The Science of Computing
A Problem Solving Approach

John S. Conery

January 5, 2009
# Contents

1 Introduction ................................................. 1  
1.1 Computation ............................................. 1  
1.2 Algorithms ............................................... 6  
1.3 The Stored Program Computer ......................... 8  
1.4 The Science of Computing .............................. 10  
1.5 A Laboratory for Computational Experiments ........ 13  

2 The Ruby Workbench ...................................... 17  
2.1 Interactive Ruby ........................................ 17  
2.2 Numbers .................................................. 21  
2.3 Variables .................................................. 23  
2.4 Methods ................................................... 27  
2.5 The celsius Method ...................................... 30  
2.6 Chapter Review .......................................... 32  

3 The Sieve of Eratosthenes ............................... 35  
3.1 The mod Function ........................................ 37  
3.2 Containers ................................................ 37  
3.3 Iterators ................................................... 41  
3.4 Exploring the Algorithm ............................... 44  
3.5 The sieve Method ....................................... 47  
3.6 A Better Sieve .......................................... 49  

4 * Sorting .................................................... 53  

5 * The von Neumann Machine ........................... 55  

6 * Searching, Part I ....................................... 57  

7 ? Searching, Part II ...................................... 59  

8 When Words Collide ...................................... 61  
8.1 Word Lists ............................................... 61  
8.2 Table Structure ......................................... 64  
8.3 Radix-26 .................................................. 67  
8.4 The mod Function ....................................... 70  
8.5 Dealing with Collisions ............................... 72  
8.6 Experiments .............................................. 77  
8.7 Associative Maps ....................................... 77  
8.8 Google ..................................................... 77
Introduction

1.1 Computation

If you were born after 1985 you have always lived in a world with personal computers. Computers have become common household appliances throughout the industrialized world. In a recent survey, 85% of American adults reported they use information and communication technology to some extent, ranging from those who surround themselves with technology and are constantly connected to the Internet to those who have access to technology but seldom use it. Another survey found that over 50% of the households in European countries own at least one computer. We use computers to write mail, share photographs, do homework, manage personal finances, play games, play music, write books. Information and computer technology is ubiquitous, and (when it is working) taken for granted.

For older generations computers were much more mysterious. Computers were giant machines found only in the largest government bureaus, corporate headquarters, or university research labs. These computers were mainly “number crunchers” that worked on large scale numeric calculations, such as computing the trajectory of a rocket or creating actuarial tables used by insurance agencies. Computers were commonly described as “electronic brains” since the calculations were the sorts of things that were previously done by people with advanced mathematical skills. Science fiction took this machine-as-brain viewpoint a step further, portraying computers as super-intelligent beings that could converse with humans, either as helpful assistants (the shipboard computer in Star Trek) or malevolent adversaries (Colossus in The Forbin Project or HAL-9000 in 2001: A Space Odyssey).

Whether you think of computers as useful appliances or electronic brains you probably have a sense that there are limits to what computers can do. Often a computer’s limitations are simply technological barriers that can be overcome by using a more advanced or complex piece of hardware. In principle you could type an essay for a literature class on your cell phone, or create a full-length animated feature using your laptop. But you would be better off writing your paper on a computer with a mouse, keyboard and a screen large enough to show you a full page of text, and since each frame in an animation is the result of hours of computing on a high-speed supercomputer a personal computer is not a practical choice for making a movie.

But in many cases the limits of what a computer can do are more than physical limits or things that can be fixed by choosing a different piece of hardware. These barriers are computational limitations: some problems simply cannot be solved using a computer. When
you want to send a message you need to know the recipient’s e-mail address. If you have the person in your list of contacts it’s trivial to look up their address, but if you don’t know the person you’re stuck – you might be able to use a search engine to find the address of the personnel manager at a company where you want to apply for a job, but you can’t just ask a computer to find the e-mail address of the cute guy you met at the coffee shop. When you are using a computer to manage your finances you can connect to your bank or credit card company to download a list of transactions, and you might be able to have a program do some calculations for different investment options, but you know the computer can’t choose the perfect investment because it can’t predict which companies will succeed or what future interest rates will be. Even in science fiction movies there are some tasks that only humans do – computers are, after all, only machines that are not capable of feeling or creativity. We ask our computers to help gather information but we don’t expect them to make important personal decisions. A machine can help find mileage estimates for different types of cars or admission and enrollment statistics for different colleges, but it can’t determine whether the fun of driving a flashy convertible outweighs the practicality of an all-wheel drive station wagon, and we can’t quantify all the attributes of different colleges and universities that would allow a computer to make a perfect decision on the best school to attend.

