OXFORD

POEMS THAT SOLVE PUZZLES

The History and Science of Algorithms

CHRIS BLEAKLEY

POEMS THAT SOLVE PUZZLES

The History and Science of Algorithms

Chris Bleakley

OXFORD
UNIVERSITY PRESS

OXFORD
UNIVERSITY PRESS

Great Clarendon Street, Oxford, OX2 6DP,
United Kingdom

Oxford University Press is a department of the University of Oxford.
It furthers the University's objective of excellence in research, scholarship,
and education by publishing worldwide. Oxford is a registered trade mark of
Oxford University Press in the UK and in certain other countries

First Edition published in 2020

Impression: 1

Published in the United States of America by Oxford University Press
198 Madison Avenue, New York, NY 10016, United States of America

British Library Cataloguing in Publication Data
Data available

Library of Congress Control Number: 2020933199

ISBN 978–0–19–885373–2

Printed and bound by
CPI Group (UK) Ltd, Croydon, CR0 4YY

Contents

Introduction

'One for you. One for me. One for you. One for me.' You are in the school yard. The sun is shining. You are sharing a packet of sweets with your best friend. 'One for you. One for me.' What you didn't realize back then was that sharing your sweets in this way was an enactment of an algorithm.

An algorithm is a series of steps that can be performed to solve an information problem. On that sunny day, you used an algorithm to share your sweets fairly. The input to the algorithm was the number of sweets in the packet. The output was the number of sweets that you and your friend each received. If the total number of sweets in the packet happened to be even, then both of you received the same number of sweets. If the total was odd, your friend ended up with one sweet more than you.

An algorithm is like a recipe. It is a list of simple steps that, if followed, transforms a set of inputs into a desired output. The difference is that an algorithm processes information, whereas a recipe prepares food. Typically, an algorithm operates on physical quantities that represent information.

Often, there are alternative algorithms for solving a given problem. You could have shared your sweets by counting them, dividing the total by two in your head, and handing over the correct number of sweets. The outcome would have been the same, but the algorithm—the means of obtaining the output—would have been different.

An algorithm is written down as a list of instructions. Mostly, these instructions are carried out in sequence, one after another. Occasionally, the next instruction to be performed is not the next sequential step but an instruction elsewhere in the list. For example, a step may require the person performing the algorithm to go back to an earlier step and carry on from there. Skipping backwards like this allows repetition of groups of steps—a powerful feature in many algorithms. The steps, 'One for you. One for me.' were repeated in the sweet sharing algorithm. The act of repeating steps is known as *iteration*.

If the number of sweets in the packet was even, the following iterative algorithm would have sufficed:

```
Repeat the following steps:
    Give one sweet to your friend.
    Give one sweet to yourself.
Stop repeating when the packet is empty.
```

In the exposition of an algorithm such as this, steps are usually written down line-by-line for clarity. Indentation normally groups inter-related steps.

If the number of sweets in the packet could be even or odd, the algorithm becomes a little more complicated. A decision-making step must be included. Most algorithms contain decision-making steps. A decision-making step requires the operator performing the algorithm to choose between two possible courses of action. Which action is carried out depends on a *condition*. A condition is a statement that is either true or false. The most common decision-making construct—'if-then-else'—combines a condition and two possible actions. 'If' the condition is true, 'then' the immediately following action (or actions) is performed. 'If' the condition is false, the step (or steps) after the 'else' are performed.

To allow for an odd number of sweets, the following decision-making steps must be incorporated in the algorithm:

```
If this is the first sweet or you just received a sweet,
then give this sweet to your friend,
else give this sweet to yourself.
```

The condition here is *compound*, meaning that it consists of two (or more) simple conditions. The simple conditions are 'this is the first sweet' together with 'you just received a sweet'. The two simple conditions are conjoined by an 'or' operation. The compound condition is true if either one of the simple conditions is true. In the case that the compound condition is true, the step 'give this sweet to your friend' is carried out. Otherwise, the step 'give this sweet to yourself' is performed.

The complete algorithm is then:

```
Take a packet of sweets as input.
Repeat the following steps:
    Take a sweet out of the packet.
    If this is the first sweet or you just received a sweet,
    then give this sweet to your friend,
    else give this sweet to yourself.
Stop repeating when the packet is empty.
Put the empty packet in the bin.
The sweets are now shared fairly.
```

Like all good algorithms, this one is neat and achieves its objective in an efficient manner.

The Trainee Librarian

Information problems crop up every day. Imagine a trainee librarian on their first day at work. One thousand brand new books have just been delivered and are lying in boxes on the floor. The boss wants the books to be put on the shelves in alphabetical order by author name, as soon as possible. This is an information problem and there are algorithms for solving it.

