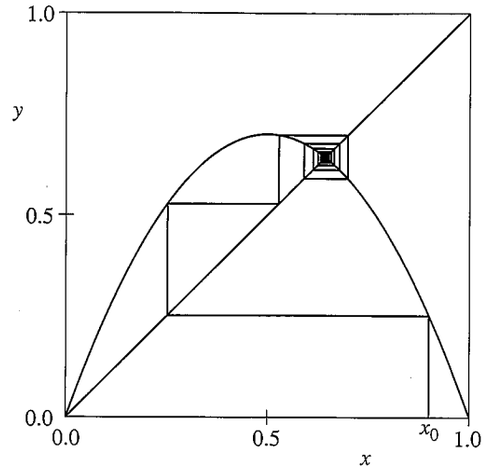

Figure 6.3 Graphical representation of the iteration of the logistic map (6.5) with $r = 0.7$ and $x_0 = 0.9$. Note that the graphical solution converges to the fixed point $x^* \approx 0.643$.

An inspection of $f(x)$ in Figure 6.3 shows that $x = 0$ is unstable because the slope of $f(x)$ at $x = 0$ is greater than unity. In contrast, the magnitude of the slope of $f(x)$ at $x = x^* \approx 0.643$ is less than unity and this fixed point is stable. In Appendix 6A we show that

$$x^* = 0 \text{ is stable for } 0 < r < 1/4, \tag{6.6a}$$

and

$$x^* = 1 - \frac{1}{4r} \text{ is stable for } 1/4 < r < 3/4. \tag{6.6b}$$

Thus for $0 < r < 3/4$, the behavior after many iterations is known.

What happens if r is greater than $3/4$? We found in Section 6.2 that if r is slightly greater than $3/4$, the fixed point of f becomes unstable and bifurcates to a cycle of period 2. Now x returns to the same value after every second iteration, and the fixed points of $f(f(x))$ are the stable attractors of $f(x)$. In the following, we write $f^{(2)}(x) = f(f(x))$ and $f^{(n)}(x)$ for the nth iterate of $f(x)$. (Do not confuse $f^{(n)}(x)$ with the nth derivative of $f(x)$.) For example, the second iterate $f^{(2)}(x)$ is given by the fourth-order polynomial:

$$\begin{aligned} f^{(2)}(x) &= 4r[4rx(1-x)] - 4r[4rx(1-x)]^2 \\ &= 4r[4rx(1-x)][1 - 4rx(1-x)] \\ &= 16r^2x[-4rx^3 + 8rx^2 - (1+4r)x + 1]. \end{aligned} \tag{6.7}$$

What happens if we increase r still further? Eventually the magnitude of the slope of the fixed points of $f^{(2)}(x)$ exceeds unity, and the fixed points of $f^{(2)}(x)$ become unstable. Now the cycle of f is period 4, and the fixed points of the fourth iterate $f^{(4)}(x) = f^{(2)}(f^{(2)}(x)) = f(f(f(f(x))))$ are stable. These fixed points also eventually become unstable, and we are led to the phenomena of *period doubling* that we observed in Problem 6.2.

`GraphicalSolutionApp` implements the graphical analysis of the iterations of $f(x)$. The nth-order iterates are defined in `f(x,r,iterate)`, a *recursive* method. (The parameter

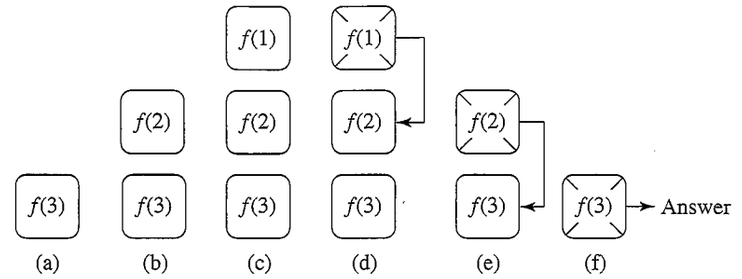

Figure 6.4 Example of the calculation of `f(0.4,0.8,3)` using the recursive function defined in `GraphicalSolutionApp`. The number in each box is the value of the variable `iterate`. The computer executes code from left to right, and each box represents a copy of the function in the computer's memory. The input values $x = 0.4$ and $r = 0.8$, which are the same in each copy, are not shown. The arrows indicate when a copy is finished and its value is returned to one of the other copies. Notice that the first copy of the function $f(3)$ is the last one to finish. The value of `f(x,r,3)` $= 0.7842$.

`iterate` is 1, 2, and 4 for the functions $f(x)$, $f^{(2)}(x)$, and $f^{(4)}(x)$, respectively.) Recursion is an idea that is simple once you understand it, but it can be difficult to grasp initially. Although the method calls itself, the rules for method calls remain the same. Imagine that a recursive method is called. The computer then starts to execute the code in the method, but comes to another call of the same method as itself. At this point the computer stops executing the code of the original method, and makes an exact copy of the method with possibly different input parameters, and starts executing the code in the copy. There are now two possibilities. One is that the computer comes to the end of the copy without another recursive call. In that case the computer deletes the copy of the method and continues executing the code in the original method. The other possibility is that a recursive call is made in the copy, and a third copy is made of the method, and the code in the third copy is now executed. This process continues until the code in all the copies is executed. Every recursive method must have a possibility of reaching the end of the method; otherwise, the program will eventually crash.

To understand `f(x,r,iterate)`, suppose we want to compute `f(0.4,0.8,3)`. First we write `f(0.4,0.8,3)` as in Figure 6.4a. Follow the statements within the method until another call to `f(0.4,0.8,iterate)` occurs. In this case the call is to `f(0.4,0.8,iterate-1)` which equals `f(0.4,0.8,2)`. Write `f(0.4,0.8,2)` above `f(0.4,0.8,3)` (see Figure 6.4b). When you come to the end of the definition of the method, write down the value of `f` that is actually returned, and remove the method from the stack by crossing it out (see Figure 6.4d). This returned value for `f` equals `y` if `iterate` > 1, or it is the output of the method for `iterate` $= 1$. Continue deleting copies of `f` as they are finished until there are no copies left on the paper. The final value of `f` is the value returned by the computer. Write a short program that defines `f(x,r,iterate)` and prints the value of `f(0.4,0.8,3)`. Is the answer the same as your hand calculation?

Listing 6.3 `GraphicalSolutionApp` displays the graphical solution of the logistic map trajectory.

```
package org.opensourcephysics.sip.ch06;
import org.opensourcephysics.controls.*;
import org.opensourcephysics.frames.PlotFrame;

public class GraphicalSolutionApp extends AbstractSimulation {
```