Scientists do not simply accept assertions that something cannot be done. A computer scientist, when thinking about whether a task can be accomplished by a computer, would want to know why something could not be computed. A scientist would look for proof that there is some fundamental aspect of a problem that would prevent it from being solved with a computer. Is it possible that some future computer will be able to surpass current limitations and solve some of the problems described above? Or is it the case that some problems do not have computational solutions? Can we be more precise about the difference between what is impractical and what is truly not computable?

When we try to analyze difficult problems and potential methods for solving them we quickly encounter several more questions that need to be addressed. Is there a way to characterize problems that we know have computational solutions? Do unsolvable problems have anything in common? What is the boundary between the computable and the non-computable? For that matter, what does it mean to “compute” something?

The generally accepted definition of a computation is that it is a sequence of simple, well-defined steps that lead to the solution of a problem. The problem must be defined exactly and unambiguously, and each step in the computation must be described in very specific terms. As a simple example of a computation, consider the problem of figuring out the average age of the group of students. To solve this problem the first step is to make it clear what is meant by “average” age; the most common definition is the arithmetic mean, which in this case would be the sum of the all the ages divided by the number of students. Next we need to make sure we have the necessary information to solve the problem. If we’re calculating the average age for a class and the roster contains only names we are stuck – we don’t have any way to calculate the age of each student. But if the roster includes birth dates, or it has student ID numbers and we have access to a central database of student records, we can compute the mean age. The steps in the calculation are to scan the list of students, and for each student compute their age and add it to a running total. After adding the last student’s age, divide the total by the number of students and the quotient will be the mean age of all students (Figure 1.1).
1.1 Computation

<table>
<thead>
<tr>
<th>Today’s Date: Oct 1, 2008</th>
<th>Average (Mean) Age:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sanchez, Maria Feb 14, 1988 20</td>
<td>((20 + 30 + 18 + 25 + 18) ÷ 5 = 22.2)</td>
</tr>
<tr>
<td>Sanders, Eric Mar 24, 1978 30</td>
<td></td>
</tr>
<tr>
<td>Sato, Noriko Oct 14, 1989 18</td>
<td></td>
</tr>
<tr>
<td>Singer, Fred Apr 30, 1983 25</td>
<td></td>
</tr>
<tr>
<td>Smith, John Feb 26, 1990 18</td>
<td></td>
</tr>
</tbody>
</table>

*Figure 1.1:* To compute the mean age of a group of students determine the age of each student by subtracting their birth date from today’s date, add up the ages, and divide by the number of students.

Note that there is nothing about this general notion of computation that involves the word “computer.” A computation is a process, a sequence of simple operations that leads from an initial state to the desired final result. The process can be carried out entirely by a person, or by a person using the help of mechanical or electronic devices, or completely automatically by a computer. The choice of which technology would be most effective depends on the situation. For a very small class, a person could write all the numbers in a single column on a sheet of paper, add them up, and then do the division using paper and pencil. For a larger group, a person would likely want to use a calculator or abacus – the state of the device will show the running total, so that after the last age is added the final step is to divide the total by the number of students. For very large groups – especially if the data is available electronically – the best solution would be to use a computer and program it to add the ages and do the division. No matter what technology is used, these situations all involve the same basic computation: in each case the average age is determined by calculating the sum of ages and then dividing by the number of students.

Now that we have an idea of what it means to compute something – a computation is a series of well-defined steps that lead from an unambiguous starting state to an accepted final result – let’s return to the assertion that some problems cannot be solved computationally. If we look more closely at examples of problems we don’t expect a computer to solve we can start to see some interesting and important differences:

- The problem of sending e-mail to that cute guy from the coffee shop is simply too vague. It lacks a well-defined input and there is no sequence of steps you can carry out to solve it. It’s worth noting this is a problem that cannot be solved by a human, either – a friend won’t be able to help you any more than your computer unless you supply a lot more information.

- The problem of deciding which university to attend has a well-defined set of inputs, in the form of a list of names of universities. Some things that go into making a decision are well defined. For most people, the cost of tuition and living expenses and the average GPA and SAT scores of entering freshmen are important factors. If those easily quantified sorts of things are the only criteria that are important to you, and if you can specify an accurate weight for each one, then you could perform a computation to decide which university is best for you. But for most people intangible qualities
like geographical location and recreational opportunities are important, and these are
hard if not impossible to quantify. People do solve problems that involve intangible
qualities, and it might seem like this is the sort of thing a person could do better
than a computer, but the “solutions” obtained by people aren’t like the solutions to
the average-age problem: they are recommendations, not unambiguous and reliably
correct answers.