Most people would intuitively use an algorithm called Insertion Sort (Figure I.1). Insertion Sort operates in the following way:

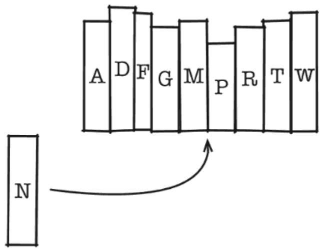

Figure I.1 Insertion Sort in action.

```
Take a pile of unsorted books as input.
Repeat the following steps:
    Pick up a book.
    Read the author's name.
    Scan across the shelf until you find where the book should
        be inserted.
    Shift all the books after that point over by one.
    Insert the new book.
Stop repeating when there are no books left on the floor.
The books are now sorted.
```

At any moment in time, the books on the floor are unsorted. One by one, the books are transferred to the shelf. Every book is placed on the shelf in alphabetical order. As a result, the books on the shelf are always in order.

Insertion Sort is easy to understand and works but is slow. It is slow because, for every book taken from the floor, the librarian has to scan past or shift every book already on the shelf. At the start, there are very few books on the shelf, so scanning and shifting is fast. At the end, our librarian has almost 1,000 books on the shelf. On average, putting a book in the right place requires 500 *operations*, where an operation is an author name comparison or a book shift. Thus, sorting all of the books takes 500,000 (1,000 × 500) operations, on average. Let's say that a single operation takes one second. That being the case, sorting the books using Insertion Sort will take around seventeen working days. The boss isn't going to be happy.

A faster alternative algorithm—Quicksort—was invented by computer scientist Tony Hoare in 1962. Hoare was born in Sri Lanka, to British parents in 1938. He was educated in England and attended Oxford University before entering academia as a lecturer. His method for sorting is a divide-and-conquer algorithm. It is more complicated than Insertion Sort but, as the name suggests, much faster.

Quicksort (Figure I.2) splits the pile of books into two. The split is governed by a *pivot letter*. Books with author names before the pivot letter are put on a new pile to the left of the current pile. Books with author names after the pivot are placed on a pile to the right. The resulting piles are then split using new pivot letters. In doing so, the piles are kept in sequence. The leftmost pile contains the books that come first in the alphabet. The next pile holds the books that come second, and so on. This pile-splitting process is repeated for the largest pile until

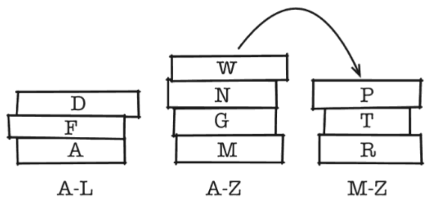

Figure I.2 Quicksort in action.

the biggest stack contains just five books. The piles are then sorted separately using Insertion Sort. Finally, the sorted piles are transferred, in order, to the shelf.

For maximum speed, the pivot letters should split the piles into two halves.

Let's say that the original pile contains books from A to Z. A good choice for the first pivot would likely be M. This would give two new piles: A–L and M–Z (Figure I.2). If the A–L pile is larger, it will be split next. A good pivot for A–L might be F. After this split, there will be three piles: A–E, F–L, and M–Z. Next, M–Z will be split and so on. For twenty books, the final piles might be: A–C, D–E, F–L, M–R, and S–Z. These piles are ordered separately using Insertion Sort and the books transferred pile-after-pile onto the shelf.

The complete Quicksort algorithm can be written down as follows:

```
Take a pile of unsorted books as input.
Repeat the following steps:
    Select the largest pile.
    Clear space for piles on either side.
    Choose a pivot letter.
    Repeat the following steps:
        Take a book from the selected pile.
        If the author name is before the pivot letter,
        then put the book on the pile to the left,
        else put the book on the pile to the right.
    Stop repeating when the selected pile is empty.
Stop repeating when the largest pile has five books or less.
Sort the piles separately using Insertion Sort.
Transfer the piles, in order, to the shelf.
The books are now sorted.
```

Quicksort uses two repeating sequences of steps, or *loops*, one inside the other. The outer repeating group deals with all of the piles. The inner group processes a single pile.

Quicksort is much faster than Insertion Sort for large numbers of books. The trick is that splitting a pile is fast. Each book need only be compared with the pivot letter. Nothing needs to be done to the other books—no author name comparisons, no book shifts. Applying Insertion Sort at the end of Quicksort is efficient since the piles are small. Quicksort only requires about 10,000 operations to sort 1,000 books. The exact number of operations depends on how accurately the pivots halve the piles. At one second per operation, the job takes less than three hours—a big improvement on seventeen working days. The boss will be pleased.

