IN THIS CHAPTER

- Defining what is meant by *algorithm*
- Relying on computers to use algorithms to provide solutions
- Determining how issues differ from solutions
- Performing data manipulation so that you can find a solution

Chapter 1
Introducing Algorithms

If you're in the majority of people, you're likely confused as you open this book and begin your adventure with algorithms, because most texts never tell you what an algorithm is, much less why you'd want to use one. Hearing about algorithms is like being in school again with the teacher droning on; you're falling asleep from lack of interest because algorithms don't seem particularly useful to understand at the moment.

The first section of this chapter is dedicated to helping you understand precisely what the term *algorithm* means and why you benefit from knowing how to use algorithms. Far from being arcane, algorithms are actually used all over the place, and you have probably used or been helped by them for years without really knowing it. So, they're stealth knowledge! In truth, algorithms are becoming the spine that supports and regulates what is important in an increasingly complex and technological society like ours.

The second section of this chapter discusses how you use computers to create solutions to problems using algorithms, how to distinguish between issues and solutions, and what you need to do to manipulate data to discover a solution. The goal is to help you differentiate between algorithms and other tasks that people confuse with algorithms. In short, you discover why you really want to know about algorithms, as well as how to apply them to data.

The third section of the chapter discusses algorithms in a real-world manner, that is, by viewing the terminologies used to understand algorithms and to present algorithms in a way that shows that the real world is often less than perfect. Understanding how to describe an algorithm in a realistic manner also helps to temper expectations to reflect the realities of what an algorithm can actually do.

The final section of the chapter discusses data. The algorithms you work with in this book require data input in a specific form, which sometimes means changing the data to match the algorithm's requirements. Data manipulation doesn't change the content of the data. Instead, it changes the presentation and form of the data so that an algorithm can help you see new patterns that weren't apparent before (but were actually present in the data all along).

Describing Algorithms

Even though people have solved algorithms manually for thousands of years, doing so can consume huge amounts of time and require many numeric computations, depending on the complexity of the problem you want to solve. Algorithms are all about finding solutions, and the speedier and easier, the better. A huge gap exists between mathematical algorithms historically created by geniuses of their time, such as Euclid (`https://www.britannica.com/biography/Euclid-Greek-mathematician`), Sir Isaac Newton (`https://www.britannica.com/biography/Isaac-Newton`), or Carl Friedrich Gauss (`https://www.britannica.com/biography/Carl-Friedrich-Gauss`), and modern algorithms created in universities as well as private research and development laboratories. The main reason for this gap is the use of computers. Using computers to solve problems by employing the appropriate algorithm speeds up the task significantly. You may notice that more problem solutions appear quickly today, in part, because computer power is both cheap and constantly increasing.

When working with algorithms, you consider the inputs, desired outputs, and the process (a sequence of actions) used to obtain a desired output from a given input. However, you can get the terminology wrong and view algorithms in the wrong way because you haven't really considered how they work in a real-world setting.

Sources of information about algorithms often present them in a way that proves confusing because they're too sophisticated or even downright incorrect. Although you may find other definitions, this book uses the following definitions for terms that people often confuse with algorithms (but aren't):

- **Equation:** Numbers and symbols that, when taken as a whole, equate to a specific value. An equation always contains an equals sign so that you know that the numbers and symbols represent the specific value on the other side of the equals sign. Equations generally contain variable information presented as a symbol, but they're not required to use variables.
- **Formula:** A combination of numbers and symbols used to express information or ideas. Formulas normally present mathematical or logical concepts, such as defining the Greatest Common Divisor (GCD) of two integers (the video at `https://www.khanacademy.org/math/cc-sixth-grade-math/cc-6th-factors-and-multiples/cc-6th-gcf/v/greatest-common-divisor` tells how this works). Generally, they show the relationship between two or more variables.


REMEMBER

- **Algorithm:** A sequence of steps used to solve a problem. The sequence presents a unique method of addressing an issue by providing a particular solution. An algorithm need not represent mathematical or logical concepts, even though the presentations in this book often do fall into those categories because people most commonly use algorithms in this manner. In order for a process to represent an algorithm, it must be:
  - **Finite:** The algorithm must eventually solve the problem. This book discusses problems with a known solution so that you can evaluate whether an algorithm solves the problem correctly.
  - **Well-defined:** The series of steps must be precise and present steps that are understandable. Especially because computers are involved in algorithm use, the computer must be able to understand the steps to create a usable algorithm.
  - **Effective:** An algorithm must solve all cases of the problem for which someone defined it. An algorithm should always solve the problem it has to solve. Even though you should anticipate some failures, the incidence of failure is rare and occurs only in situations that are acceptable for the intended algorithm use.