• Suppose you want to invite a group of friends over to your house to watch a movie.
If you send them e-mail, you need to compose the message and send it. It would
be nice if you could open your cell phone, say “send a message to Aleah, Katie, and
Erica to see if they want to come over to watch a movie” and do something else while
your phone composes the e-mail, sends it, and negotiates with your friends’ phones
to find a time when everyone is available. This sort of interaction is not possible with
current computers, but researchers in the field of artificial intelligence are trying to
understand what is involved in these types of communications and developing com-
putational methods to carry them out. A human personal assistant can accomplish this
task, so this is an example of the kind of problem that humans can solve but computers
cannot. It is an open question of whether this problem is beyond the limits of compu-
tation, i.e. it very well might be possible for some future personal digital assistant to
accomplish this task.

• You might be tempted to think that a computer could help you win a chess tournament
by considering all possible moves you can make starting from a given board configu-
ration, then examining each possible move your opponent could make to each of your
moves, and eventually telling you the one that leads to a certain victory. This is an
example of a problem that should be solvable by a computer. The problem has a very
well defined input – the current configuration of the chessboard can be represented
as a set of symbols, much the same way as the ages of students in a class can be rep-
resented by strings of digits – and it also has a very specific and well-defined set of
operations, since the possible moves in chess can be defined as symbol manipulations
that change one chessboard configuration into another. But if someone tries to sell
you a program that does this computation you should do some “back of the envelope”
calculations first. The number of possible chess games has been estimated to be more
than $10^{43}$. Even if this hypothetical program runs on a supercomputer far more pow-
erful than the fastest current personal computer, and somehow computes 1 trillion
($10^{12}$) alternative board combinations per second, it will require $10^{43}/10^{12} = 10^{31}$
seconds, or roughly $10^{31}$ years. To put this in perspective, the age of the universe
is $10^{13}$ years. So even though this is a well-defined computation, it is one that no
computer will ever complete, and in that sense it is beyond the limits of computation.
It goes without saying that no human will ever do this computation, either. When
humans and computers play chess successfully they are using other strategies than
simple brute force exploration of all possible moves.

• Mathematicians in the 1930s made a startling discovery that some problems simply
have no solutions. In logic, these problems are called **undecidable functions**. This group
includes paradoxes like the familiar statement “this sentence is false” – it can’t be true,
because that would mean it is false, and likewise it can’t be false because that would
imply it is true. In mathematical terms it is simply undecidable. The computer science
equivalent of the undecidable functions are non-computable problems. The most well-known, the halting problem, asks whether it is possible to examine a running program and decide whether that program will ever terminate. Suppose you are running an application on your laptop and you see an icon on your screen that says the program is busy, and after five minutes you start to wonder whether the application is progressing very slowly or if it has crashed. It would be nice to be able to run another program that would examine the first one and come back and say “be patient, it will terminate” or “it crashed, you need to kill it and restart it.” A fundamental result in theoretical computer science tells us that this problem is logically equivalent to an undecidable function and that it is impossible to write such a “halt-checking” program. This is a different type of limitation than the one encountered by the chess-playing program. The chess player will compute the perfect game of chess if we are patient enough, so it is only unsolvable in a practical sense. The halt-checker requires us to evaluate an undecidable function, so it is beyond the limits of computation in that we know it is impossible to write a general purpose program that could carry out a sequence of steps that will let it determine if another program will terminate.

So is a computer an appliance or an artificial brain? The best answer is that a computer is a machine that carries out a computation. A computer might be considered an electronic brain because computations are sequences of mathematical and logical operations that humans (but not other animals) are capable of doing. Computers can do billions of these operations in one second, and the combined results of all these simple operations add up to truly remarkable feats that are well beyond what the smartest humans can accomplish. Computers are common household appliances because technology is inexpensive and widely available and the computations performed by computers have become an indispensable part of modern life.

**Exercises**

1. What are some of the tasks you use a personal computer for? What are some of the limitations of how well your computer carries out these tasks? Are the limits technical or computational?

