than “just” programming. The most important computer for any computing professional is still the one between the ears.

Hopefully this book has helped you on the road to becoming a computer programmer. Along the way, I have tried to pique your curiosity about the science of computing. If you have mastered the concepts in this text, you can already write interesting and useful programs. You should also have a firm foundation of the fundamental ideas of computer science and software engineering. Should you be interested in studying these fields in more depth, I can only say “go for it.” Perhaps one day you will also consider yourself a computer scientist; I would be delighted if my book played even a very small part in that process.

13.5 Chapter Summary

This chapter has introduced you to a number of important concepts in computer science that go beyond just programming. Here are the key ideas:

- One core subfield of computer science is analysis of algorithms. Computer scientists analyze the time efficiency of an algorithm by considering how many steps the algorithm requires as a function of the input size.

- Searching is the process of finding a particular item among a collection. Linear search scans the collection from start to end and requires time linearly proportional to the size of the collection. If the collection is sorted, it can be searched using the binary search algorithm. Binary search only requires time proportional to the log of the collection size.

- Binary search is an example of a divide-and-conquer approach to algorithm development. Divide-and-conquer often yields efficient solutions.

- A definition or function is recursive if it refers to itself. To be well-founded, a recursive definition must meet two properties:

  1. There must be one or more base cases that require no recursion.
  2. All chains of recursion must eventually reach a base case.

  A simple way to guarantee these conditions is for recursive calls to always be made on smaller versions of the problem. The base cases are then simple versions that can be solved directly.

- Sequences can be considered recursive structures containing a first item followed by a sequence. Recursive functions can be written following this approach.

- Recursion is more general than iteration. Choosing between recursion and looping involves the considerations of efficiency and elegance.
- Sorting is the process of placing a collection in order. A selection sort requires time proportional to the square of the size of the collection. Merge sort is a divide and conquer algorithm that can sort a collection in $n \log n$ time.
- Problems that are solvable in theory but not in practice are called intractable. The solution to the famous Tower of Hanoi can be expressed as a simple recursive algorithm, but the algorithm is intractable.
- Some problems are in principle unsolvable. The halting problem is one example of an unsolvable problem.
- You should consider becoming a computer scientist.

13.6 Exercises

Review Questions

True/False

1. Linear search requires a number of steps proportional to the size of the list being searched.
2. The Python operator `in` performs a binary search.
3. Binary search is an $n \log n$ algorithm.
4. The number of times n can be divided by 2 is $exp(n)$.
5. All proper recursive definitions must have exactly one non-recursive base case.
6. A sequence can be viewed as a recursive data collection.
7. A word of length n has $n!$ anagrams.
8. Loops are more general than recursion.
9. Merge sort is an example of an $n \log n$ algorithm.
10. Exponential algorithms are generally considered intractable.

Multiple Choice

1. Which algorithm requires time directly proportional to the size of the input?
   a) linear search b) binary search
   c) merge sort d) selection sort

2. Approximately how many iterations will binary search need to find a value in a list of 512 items?
   a) 512 b) 256 c) 9 d) 3

3. Recursions on sequences often use this as a base case:
   a) 0 b) 1 c) an empty sequence d) `None`

4. An infinite recursion will result in
   a) a program that "hangs"
   b) a broken computer
   c) a reboot
   d) a run-time exception

5. The recursive Fibonacci function is inefficient because
   a) it does many repeated computations
   b) recursion is inherently inefficient compared to iteration
   c) calculating Fibonacci numbers is intractable
   d) fibbing is morally wrong

6. Which is a quadratic time algorithm?
   a) linear search b) binary search
   c) Tower of Hanoi d) selection sort

7. The process of combining two sorted sequences is called
   a) sorting b) shuffling c) dovetailing d) merging

8. Recursion is related to the mathematical technique called
   a) looping b) sequencing c) induction d) contradiction

9. How many steps would be needed to solve the Tower of Hanoi for a tower of size 5?
   a) 5 b) 10 c) 25 d) 31

10. Which of the following is *not* true of the halting problem?
    a) It was studied by Alan Turing.
    b) It is harder than intractable.
    c) Someday a clever algorithm may be found to solve it.
    d) It involves a program that analyzes other programs.