With these definitions in mind, the following sections help to clarify the precise nature of algorithms. The goal isn't to provide a precise definition for algorithms, but rather to help you understand how algorithms fit into the grand scheme of things so that you can develop your own understanding of what algorithms are and why they're so important.

The right way to make toast: Defining algorithm uses

An algorithm always presents a series of steps and doesn't necessarily perform these steps to solve a math formula. The scope of algorithms is incredibly large. You can find algorithms that solve problems in science, medicine, finance, industrial production and supply, and communication. Algorithms provide support for all parts of a person's daily life. Anytime a sequence of actions achieving something in our life is finite, well-defined, and effective, you can view it as an algorithm. For example, you can turn even something as trivial and simple as making toast into an algorithm. In fact, the making toast procedure often appears in computer science classes, as discussed at `http://brianaspinall.com/now-thats-how-you-make-toast-using-computer-algorithms/`.

Unfortunately, the algorithm on the site is flawed. The instructor never removes the bread from the wrapper and never plugs the toaster in, so the result is damaged plain bread still in its wrapper stuffed into a nonfunctional toaster (see the discussion at `http://blog.johnmuellerbooks.com/2013/03/04/procedures-in-technical-writing/` for details). Even so, the idea is the correct one, yet it requires some slight, but essential, adjustments to make the algorithm finite and effective.

One of the most common uses of algorithms is as a means of solving formulas. For example, when working with the GCD of two integer values, you can perform the task manually by listing each of the factors for the two integers and then selecting the greatest factor that is common to both. For example, GCD (20, 25) is 5 because 5 is the largest number that divides evenly into both 20 and 25. However, processing every GCD manually is time consuming and error prone, so the Greek mathematician Euclid created a better algorithm to perform the task. You can see the Euclidean method demonstrated at `https://www.khanacademy.org/computing/computer-science/cryptography/modarithmetic/a/the-euclidean-algorithm`.

However, a single formula, which is a presentation of symbols and numbers used to express information or ideas, can have multiple solutions, each of which is an algorithm. In the case of GCD, another common algorithm is one created by Derrick Henry Lehmer (`https://www.imsc.res.in/~kapil/crypto/notes/node11.html`). Because you can solve any formula multiple ways, people spend a great deal of time comparing algorithms to determine which one works best in a given situation. (See a comparison of Euclid to Lehmer at `http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.31.693&rep=rep1&type=pdf`.)

Because our society and its accompanying technology are changing quickly, we need algorithms that can keep the pace. Scientific achievements such as

sequencing the human genome were possible in our age because scientists found algorithms that run fast enough to complete the task. Measuring which algorithm is better in a given situation, or in an average usage situation, is really serious stuff and is a topic of discussion among computer scientists.

When it comes to computer science, the same algorithm can have multiple presentations; why do it one way when you can invent multiple methods just for fun? For example, you can *present* the Euclidean algorithm in both recursive and iterative forms, as explained at `http://cs.stackexchange.com/questions/1447/what-is-most-efficient-for-gcd`. In short, algorithms present a method of solving formulas, but it would be a mistake to say that just one acceptable algorithm exists for any given formula or that only one acceptable presentation of an algorithm exists. Using algorithms to solve problems of various sorts has a long history — it isn't something that has just happened.

Even if you limit your gaze to computer science, data science, artificial intelligence, and other technical areas, you find many kinds of algorithms — too many for a single book. For example, *The Art of Computer Programming*, by Donald E. Knuth (Addison-Wesley), spans 3,168 pages in four volumes (see `http://www.amazon.com/exec/obidos/ASIN/0321751043/datacservip0f-20/`) and still doesn't manage to cover the topic (the author intended to write more volumes). However, here are some interesting uses for you to consider:

- **Searching:** Locating information or verifying that the information you see is the information you want is an essential task. Without this ability, you couldn't perform many tasks online, such as finding the website on the Internet selling the perfect coffee pot for your office. These algorithms change constantly, as shown by Google's recent change in its algorithm (`https://www.youaretech.com/blog/2021/1/26/webpage-experience-a-major-google-algorithm-update-in-2021nbsp`).
- **Sorting:** Determining which order to use to present information is important because most people today suffer from information overload, and putting information in order is one way to reduce the onrush of data. Imagine going to Amazon, finding that more than a thousand coffee pots are for sale there, and yet not being able to sort them in order of price or the most positive review. Moreover, many complex algorithms require data in the proper order to work dependably, so ordering is an important requisite for solving more problems.
- **Transforming:** Converting one sort of data to another sort of data is critical to understanding and using the data effectively. For example, you might understand imperial weights just fine, but all your sources use the metric system. Converting between the two systems helps you understand the data.

