT implementation for N points
      FFT fft = new FFT(N);
      // mode or harmonic of e^(i*x)
      int mode = control.getInt("mode");
      double x = 0, delta = 2*Math.PI/N;
      for(int i = 0;i<N;i++) {
         z[2*i] = Math.cos(mode*x);     // real component of e^(i*mode*x)
         z[2*i+1] = Math.sin(mode*x); // imaginary component
         x += delta;                    // increase x
      }
      // transform data; data will be in wrap-around order
      fft.transform(z);
      for(int i = 0;i<N;i++) {
         System.out.println("index = "+i+"\t real = "
                           +decimal.format(z[2*i])+"\t imag = "
                           +decimal.format(z[2*i+1]));
      }
   }

   public void reset() {
      control.setValue("mode", -1);
   }

   public static void main(String[] args) {
      CalculationControl.createApp(new FFTApp());
   }
}
```

The `FFT` class replaces the data in the input array with transformed values. (Make an array copy if you need to retain the original data.) Because the transformation assumes N complex data points, and Java does not support a primitive complex data type, the input array has length $2N$. The real part of the nth data point is located in array element $2n$ and

the imaginary part is in element $2n + 1$. The transformed data maintains this same ordering of real and imaginary parts. We test the `FFT` class in Problem 9.12.

Problem 9.12 The `FFT` class

(a) The `FFTApp` initializes the array that will be transformed by evaluating the following complex function:

$$f_n = f(n\Delta) = e^{in\Delta} = \cos n\Delta + i \sin n\Delta, \tag{9.38}$$

where n is an integer and $\Delta = 2\pi/N$ is the interval between data points. We refer to n as the `mode` variable in the program. What happens to the Fourier component if the phase of the complex exponential is shifted by α? In other words, what happens if the data is initialized using $f_n = f(n\Delta) = e^{in\Delta+\alpha}$?

(b) Modify and run `FFTApp` to show that the fast Fourier transform produces a single component only if the grid contains an integer number of wavelengths.

(c) Change the number of grid points N and show that the value of the nonzero Fourier coefficient is equal to N if the input function is (9.38). Note that some other FFT implementations normalize the result by dividing by N.

(d) Show that the original function can be recovered by invoking the `FFT.inverse` method. ■

Problem 9.13 Negative frequencies

(a) Use (9.38) as the input function and show that `FFTApp` produces the same Fourier coefficients if $\omega = 2\pi/(N\Delta)$ or $\omega = -2\pi/(N\Delta)$. Repeat for $\omega = 4\pi/(N\Delta)$ and $\omega = -4\pi/(N\Delta)$.

(b) Compute the Fourier coefficients using $f_n = f(n\Delta) = \cos n\Delta$, where $n = 0, 1, 2, \ldots, N-1$. Repeat using a sine function. Interpret your results using

$$\cos\theta = \frac{e^{i\theta} + e^{-i\theta}}{2} \tag{9.39a}$$

$$\sin\theta = \frac{e^{i\theta} - e^{-i\theta}}{2}. \tag{9.39b}$$

■

As shown in Problem 9.13, the Fourier coefficient indices for $n \geq N/2$ should be interpreted as negative frequencies. The transformed data still contains N frequencies, and these frequencies are still separated by $2\pi/(N\Delta)$.

Because the frequencies switch sign at the array's midpoint index $N/2$, we refer to the transformed data as being in *wrap-around order*. The `toNaturalOrder` method can be used to sort and normalize the Fourier components in order of increasing frequency starting at $\omega_Q = -\pi/\Delta$ and continuing to $\omega_Q = \pi/\Delta((N-2)/N)$ if N is even and continuing to $\omega_Q = \pi/\Delta$ if N is odd.