Discussion

1. Place these algorithm classes in order from fastest to slowest: $n \log n$, n, n^2, $\log n$, 2^n.

2. In your own words, explain the two rules that a proper recursive definition or function must follow.

3. What is the exact result of `anagram("foo")`?

4. Trace `recPower(3,6)` and figure out exactly how many multiplications it performs.

5. Why are divide-and-conquer algorithms often very efficient?

Programming Exercises

1. Modify the recursive Fibonacci program given in this chapter so that it prints tracing information. Specifically, have the function print a message when it is called and when it returns. For example, the output should contain lines like these:

   ```
   Computing fib(4)
   ...
   Leaving fib(4) returning 3
   ```

   Use your modified version of `fib` to compute `fib(10)` and count how many times `fib(3)` is computed in the process.

2. This exercise is another variation on "instrumenting" the recursive Fibonacci program to better understand its behavior. Write a program that counts how many times the `fib` function is called to compute `fib(n)` where `n` is a user input.

   Hint: To solve this problem, you need an accumulator variable whose value "persists" between calls to `fib`. You can do this by making the count an instance variable of an object. Create a `FibCounter` class with the following methods:

`__init__(self)` Creates a new `FibCounter`, setting its count instance variable to 0.

`getCount(self)` Returns the value of count.

`fib(self,n)` Recursive function to compute the nth Fibonacci number. It increments the count each time it is called.

`resetCount(self)` Sets the count back to 0.

3. A *palindrome* is a sentence that contains the same sequence of letters reading it either forwards or backwards. A classic example is "Able was I, ere I saw Elba." Write a recursive function that detects whether a string is a palindrome. The basic idea is to check that the first and last letters of the string are the same letter; if they are, then the entire string is a palindrome if everything between those letters is a palindrome.

   There are a couple of special cases to check for. If either the first or last character of the string is not a letter, you can check to see if the rest of the string is a palindrome with that character removed. Also, when you compare letters, make sure that you do it in a case-insensitive way.

   Use your function in a program that prompts a user for a phrase and then tells whether or not it is a palindrome. Here's another classic for testing: "A man, a plan, a canal, Panama!"

4. Write and test a recursive function `max` to find the largest number in a list. The max is the larger of the first item and the max of all the other items.

5. Computer scientists and mathematicians often use numbering systems other than base 10. Write a program that allows a user to enter a number and a base and then prints out the digits of the number in the new base. Use a recursive function `baseConversion(num,base)` to print the digits.

   Hint: Consider base 10. To get the rightmost digit of a base 10 number, simply look at the remainder after dividing by 10. For example, `153 % 10` is 3. To get the remaining digits, you repeat the process on 15, which is just `153 // 10`. This same process works for any base. The only problem is that we get the digits in reverse order (right to left).

   The base case for the recursion occurs when `num` is less than `base` and the output is simply `num`. In the general case, the function (recursively) prints the digits of `num // base` and then prints `num % base`. You should put a space between successive outputs, since bases greater than 10 will print out with multi-character "digits." For example, `baseConversion(1234, 16)` should print `4 13 2`.

6. Write a recursive function to print out the digits of a number in English. For example, if the number is 153, the output should be "One Five Three." See the hint from the previous problem for help on how this might be done.

7. In mathematics, C_k^n denotes the number of different ways that k things can be selected from among n different choices. For example, if you are choosing among six desserts and are allowed to take two, the number of different combinations you could choose is C_2^6. Here's one formula to compute this value:

$$C_k^n = \frac{n!}{k!(n-k)!}$$

This value also gives rise to an interesting recursion:

$$C_k^n = C_{k-1}^{n-1} + C_k^{n-1}$$

Write both an iterative and a recursive function to compute combinations and compare the efficiency of your two solutions. *Hints*: When $k = 1$, $C_k^n = n$ and when $n < k$, $C_k^n = 0$.