- **Scheduling:** Making the use of resources fair to all concerned is another way in which algorithms make their presence known in a big way. For example, timing lights at intersections are no longer simple devices that count down the seconds between light changes. Modern devices consider all sorts of issues, such as the time of day, weather conditions, and flow of traffic.
- **Graph analysis:** Deciding on the shortest path between two points finds all sorts of uses. For example, in a routing problem, your GPS couldn't function without this particular algorithm because it could never direct you along city streets using the shortest route from point A to point B. And even then, your GPS might direct you to drive into a lake (`https://theweek.com/articles/464674/8-drivers-who-blindly-followed-gps-into-disaster`).
- **Cryptography:** Keeping data safe is an ongoing battle with hackers constantly attacking data sources. Algorithms make it possible to analyze data, put it into some other form, and then return it to its original form later.
- **Pseudorandom number generation:** Imagine playing games that never varied. You start at the same place; perform the same steps, in the same manner, every time you play. Without the capability to generate seemingly random numbers, many computer tasks become impossible.

REMEMBER

This list presents an incredibly short overview. People use algorithms for many different tasks and in many different ways, and constantly create new algorithms to solve both existing problems and new problems. The most important issue to consider when working with algorithms is that given a particular input, you should expect a specific output. Secondary issues include how many resources the algorithm requires to perform its task and how long it takes to complete the task. Depending on the kind of issue and the sort of algorithm used, you may also need to consider issues of accuracy and consistency.

Finding algorithms everywhere

The previous section mentions the toast algorithm for a specific reason. For some reason, making toast is probably the most popular algorithm ever created. Many grade-school children write their equivalent of the toast algorithm long before they can even solve the most basic math. It's not hard to imagine how many variations of the toast algorithm exist and what the precise output is of each of them. The results likely vary by individual and the level of creativity employed. There are also websites dedicated to telling children about algorithms, such as the one at `https://www.idtech.com/blog/algorithms-for-kids`. In short, algorithms appear in great variety and often in unexpected places.

Every task you perform on a computer involves algorithms. Some algorithms appear as part of the computer hardware. The very act of booting a computer involves the use of an algorithm. You also find algorithms in operating systems, applications, and every other piece of software. Even users rely on algorithms. Scripts help direct users to perform tasks in a specific way, but those same steps could appear as written instructions or as part of an organizational policy statement.

Daily routines often devolve into algorithms. Think about how you spend your day. If you're like most people, you perform essentially the same tasks every day in the same order, making your day an algorithm that solves the problem of how to live successfully while expending the least amount of energy possible. After all, that's what a routine does; it makes us efficient.

Throughout this book, you see the same three elements for every algorithm:

1. Describe the problem.
2. Create a series of steps to solve the problem (well defined).
3. Perform the steps to obtain a desired result (finite and effective).

Using Computers to Solve Problems

The term *computer* sounds quite technical and possibly a bit overwhelming to some people, but people today are neck deep (possibly even deeper) in computers. You wear at least one computer, your smartphone, most of the time. If you have any sort of special device, such as a pacemaker, it also includes a computer. A car can contain as many as 150 computers in the form of embedded microprocessors that regulate fuel consumption, engine combustion, transmission, steering, and stability (see `https://spectrum.ieee.org/software-eating-car` for details), provide Advanced Driver-Assist Systems (ADAS), and more lines of code than a jet fighter. A computer exists to solve problems quickly and with less effort than solving them manually. Consequently, it shouldn't surprise you that this book uses still more computers to help you understand algorithms better.

Computers vary in a number of ways. The computer in a watch is quite small; the one on a desktop quite large. Supercomputers are immense and contain many smaller computers all tasked to work together to solve complex issues, such as predicting tomorrow's weather. The most complex algorithms rely on special computer functionality to obtain solutions to the issues people design them to solve. Yes, you could use lesser resources to perform the task, but the trade-off is waiting a lot longer for an answer, or getting an answer that lacks sufficient

accuracy to provide a useful solution. In some cases, you wait so long that the answer is no longer important. With the need for both speed and accuracy in mind, the following sections discuss some special computer features that can affect algorithms.