2. Below is a list of fields where computers have been used – in some cases computers are well established and people who work in that area would be much less productive without computers, but in other cases the use of computers is still very tentative. Pick an area that interests you and write a short paper on how computers help solve problems in that area. Start by writing down some initial impressions and then do some research on the Internet to see what progress is being made by computer scientists and their colleagues from the problem domain (the web site for this book will have an up to date set of links for suggested reading). Questions to ask yourself as you do your research might include “What are the barriers to the use of computers in this field? Are those limits technological or computational? What are some of the social impacts and ethical issues arising from the use of computers in this area?”
   a) Medicine: Can computers diagnose illnesses or prescribe medicines? Help us advance our understanding of basic human physiology?
   b) Engineering: What role do computers play in the design and construction of new cars, airplanes, or other complex systems?
   c) Architecture: How are computers used to plan new buildings? How do they help architects come up with energy-efficient designs?
The History of Algorithms

The world *algorithm* comes from the name of a Persian mathematician, Mohammed ibn Mûsâ al-Khowârizmî (ca. 780-850), whose book on the use of Indian numerals introduced Europeans to the numeral 0. In Latin the name of the book was *Algoritmi de numero Indorum* (“Al-Khowârizmî Concerning the Hindu Art of Reckoning”).

The earliest known algorithm, known as Euclid’s method for finding the lowest common denominator of two numbers, dates from at least 300 BC. Other ancient algorithms include the Sieve of Eratosthenes, a method for making lists of prime numbers (and the basis for one of the projects in this book) and methods used by Sun Tzu and other Chinese mathematicians around 200 AD.

d) Meteorology: Are computer models being used to generate weather forecasts? Track hurricanes and other storms? How well do these models predict weather 24, 48, or 96 hours in advance?

e) Art and Entertainment: How have computers had an effect on music, video, or other artistic endeavors?

f) Libraries: What impacts are computers having on your university library or local community’s public library?

g) Banking and Finance: Do you do your banking on-line? Purchase and pay for any items using the internet?

h) Journalism: How does your local paper or school paper use computer technology? How are blogs or other social networking sites changing journalism?

i) Government: What role does computer and information technology play in local government in your area? Do you live in a place where electronic voting technology is used?

1.2 Algorithms

Computer scientists use the word *algorithm* to describe a method for solving a problem computationally. Although the term can be used to refer to any method for systematically solving a problem, and algorithms were widely used long before anyone thought of building a machine to perform the steps in a computation, today the term generally refers to a method that will be carried out automatically by a computer.

In the description of an algorithm, the steps have to be simple enough to be “understood” by a machine. One way to think of what a machine is capable of doing is to think in terms of symbols, such as numbers or letters. The steps in an algorithm are basically *symbol manipulations* like simple arithmetic operations or comparisons that define which words come before others in the alphabet. By putting together a large number of simple symbolic operations a machine can do very complex tasks, such as sorting long lists of names, counting millions of votes cast in an election, or using words to build an index of web pages gathered from the Internet.

As an example of how a task can be described as a sequence of symbol manipulations, the procedure for computing the mean age of a group of students is shown in the form of
1.2 Algorithms

Figure 1.2: An algorithm for computing the mean of a list of numbers. The lines on the right show the value of the sum and the items in the list after each step. The top line shows the initial value of the sum and the original input list. The remaining lines show the updated sum and list just after carrying out the operations on the line marked with a bullet symbol.

Figure 1.2: An algorithm for computing the mean of a list of numbers. The lines on the right show the value of the sum and the items in the list after each step. The top line shows the initial value of the sum and the original input list. The remaining lines show the updated sum and list just after carrying out the operations on the line marked with a bullet symbol.

an algorithm in Figure 1.2. The input to the algorithm is in the form of a set of symbols, in this case a list of numbers (for simplicity the figure only shows the calculations after the ages have been determined; we’ll assume some other algorithm has already transformed dates into ages). Each step in the algorithm is a matter of symbol manipulation, where the list is rewritten with one less item and a sum has been updated through simple arithmetic operations. The final output is a single number, again the result of a simple arithmetic operation.

The idea of an algorithm is the central concept in computer science. In order to use a computer to solve a problem, one must find (or invent) an algorithm for the computer to follow. An algorithm is characterized by:

- A precise statement of the starting conditions, which are the inputs to the algorithm.
- A specification of the final state of the algorithm, including the conditions which will cause the algorithm to terminate.
- A detailed description of the individual steps, each of which is a simple and straightforward operation that will help move the algorithm toward its final state.