8. Some interesting geometric curves can be described recursively. One famous example is the Koch curve. It is a curve that can be infinitely long in a finite amount of space. It can also be used to generate pretty pictures.

The Koch curve is described in terms of "levels" or "degrees." The Koch curve of degree 0 is just a straight line segment. A first degree curve is formed by placing a "bump" in the middle of the line segment (see Figure 13.6). The original segment has been divided into four, each of which is $\frac{1}{3}$ the length of the original. The bump rises at 60 degrees, so it forms two sides of an equilateral triangle. To get a second-degree curve, you put a bump in each of the line segments of the first-degree curve. Successive curves are constructed by placing bumps on each segment of the previous curve.

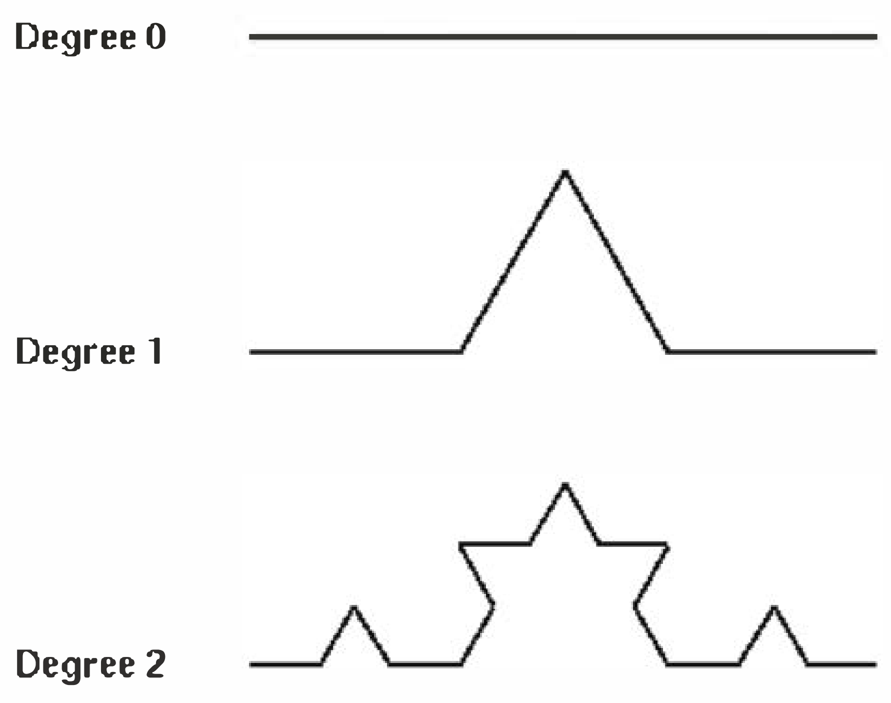

Figure 13.6: Koch curves of degree 0 to 2

You can draw interesting pictures by "Kochizing" the sides of a polygon. Figure 13.7 shows the result of applying a fourth-degree curve to the sides of an equilateral triangle. This is often called a "Koch snowflake." You are to write a program to draw a snowflake.