Getting the most out of modern CPUs and GPUs

General-purpose processors, CPUs, started out as a means to solve problems using algorithms. However, their general-purpose nature also means that a CPU can perform a great many other tasks, such as moving data around or interacting with external devices. A general-purpose processor does many things well, which means that it can perform the steps required to complete an algorithm, but not necessarily fast. Owners of early general-purpose processors could add math coprocessors (special math-specific chips) to their systems to gain a speed advantage (see `https://www.computerhope.com/jargon/m/mathcopr.htm` for details). Today, general-purpose processors have the math coprocessor embedded into them, so when you get an Intel i9 processor, you actually get multiple processors in a single package.

A *GPU* is a special-purpose processor with capabilities that lend themselves to faster algorithm execution. For most people, GPUs are supposed to take data, manipulate it in a special way, and then display a pretty picture onscreen. However, any computer hardware can serve more than one purpose. It turns out that GPUs are particularly adept at performing data transformations, which is a key task for solving algorithms in many cases. It shouldn't surprise you to discover that people who create algorithms spend a lot of time thinking outside the box, which means that they often see methods of solving issues in nontraditional approaches.

The point is that CPUs and GPUs form the most commonly used chips for performing algorithm-related tasks. The first performs general-purpose tasks quite well, and the second specializes in providing support for math-intensive tasks, especially those that involve data transformations. Using multiple cores makes parallel processing (performing more than one algorithmic step at a time) possible. Adding multiple chips increases the number of cores available. Having more cores adds speed, but a number of factors keeps the speed gain to a minimum. Using two i9 chips won't produce double the speed of just one i9 chip.

Working with special-purpose chips

A math coprocessor and a GPU are two examples of common special-purpose chips in that you don't see them used to perform tasks such as booting the system. However, algorithms often require the use of uncommon special-purpose chips to solve problems. This isn't a hardware book, but it pays to spend time showing you all sorts of interesting chips, such as the artificial neurons that UCSD is working on (https://www.marktechpost.com/2021/06/03/ucsd-researchers-develop-an-artificial-neuron-device-that-could-reduce-energy-use-and-size-of-neural-network-hardware/). Imagine performing algorithmic processing using devices that simulate the human brain. It would create an interesting environment for performing tasks that might not otherwise be possible today.

Neural networks, a technology that is used to simulate human thought and make deep learning techniques possible for machine learning scenarios, are now benefitting from the use of specialized chips (the article at https://analyticsindiamag.com/top-10-gpus-for-deep-learning-in-2021/ outlines ten of them). These kinds of chips not only perform algorithmic processing extremely fast but also learn as they perform the tasks, making them faster still with each iteration. Learning computers will eventually power robots that can move (after a fashion) on their own, akin to the robots seen in the movie *I Robot.* Some robots even sport facial expressions now (https://www.sciencedaily.com/releases/2021/05/210527145244.htm).

REMEMBER

No matter how they work, specialized processors will eventually power all sorts of algorithms that will have real-world consequences. You can already find many of these real-world applications in a relatively simple form. For example, imagine the tasks that a pizza-making robot would have to solve — the variables it would have to consider on a real-time basis. This sort of robot already exists (this is just one example of the many industrial robots used to produce material goods by employing algorithms), and you can bet that it relies on algorithms to describe what to do, as well as on special chips to ensure that the tasks are done quickly (https://www.therobotreport.com/picnic-pizza-making-robot-is-now-available/).

TECHNICAL STUFF

Eventually, it might even be possible to use the human mind as a processor and output the information through a special interface. At one point, people experimented with putting processors directly into the brain, but the latest innovation relies on the use of veins to make the connection (https://www.independent.co.uk/life-style/gadgets-and-tech/brain-computer-interface-veinals-stent-neuralink-b1556167.html). Imagine a system in which humans can solve problems using algorithms at the speed of computers, but with the creative "what if" potential of humans.

Networks: Sharing is more than caring

Unless you have unlimited funds, using some algorithms effectively may not be possible, even with specialized chips. In that case, you can network computers together. Using special software, one computer, a host, can use the processors of all client computers running an *agent* (a kind of in-memory background application that makes the processor available). Using this approach, you can solve incredibly complex problems by offloading pieces of the problem to a number of client computers. As each computer in the network solves its part of the problem, it sends the results back to the host, which puts the pieces together to create a consolidated answer, a technique called *cluster computing.*

Lest you think this is the stuff of science fiction, people are using cluster computing techniques (`https://www.geeksforgeeks.org/an-overview-of-cluster-computing/`) in all sorts of interesting ways. For example, the article at `https://turingpi.com/12-amazing-raspberry-pi-cluster-use-cases/` details how you can build your own supercomputer (among other tasks) by combining multiple Raspberry Pi (`https://www.raspberrypi.org/`) boards into a single cluster.