Clearly, an algorithm's speed is important. Algorithms are rated according to their *computational complexity*. Computational complexity relates the number of steps required for execution of an algorithm to the number of inputs. The computational complexity of Quicksort is significantly lower than that of Insertion Sort.

Quicksort is called a divide-and-conquer algorithm because it splits the original large problem into smaller problems, solves these smaller problems separately, and then assembles the partial solutions to form the complete solution. As we will see, divide-and-conquer is a powerful strategy in algorithm design.

Many algorithms have been invented for sorting, including Merge Sort, Heapsort, Introsort, Timsort, Cubesort, Shell Sort, Bubble Sort, Binary Tree Sort, Cycle Sort, Library Sort, Patience Sorting, Smoothsort, Strand Sort, Tournament Sort, Cocktail Sort, Comb Sort, Gnome Sort, UnShuffle Sort, Block Sort, and Odd-Even Sort. All of these algorithms sort data, but each is unique. Some are faster than others. Some need more storage space than others. A few require that the inputs are prepared in a special way. A handful have simply been superseded.

Nowadays, algorithms are inextricably linked with computers. By definition, a computer is a machine that performs algorithms.

The Algorithm Machine

As discussed, an algorithm is an abstract method for solving a problem. An algorithm can be performed by a human or a computer. Prior to execution on a computer, an algorithm must be encoded as a list of instructions that the computer can carry out. A list of computer instructions is called a *program*. The great advantage of a computer is that it can automatically execute large numbers of instructions one-after-another at high speed. Surprisingly, a computer need not support a great variety of instructions. A few basic instruction types will suffice. All that is needed are instructions for data storage and retrieval, arithmetic, logic, repetition, and decision-making. Algorithms can be broken down into simple instructions such as these and executed by a computer.

The list of instructions to be performed and the data to be operated on are referred to as the computer *software*. In a modern computer, software is encoded as electronic voltage levels on microscopic wires. The computer *hardware*—the physical machine—executes the program one instruction at a time. Program execution causes the input data to be processed and leads to creation of the output data.

There are two reasons for the phenomenal success of the computer. First, computers can perform algorithms much more quickly than humans. A computer can perform billions of operations per second, whereas a human might do ten. Second, computer hardware is *general-purpose*, meaning that it can execute any algorithm. Just change the software and a computer will perform a completely different task. This gives the machine great flexibility. A computer can perform a wide range of duties—everything from word processing to video games. The key to this flexibility is that the program dictates what the general-purpose hardware does. Without the software, the hardware is idle. It is the program that animates the hardware.

The algorithm is the abstract description of what the computer must do. Thus, in solving a problem, the algorithm is paramount. The algorithm is the blueprint for what must be done. The program is the precise, machine-executable formulation of the algorithm. To solve an information problem, a suitable algorithm must first be found. Only then can the program be typed into a computer.

The invention of the computer in the mid-twentieth century gave rise to an explosion in the number, variety, and complexity of algorithms. Problems that were once thought impossible to solve are now routinely dispatched by cheap computers. New programs are released on a daily basis, extending the range of tasks that computers can undertake.

Algorithms are embedded in the computers on our desktops, in our cars, in our television sets, in our washing machines, in our smartphones, on our wrists, and, soon, in our bodies. We engage a plethora of algorithms to communicate with our friends, to accelerate our work, to play games, and to find our soulmates. Algorithms have undoubtedly made our lives easier. They have also provided humankind with unprecedented access to information. From astronomy to particle physics, algorithms have enhanced our comprehension of the universe. Recently, a handful of cutting-edge algorithms have displayed superhuman intelligence.

All of these algorithms are ingenious and elegant creations of the human mind. This book tells the story of how algorithms emerged from the obscure writings of ancient scholars to become one of the driving forces of the modern computerized world.

1

Ancient Algorithms

Go up on to the wall of Uruk, Ur-shanabi, and walk around,
Inspect the foundation platform and scrutinise the brickwork!
Testify that its bricks are baked bricks,
And that the Seven Counsellors must have laid its foundations!
One square mile is city, one square mile is orchards,
one square mile is clay pits,
as well as the open ground of Ishtar's temple.
Three square miles and the open ground comprise Uruk.