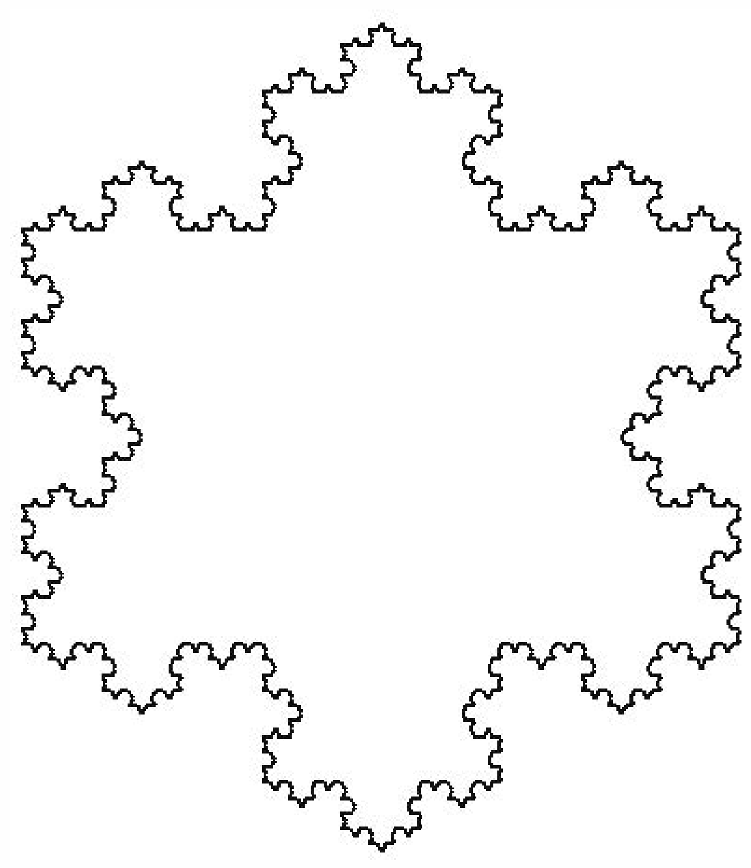

Figure 13.7: Koch snowflake

Think of drawing a Koch curve as if you were giving instructions to a turtle. The turtle always knows where it currently sits and what direction it is facing. To draw a Koch curve of a given length and degree, you might use an algorithm like this:

```
Algorithm Koch(Turtle, length, degree):
    if degree == 0:
        Tell the turtle to draw for length steps
    else:
        length1 = length/3
        degree1 = degree-1
        Koch(Turtle, length1, degree1)
        Tell the turtle to turn left 60 degrees
        Koch(Turtle, length1, degree1)
        Tell the turtle to turn right 120 degrees
        Koch(Turtle, length1, degree1)
        Tell the turtle to turn left 60 degrees
        Koch(Turtle, length1, degree1)
```

Implement this algorithm with a `Turtle` class that contains instance variables `location` (a `Point`) and `Direction` (a float) and methods such as `moveTo(somePoint)`, `draw(length)`, and `turn(degrees)`. If you maintain direction as an angle in radians, the point you are going to can easily be computed from your current location. Just use `dx = length * cos(direction)` and `dy = length * sin(direction)`.

9. Another interesting recursive curve (see previous problem) is the C-curve. It is formed similarly to the Koch curve except whereas the Koch curve breaks a segment into four pieces of $length/3$, the C-curve replaces each segment with just two segments of $length/\sqrt{2}$ that form a 90-degree elbow. Figure 13.8 shows a degree 12 C-curve.

   Using an approach similar to the previous exercise, write a program that draws a C-curve. *Hint*: Your turtle will do the following:

```
turn left 45 degrees
draw a c-curve of size length/sqrt(2)
turn right 90 degrees
draw a c-curve of size length/sqrt(2)
turn left 45 degrees
```