Distributed computing, another version of cluster computing (but with a looser organization) is also popular. In fact, you can find a list of distributed computing projects at `https://en.wikipedia.org/wiki/List_of_distributed_computing_projects`. The list of projects includes some major endeavors, such as Search for Extraterrestrial Intelligence (SETI). You can also donate your computer's extra processing power to work on a cure for cancer. The list of potential projects is amazing. Most of these projects are hosted by Berkeley Open Infrastructure for Network Computing (`https://boinc.berkeley.edu/`), but you can find other sponsors.

Leveraging available data

Part of solving problem using an algorithm has nothing to do with processing power, creative thinking outside the box, or anything of a physical nature. To create a solution to most problems, you also need data on which to base a conclusion. For example, in the toast-making algorithm, you need to know about the availability of bread, a toaster, electricity to power the toaster, and so on before you can solve the problem of actually making toast. The data becomes important because you can't finish the algorithm when missing even one element of the required solution. Of course, you may need additional input data as well. For example, the person wanting the toast may not like rye. If this is the case and all you have is rye bread to use, the presence of bread still won't result in a successful result.

Data comes from all sorts of sources and in all kinds of forms. You can stream data from a source such as a real-time monitor, access a public data source, rely on

private data in a database, scrape the data from websites, or get it in myriad other ways too numerous to mention here. The data may be static (unchanging) or dynamic (constantly changing). You may find that the data is complete or missing elements. The data may not appear in the right form (such as when you get imperial units and require metric units when solving a weight problem). The data may appear in a tabular format when you need it in some other form. It may reside in an unstructured way (for instance, in a NoSQL database or just in a bunch of different data files) when you need the formatting of a relational database. In short, you need to know all sorts of things about the data used with your algorithm in order to solve problems with it.


REMEMBER

Because data comes in so many forms and you need to work with it in so many ways, this book pays a lot of attention to data. Starting in Chapter 6, you discover just how data structure comes into play. Moving on to Chapter 7, you begin looking at how to search through data to find what you need. Chapters 12 through 14 help you work with big data. However, you can find some sort of data-specific information in just about every chapter of the book because without data, an algorithm can't solve any problems.

Distinguishing between Issues and Solutions

This book discusses two parts of the algorithmic view of the real world. On the one hand, you have *issues,* which are problems that you need to solve. An issue can describe the desired output of an algorithm, or it can describe a hurdle you must overcome to obtain the desired output. *Solutions* are the methods, or steps, used to address the issues. A solution can relate to just one step or many steps within the algorithm. In fact, the output of an algorithm, the response to the last step, is a solution. The following sections help you understand some of the important aspects of issues and solutions.

Being correct and efficient

Using algorithms is all about getting an acceptable answer. The reason you look for an acceptable answer is that some algorithms generate more than one answer in response to fuzzy input data. Life often makes precise answers impossible to get. Of course, getting a precise answer is always the goal, but often you end up with an acceptable answer instead.

Getting the most precise answer possible may take too much time. When you seek a precise answer that takes too long to obtain, the information becomes useless

and you've wasted your time. Choosing between two algorithms that address the same issue may come down to a choice between speed and precision. A fast algorithm may not generate a precise answer, but the answer may still work well enough to provide useful output.

Wrong answers can be a problem. Creating a lot of wrong answers fast is just as bad as creating a lot of precisely correct answers slowly. Part of the focus of this book is helping you find the middle ground between too fast and too slow, and between inaccurate and too accurate. Even though your math teacher stressed the need for providing the correct answer in the way expressed by the book you used at the time, real-world math often involves weighing choices and making middle-ground decisions that affect you in ways you might not think possible.

Discovering there is no free lunch

You may have heard the common myth that you can have everything in the way of computer output without putting much effort into deriving the solution. Unfortunately, no absolute solution exists to any problem, and better answers are often quite costly. When working with algorithms, you quickly discover the need to provide additional resources when you require precise answers quickly. The size and complexity of the data sources you use greatly affect the solution resolution as well. As size and complexity increase, you find that the need to add resources increases as well.