Unknown author, translated by Stephanie Dalley
The Epic of Gilgamesh, circa 2,000 BCE[2]

The desert has all but reclaimed Uruk. Its great buildings are almost entirely buried beneath accretions of sand, their timbers disintegrated. Here and there, clay brickwork is exposed, stripped bare by the wind or archaeologists. The abandoned ruins seem irrelevant, forgotten, futile. There is no indication that seven thousand years ago, this land was the most important place on Earth. Uruk, in the land of Sumer, was one of the first cities. It was here, in Sumer, that civilization was born.

Sumer lies in southern Mesopotamia (Figure 1.1). The region is bounded by the Tigris and Euphrates rivers, which flow from the mountains of Turkey in the north to the Persian Gulf in the south. Today, the region straddles the Iran–Iraq border. The climate is hot and dry, and the land inhospitable, save for the regular flooding of the river plains. Aided by irrigation, early agriculture blossomed in the 'land between the rivers'. The resulting surplus of food allowed civilization to take hold and flourish.

The kings of Sumer built great cities—Eridu, Uruk, Kish, and Ur. At its apex, Uruk was home to sixty thousand people. All of life was there—family and friends, trade and religion, politics and war. We know this because writing was invented in Sumer around 5,000 years ago.

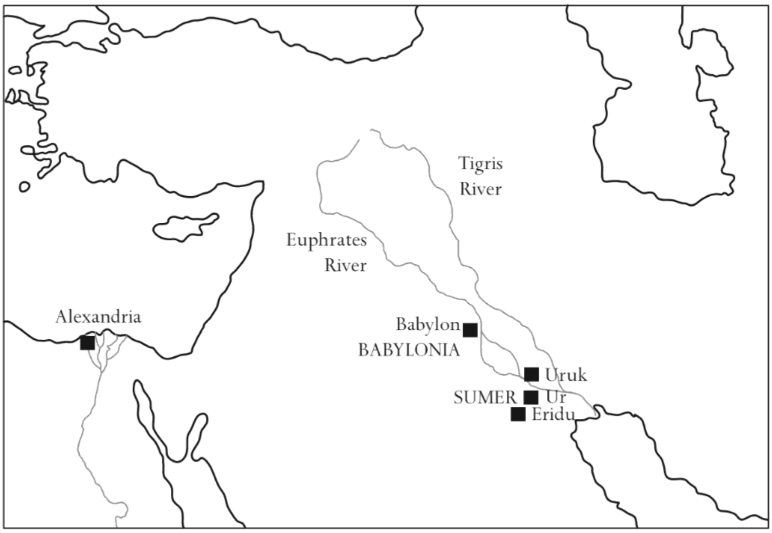

Figure 1.1 Map of ancient Mesopotamia and the later port of Alexandria.

Etched in Clay

It seems that writing developed from simple marks impressed on wet clay tokens. Originally, these tokens were used for record keeping and exchange. A token might equate to a quantity of gain or a headcount of livestock. In time, the Sumerians began to inscribe more complex patterns on larger pieces of clay. Over the course of centuries, simple pictograms evolved into a fully formed writing system. That system is now referred to as *cuneiform* script. The name derives from the script's distinctive 'wedge shaped' markings, formed by impressing a reed stylus into wet clay. Symbols consisted of geometric arrangements of wedges. These inscriptions were preserved by drying the wet tablets in the sun. Viewed today, the tablets are aesthetically pleasing—the wedges thin and elegant, the symbols regular, the text neatly organized into rows and columns.

The invention of writing must have transformed these communities. The tablets allowed communication over space and time. Letters could be sent. Deals could be recorded for future reference. Writing facilitated the smooth operation and expansion of civil society.

For a millennium, cuneiform script recorded the Sumerian language. In the twenty-fourth century BCE, Sumer was invaded by the armies of the Akkadian Empire. The conquerors adapted the Sumerian writing methods to the needs of their own language. For a period, both languages were used on tablets. Gradually, as political power shifted, Akkadian became the exclusive language of the tablets.

The Akkadian Empire survived for three centuries. Thereafter, the occupied city states splintered, later coalescing into Assyria in the north and Babylonia in south. In the eighteenth century BCE, Hammurabi, King of Babylon, reunited the cities of Mesopotamia. The city of Babylon became the undisputed centre of Mesopotamian culture. Under the King's direction, the city expanded to include impressive monuments and fine temples. Babylonia became a regional superpower. The Akkadian language, and its cuneiform script, became the *lingua franca* of international diplomacy throughout the Middle East.