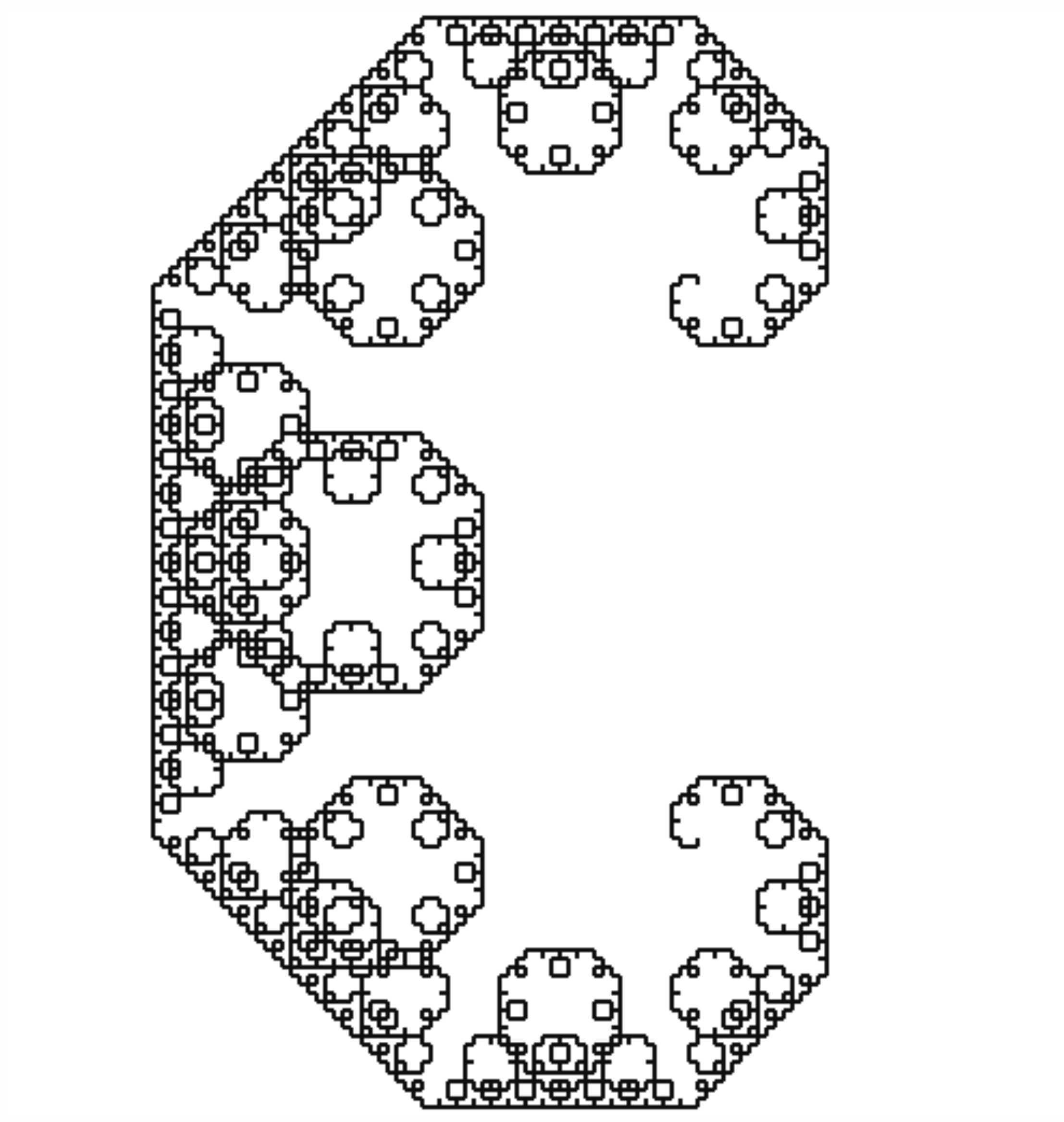

Figure 13.8: C-curve of degree 12

10. Automated spell-checkers are used to analyze documents and locate words that might be misspelled. These programs work by comparing each word in the document to a large dictionary (in the non-Python sense) of words. Any word not found in the dictionary, it is flagged as potentially incorrect.

    Write a program to perform spell-checking on a text file. To do this, you will need to get a large file of English words in alphabetical order. If you have a Unix or Linux system available, you might poke around for a file called `words`, usually located in `/usr/dict` or `/usr/share/dict`. Otherwise, a quick search on the Internet should turn up something usable.

    Your program should prompt for a file to analyze and then try to look up every word in the file using binary search. If a word is not found in the dictionary, print it on the screen as potentially incorrect.

11. Write a program that solves word jumble problems. You will need a large dictionary of English words (see previous problem). The user types in a scrambled word, and your program generates all anagrams of the word and then checks which (if any) are in the dictionary. The anagrams appearing in the dictionary are printed as solutions to the puzzle.