Often the descriptions of the steps are given in English or another human language, as in the algorithm shown in Figure 1.2. This notation, which is sometimes called pseudocode, is sufficient for talking about the algorithm, or for describing the process to another person, or for trying to understand whether or not the algorithm works. But in order to run the algorithm on a computer the steps themselves also have to be written as a set of symbols, in the form of a program written in a computer language. This idea will be explored in the next section.

Exercises

Suppose you need to find a book at a library. Several methods for finding the book are described below. In each case, assume you have a precise specification of the book you want to find, i.e. you
know the title, author, and date of publication, and you also know the library owns the book. The two desired outcomes of your search are (a) you find the book, or (b) you learn the book has been checked out. Which of the following sets of operations can be considered steps of an algorithm for reaching your final goal given a description of a book?

1. Walk up to the first person you see, ask them where the book is.
2. Find a librarian, ask them where the book is.
3. Wait by the book return until the book you want is returned.
4. Use an electronic catalog (or card catalog, if it’s an old library) to find where the book is shelved, then use a map to find the shelf.
5. Start at the shelf nearest the door, then work systematically, shelf by shelf, through all shelves in the library.
6. Pick a shelf at random, see if your book is there; if not, pick another shelf at random and repeat.
7. Recruit ten friends; divide the library into ten regions; assign each friend to a different region; ask them to search every shelf in their region and report back to you.

There is no definitive yes or no answer for some of these methods; comments on whether they might or might not be considered an algorithm can be found at the end of the book.

1.3 The Stored Program Computer

While the idea of using a systematic method to carry out an advanced mathematical calculation is very old, the idea of using a machine to do the computation is more recent.

In post-Renaissance Europe mathematicians spent much of their time constructing tables of data. These tables, used for astronomy, navigation, and many other practical applications, were filled in by long and tedious hours spent doing basic arithmetic operations, where values in one row were simple extensions of values in previous rows. Several famous scientists, including Johannes Kepler (1571–1630), Blaise Pascal (1623–62), Gottfried Leibniz (1646–1726), and Charles Babbage (1791–1871), realized that the arithmetic operations used to create tables could, in principle, be done by machines. A quote from Leibniz summarized their longing for a machine to do the simpler operations and leave the more creative work for humans:

> Astronomers surely will not have to continue to exercise the patience which is required for computation. It is this that deters them from ... the construction of Ephemerides, from working on hypotheses, and from discussions of observations with each other. For it is unworthy of excellent men to lose hours like slaves in the labor of calculation which could safely be relegated to anyone else if machines were used.

There were a few successful designs for mechanical computers from the early 1600s through 1940, but these machines were limited, both by the types of operations they could perform and by the technology that was available. The most advanced of these machines were able to do the four basic arithmetic operations of addition, subtraction, multiplication, and division, but since the moving parts were made of wood and metal they would often quickly wear out. Babbage came up with an elegant design for his “analytical engine” but was never able to build a working system with the technology available to him. The movement toward automated calculation picked up momentum with the invention of the vacuum tube, or “electronic valve” as it was known in Europe, a technological development that was
Human Computers

In the nineteenth and early twentieth centuries the word “computer” was a job title.

[add description of projects to make log tables and navigation tables]

accompanied by increased demand for calculation by both sides in World War II. After the war the invention of the transistor, a much smaller form of electronic switch, led to further spreading of computer technology. In the 1970s techniques for making integrated circuits with thousands and eventually millions of transistors in a single electronic chip paved the way for the incredible spread of computer technology of the last 25 years.

Another profound innovation occurred in the 1940s. The explosive growth in the use of computers after 1950 was not only due to improvements in technology. Electronic switches were 1,000 times faster than the mechanical switches they replaced, and modern integrated circuits are 1,000,000 times faster than vacuum tubes, but the reason we are able to apply computers to such a wide variety of problems is that modern computers manipulate functions just as if they were other types of data. The memory chips inside our machines store both instructions and data, and many of the computations performed by computers operate on sequences of instructions that implement functions in addition to numbers, letters, and other types of data.

Several of the projects in this book will explore this profound idea in more depth. But to give you an idea of why it is so important, consider how to use a hand-held (non-programmable) calculator to compute the mean age of a group of students. If you use an abacus or electronic calculator or similar device you are responsible for choosing the order of operations: you decide when to add a new number to your running total, and you decide when to divide the total by the number of people in the group. But if you write a computer program to solve this problem, the instructions in your program are loaded into the computer’s memory. When you start the computation, these instructions guide the processor, and the computer takes over, adding numbers to the running total until the last one has been added, and then doing the division and displaying the result. The mathematical function called “arithmetic mean” by statisticians has been represented in a form that can be stored in a computer’s memory and be used to control the order of arithmetic operations carried out in the execution of the program. Because the instructions are stored in memory, the computer is much faster and can work on much larger problems: if the data is all collected ahead of time and stored in its memory, a computer could calculate the mean age of a group of a million people in a fraction of a second.