Adapting the strategy to the problem

Part 5 of this book looks at strategies you can use to decrease the cost of working with algorithms. The best mathematicians use tricks to get more output from less computing. For example, you can create an ultimate algorithm to solve an issue, or you can use a host of simpler algorithms to solve the same issue, but using multiple processors. The host of simple algorithms will usually work faster and better than the single, complex algorithm, even though this approach seems counterintuitive.

Describing algorithms in a lingua franca

Algorithms do provide a basis for communication between people, even when those individuals have different perspectives and speak different languages. For example, Bayes' Theorem (the probability of an event occurring given certain premises; see `https://betterexplained.com/articles/an-intuitive-and-short-explanation-of-bayes-theorem/` for a quick explanation of this amazing theorem)

```
P(B|E) = P(E|B)*P(B)/P(E)
```

appears the same whether you speak English, Spanish, Chinese, German, French, or any other language. Regardless what language you speak, the algorithm looks the same and acts the same given the same data. Algorithms help cross all sorts of divides that serve to separate humans from each other by expressing ideas in a form that anyone can prove. As you go through this book, you discover the beauty and magic that algorithms can provide in communicating even subtle thoughts to others.


TIP

Apart from universal mathematical notations, algorithms take advantage of programming languages as a means for explaining and communicating the formulas they solve. You can find all the sorts of algorithms in C, C++, Java, Fortran, Python (as in this book), and other languages. *Pseudocode* is a way to describe computer operations by using common English words. Some writers rely on pseudocode to overcome the fact that an algorithm may be proposed in a programming language that you don't know. In addition, pseudocode can be more concise than a programming language because you can use intuitive ideas that the programming language may not capture well.

Facing problems that are like brick walls, only harder

An important consideration when working with algorithms is that you can use them to solve issues of any complexity. The algorithm doesn't think, have emotion, or care how you use it (or even abuse it). You can use algorithms in any way required to solve an issue. For example, the same group of algorithms used to perform facial recognition to act as an alternative to computer passwords (for security purposes) can find terrorists lurking in an airport or recognize a lost child wandering the streets. The same algorithm has different uses; how to use it depends on the interests of the user. Part of the reason you want to read this book carefully is to help you solve those hard problems that may require only a simple algorithm to address.

Structuring Data to Obtain a Solution

Humans think about data in nonspecific ways and apply various rules to the same data to understand it in ways that computers never can. A computer's view of data is structured, simple, uncompromising, and most definitely not creative. When humans prepare data for a computer to use, the data often interacts with the algorithms in unexpected ways and produces undesirable output. The problem is one in which the human fails to appreciate the limited view of data that a computer has. The following sections describe two aspects of data that you see illustrated in many of the chapters to follow.

Understanding a computer's point of view

A computer has a simple view of data, but it's also a view that humans typically don't understand. For one thing, everything is a number to a computer because computers aren't designed to work with any other kind of data. Humans see characters on the computer display and assume that the computer interacts with the data in that manner, but the computer doesn't understand the data or its implications. The letter *A* is simply the number 65 to the computer. In fact, it's not truly even the number 65. The computer sees a series of electrical impulses that equate to a binary value of 0100 0001.

Computers also don't understand the whole concept of uppercase and lowercase. To a human, the lowercase *a* is simply another form of the uppercase *A*, but to a computer they're two different values. A lowercase *a* appears as the number 97 (a binary value of 0110 0001).

If these simple sorts of single letter comparisons could cause such problems between humans and computers, it isn't hard to imagine what happens when humans start assuming too much about other kinds of data. For example, a computer can't hear or appreciate music. Yet, music comes out of the computer speakers. The same holds true for graphics. A computer sees a series of 0s and 1s, not a graphic containing a pretty scene of the countryside.

REMEMBER

It's important to consider data from the computer's perspective when using algorithms. The computer sees only 0s and 1s, nothing else. Consequently, when you start working through the needs of the algorithm, you must view the data in that manner. You may actually find it beneficial to know that the computer's view of data makes some solutions easier to find, not harder. You discover more about this oddity in viewing data as the book progresses.

Arranging data makes the difference

Computers also have a strict idea about the form and structure of data. When you begin working with algorithms, you find that a large part of the job involves making the data appear in a form that the computer can use when using the algorithm to find a solution to an issue. Although a human can mentally see patterns in data that isn't arranged precisely right, computers really do need the precision to find the same pattern. The benefit of this precision is that computers can often make new patterns visible. In fact, that's one of the main reasons to use algorithms with computers — to help locate new patterns and then use those patterns to perform other tasks. For example, a computer may recognize a customer's spending pattern so that you can use the information to generate more sales automatically.