In the previous section the steps in an algorithm were described as operations on symbols.
Symbolic representations of data are stored in a computer’s memory so algorithms can work on them. For now it’s not important what sort of physical representation is used for these symbols – they could be stored in electronic, magnetic, optical, or any of a variety of different types of media. Now we can see that because functions are also stored in memory, there must be a way to represent the functions as strings of symbols, also. The simple operations that a computer can perform are each assigned a different symbol. From elementary school arithmetic you know that the basic operations of arithmetic have their own symbols, such as + for addition and ÷ for division. Computers typically have a few dozen simple operations, such as comparing two letters or moving data around in memory. It turns out that these few operations are all that is needed to implement the vast array of computer applications we use since these simple operations can be organized into powerful combinations to solve a wide variety of problems.

The idea that functions can be represented by a string of symbols goes back to the 1930s, when the idea that functions could be assigned numbers and thus handled like any other piece of data was used by logician Kurt Gödel (1906–78) as part of his famous proof that some mathematical statements are undecidable. John von Neumann (1903–57) is widely given credit for incorporating this idea in the design of the EDVAC, one of the first electronic computers and the first computer that stored its programs in the same memory it used for data. Many years after World War II declassified documents showed that English mathematician Alan Turing (1912–54), who, like Gödel, had investigated properties of systems in which functions are encoded as symbols and manipulated like any other data, had designed similar systems used by British forces to decipher messages that had been encrypted by the German military. Turing and von Neumann are two of the most influential figures in the history of computer science, and their ideas have had a profound impact still felt today. We will see their names again in descriptions of several projects later in the book.

Exercises
TBD

1.4 The Science of Computing

The main point of the previous sections is that computer science is more than “the study of computers.” Computer science is the study of computation. Computer scientists are interested in algorithms, in applying them to a wide variety of interesting and important problems, and in the limitations of computational solutions to problems. In the words of one influential computer scientist, Edsger Dijkstra (1930–2002), “computer science is no more about computers than astronomy is about telescopes.”

But there is no doubt technology plays an essential role in computer science. Algorithms and technology are the yin and yang of computer science, two different aspects that influence each other and work together produce a complex whole. Advances in technology open up new areas of computation, and development of algorithms in new application areas spurs the demand for better technology. The discussion in the preceding sections was intended to show that computer science is more than just computer technology – it is about the idea of computation, of designing algorithms to solve interesting and important problems, and about using technology to help carry out the computations.
A good example of the interaction between algorithms and technology is in the growth of the Internet. In the mid 1980s high speed networks connected research labs and large corporations, but people who had computers at home used their telephone lines to dial into a local computer center. The basic methods for sending messages between computers were well established, and there were widely used communication protocols for transmitting files, e-mail, and other types of information. The idea of hypertext was gathering increasing attention, including a successful computer application named HyperCard that ran on Apple Macintosh systems. In 1990 Tim Berners-Lee, a researcher at CERN, a European research laboratory, put these ideas together and created the hypertext communication protocol that became the basis of a “world-wide web” of information. In 1992 another group, working at the University of Illinois, developed a software application that used a graphical interface to display web pages, making it much easier for users to read documents that combined words and images. The popularity of the graphical interface attracted new users, and soon businesses started using the Internet to provide customer information. In an 18-month period from 1993 to 1995 the number of web servers (computers supplying web pages) grew from around 100 to over 20,000, and by 1997 there were over 650,000 servers. The huge demand for network connectivity led to advances in technology and the development of broadband and wireless connections. Many people expect this dizzying round of innovation to continue, as the wide availability of wireless communication is leading to research in both algorithms and technology for “wearable computers” and other forms of ubiquitous computing.

Computer science is a rapidly growing field. The outline in the following sections is meant to be a brief introduction to some of the main areas of computer science research, but it is by no means comprehensive, since it would be impossible to mention every area in such a small space. As in other areas of science, the “boundaries” between one area and the next are not always clear. Are researchers who study the inner workings of cells biologists or chemists? Or maybe biochemists, or chemical biologists? The answer may depend more on the methods and goals of the research than on the topic. A similar situation occurs at the frontiers of computer science – some research questions might be considered to be as much a part of mathematics or electrical engineering or biology as they are an area in computer science. The goal of this short outline is simply to give you a sense of the wide variety of problems that are being addressed by computer scientists and colleagues from other fields.

**Theoretical Computer Science**

Theoretical computer science has much in common with mathematics. Researchers in this field study the abstract properties of algorithms, asking questions and proving theorems about the limits of computability, relationships between classes of problems, and mathematical properties of programming languages (the “semantics” of languages).

Although it might seem that many of these questions might be of interest only to theoreticians, many of the problems in this area have important consequences for real computer systems. To take just one example, the most widely used method for transmitting credit card numbers and other sensitive information over the Internet uses a technique called “public key cryptography.” The security of this method depends on the fact that finding the prime factors of large numbers is a very difficult problem. Mathematicians have been searching for methods of factoring large numbers for many years; if anyone succeeds in finding an efficient algorithm for factoring large numbers it might lead to a security loophole for systems
based on public key cryptography.

**Computer Systems**

Research in computer systems is concerned with the design and implementation of a wide range of computers, from small single-user laptops to large multi-processor supercomputers. Working with researchers from electrical and computer engineering, computer scientists help create new generations of computers and computer networks.

Not all of the effort goes into designing new hardware. Software in a computer’s operating system (the “OS”) manages various resources, such as keeping track of files on disks or allocating room in memory to launch applications. Software, in the form of communication protocols, is essential for local and wide-area networks. For example, the original communication protocols designed for the Internet assumed all messages would be generated by two systems that were exchanging information in a simple two-way conversation. Recently organizations like television networks and on-line radio stations have started sending out streams of data carrying video and audio. Current research on “broadcast protocols” that would allow a server to send the same information to more than one client at a time has the goal of improving the quality of service that would allow these organizations to deliver more and higher quality media.

**Software**

Every field where computers are being used poses interesting challenges for computer scientists and provides opportunities for creating new algorithms and implementations.

A *database* is a software system designed to organize and access large amounts of information. For example, businesses need to keep track of customers, inventory, and supplies used in manufacturing. The earliest database systems contained text only, and were often kept in a single location within an organization. Research in database technology is expanding the type of information to include images and other types of data, and modern databases are distributed over wide geographical distances.

*Modeling and simulation* is another important area in applied computer science. Just as an airplane designer or automotive engineer might build a small-scale physical model to test properties of their designs, computer scientists design computational models to simulate real-world systems. For example, a computer simulation of a freeway system could help engineers decide where to place onramps and offramps, and computer simulations of car crashes allow engineers to test a wide variety of different designs before actually building their cars.

Advances in techniques for creating and displaying visual images has led to an explosive growth in the field of *computer graphics*. Computer-generated animations are generated by algorithms that compute the path of a ray of light as it leaves a lamp and bounces off objects and eventually reaches a virtual camera. These algorithms create complex scenes where objects cast shadows over or partially hide other objects. Computer-generated imagery (CGI) is also used to enhance movies recorded by physical camera, where graphical images created by a computer and edited into scenes recorded earlier.

A field known as *computational science* combines each of the three application areas described above. Here the word “computational” is an adjective, like “theoretical” or “experimental”, that describes a way of doing science. For example, researchers in the field
of bioinformatics study the complex interaction between DNA and protein involved in the process of gene expression. Large amounts of gene sequence data created by genome sequencing projects is stored in public databases available via the Internet. After downloading sequence data, a scientist can use applications that simulate the movement of the atoms in the protein molecules as they bind with DNA. To gain some insight into these complex interactions researchers often generate detailed images using scientific visualization software.

Artificial Intelligence

In 1950 Alan Turing published a paper titled “Computing Machinery and Intelligence”, in which he asked the question of whether a computer could be said to “think.” Turing refined the problem, and proposed that a machine could be considered intelligent if it could pass what has since come to be known as the Turing test: if a person could type questions on a terminal, but could not distinguish printed responses generated by a computer from answers typed by a person, then the computer could be considered “intelligent.”

Turing’s paper was one of the first in the new field of artificial intelligence. Computer scientists, working with psychologists and linguists, are working to understand the essential nature of language and trying to see if grammar rules for understanding and generating sentences can be implemented in a computer program. Researchers in this field are also trying to develop algorithms related to other aspects of intelligence, such as understanding spoken words and phrases, locating individual objects within a complex visual image, devising complex plans, and many other activities routinely carried out by humans.

The field of robotics is also related to artificial intelligence, especially when the goal is to build an autonomous machine that is able to process information from its environment and respond in what appears to be an intelligent manner, such as analyzing images of complex terrain in order to plan a path that it can navigate to reach a goal.

1.5 A Laboratory for Computational Experiments

This book is an introduction to computation, focussing on some of the important concepts used in algorithms and computational problem solving. Each chapter includes a project that will help you set up and run a program that will let you see these algorithms in action.

A useful analogy is to compare the exercises in this book to projects in a lab course that is part of an introduction to chemistry for nonmajors. Chemistry instructors design the projects, selecting materials and methods that are accessible to beginners, and students follow detailed instructions to carry out experiments that help them learn fundamental concepts about the properties of materials. Each chapter in this book has a computational project, organized as a tutorial with specific step-by-step instructions. The data and operations have been designed to help you learn important ideas in computer science as you set up and run programs that incorporate those ideas.

The laboratory for these computational experiments can be any small desktop or laptop computer. The software we will use is an interactive programming environment based on a language named Ruby. An interactive language works much like a calculator: users type in expressions, and the system then performs a calculation and prints a result. We will start with the building blocks provided by Ruby and put them together in interesting ways to carry out experiments on a wide variety of algorithms and application areas.
In the jargon of computer programming, Ruby belongs to the family of programming languages known as object-oriented languages. These languages use the word “object” not only to refer to pieces of data, such as numbers and text, but also to collections of data, such as lists of numbers, and to things like functions and rules that can be applied to other objects. In each chapter we will be using Ruby as our computational laboratory, setting up a small virtual world were we create objects and carry out operations designed to experiment with algorithms related to the main topic of that chapter.

Here are three examples of how an interactive environment based on an object-oriented programming language will be used to set up and carry out experiments in computer science:

- The first non-trivial algorithm presented in this book is The Sieve of Eratosthenes, a very old algorithm that has been used since the time of the ancient Greeks to make lists of prime numbers. The name of the algorithm is a hint to the basic idea: create a list of numbers, and then sift out those that are not prime. It is easy to set up a straightforward program that repeatedly works its way through the list. After experimenting with the method, we will find that it is not necessary to do as many repetitions as one might think, and the insight we gain from the initial experiments will be used to implement a more elegant version that does the minimal amount of work.

- After doing several more projects using simple objects like numbers, strings, and lists, we will be ready to tackle an algorithm that uses more substantial types of data. Another classic problem from the world of mathematics, known as the Traveling Salesman Problem, has provided a perspective for looking at several important real-world problems that require efficient schedules. We will use what is known as a genetic algorithm to experiment with one way of solving the Traveling Salesman Problem. Each object in this project will represent a complete tour. We will create a set of tours and put them in a “virtual petri dish,” then sit back and watch as the tours mutate and evolve, eventually giving rise to an efficient solution to the problem.

- One of the early milestones in artificial intelligence was a program named ELIZA, which gave the appearance of carrying out a conversation by playing the role of a psychiatrist. A user would type a sentence, and ELIZA would print a response. For example, if a person typed “I don’t like computers” the program might respond “Do computers worry you?” What was fascinating about ELIZA was how well it seemed to participate in a conversation, in spite of the fact that it only did very simple syntactic transformations on input sentences. For this project, we will create objects that represent the transformation rules, and run experiments that apply the rules to test sentences. Note that this project is an example of the important idea described in the previous section, in that rules are a type of function. Our program will represent functions as a type of data object and apply these objects to input sentences.

The important point to keep in mind as you are working your way through these projects is that Ruby, like any programming language, is simply a notation for describing an algorithm. Any time one learns a language one has to deal with myriad details, and at times it will seem like this book is more concerned with teaching the ins and outs of Ruby programming than it is with the ideas of computation. But try to keep in mind that the goal here is “literacy.” You will need to learn this new notation well enough to understand the basic steps of an
algorithm, but you will not have to memorize all the details that would be required to write your own new programs. If you have ever tried to learn a foreign language you probably know it is easier to read sentences in that language than it is to write, or to understand someone when they are speaking than it is to reply in that language. A similar thing will happen with Ruby. You will soon be able to understand the basic steps of an algorithm when you see them written as statements in Ruby, even if you find it difficult to write your own Ruby programs.

The reason we are using Ruby is that there is a tremendous benefit from using a real programming language as the notation for describing an algorithm. After an algorithm has been implemented in the form of a program we can run the program: we can apply it to different inputs, modify it, extend it, and carry out any number of experiments that will help us deepen our understanding of the algorithm.