After more than one millennium of dominance, Babylon fell, almost without resistance, to Cyrus the Great, King of Persia. With its capital in modern Iran, the Persian Empire engulfed the Middle East. Cyrus's Empire stretched from the Bosporus strait to central Pakistan and from the Black Sea to the Persian Gulf. Persian cuneiform script came to dominate administration. Similar at first glance to the Akkadian tablets, these new tablets used the Persian language and an entirely different set of symbols. Use of the older Akkadian script dwindled. Four centuries after the fall of Babylon, Akkadian fell into disuse. Soon, all understanding of the archaic Sumerian and Akkadian cuneiform symbols was lost.

The ancient cities of Mesopotamia were gradually abandoned. Beneath the ruins, thousands of tablets—records of a dead civilization lay buried. Two millennia passed.

Uncovered at Last

European archaeologists began to investigate the ruins of Mesopotamia in the nineteenth century. Their excavations probed the ancient sites. The artefacts they unearthed were shipped back to Europe for inspection. Amongst their haul lay collections of the inscribed clay tablets. The tablets bore writing of some sort, but the symbols were now incomprehensible.

Assyriologists took to the daunting task of deciphering the unknown inscriptions. Certain oft repeated symbols could be identified and decoded. The names of kings and provinces became clear. Otherwise, the texts remained impenetrable.

The turning point for translators was the discovery of the Behistun (Bīsitūn) Inscription. The Inscription consists of text accompanied by a relief depicting King Darius meting out punishment to handcuffed prisoners. Judging by their garb, these captives were from across the Persian Empire. The relief is carved high on a limestone cliff face overlooking an ancient roadway in the foothills of the Zagros mountains in western Iran. The Inscription is an impressive fifteen metres tall and twenty five metres wide.

The significance of the Inscription only became apparent after Sir Henry Rawlinson—a British East India Company officer—visited the site. Rawlinson scaled the cliff and made a copy of the cuneiform text. In doing so, he spotted two other inscriptions on the cliff. Unfortunately, these were inaccessible. Undaunted, Rawlinson returned in 1844 and, with the aid of a local lad, secured impressions of the other texts.

It transpired that the three texts were in different languages—Old Persian, Elamite, and Babylonian. Crucially, all three recounted the same propaganda—a history of the King's claims to power and his merciless treatment of rebels. Some understanding of Old Persian had persisted down through the centuries. Rawlinson compiled and published the first complete translation of the Old Persian text two years later.

Taking the Old Persian translation as a reference, Rawlinson and a loose cadre of enthusiasts succeeded in decoding the Babylonian text. The breakthrough was the key to unlocking the meaning of the Akkadian and Sumerian tablets.

The tablets in the museums of Baghdad, London, and Berlin were revisited. Symbol by symbol, tablet by tablet, the messages of the Sumerians, Akkadians, and Babylonians were decoded. A long-lost civilization was revealed.

The messages on the earliest tablets were simplistic. They recorded major events, such as the reign of a king or the date of an important battle. Over time, the topics became more complex. Legends were discovered, including the earliest written story: *The Epic of Gilgamesh*. The day-to-day administration of civil society was revealed—laws, legal contracts, accounts, and tax ledgers. Letters exchanged by kings and queens were found, detailing trade deals, proposals of royal marriage, and threats of war. Personal epistles were uncovered, including love poems and magical curses. Amid the flotsam and jetsam of daily life, scholars stumbled upon the algorithms of ancient Mesopotamia.

Many of the extant Mesopotamian algorithms were jotted down by students learning mathematics. The following example dates from the Hammurabi dynasty (1,800 to 1,600 BCE), a time now known as the Old Babylonian period. Dates are approximate; they are inferred from the linguistic style of the text and the symbols employed. This algorithm was pieced together from fragments held in the British and Berlin State Museums. Parts of the original are still missing.

The tablet presents an algorithm for calculating the length and width of an underground water cistern. The presentation is formal and consistent with other Old Babylonian algorithms. The first three lines are a concise description of the problem to be solved. The remainder of the text is an exposition of the algorithm. A worked example is interwoven with the algorithmic steps to aid comprehension.[5]

A cistern.
The height is 3.33, and a volume of 27.78 has been excavated.
The length exceeds the width by 0.83.
You should take the reciprocal of the height, 3.33, obtaining 0.3.
Multiply this by the volume, 27.78, obtaining 8.33.
Take half of 0.83 and square it, obtaining 0.17.
Add 8.33 and you get 8.51.
The square root is 2.92.
Make two copies of this, adding to the one 0.42 and subtracting
from the other.
You find that 3.33 is the length and 2.5 is the width.
This is the procedure.

The question posed is to calculate the length and width of a cistern, presumably of water. The volume of the cistern is stated, as is its height. The required difference between the cistern's length and width is specified. The actual length and width are to be determined.

The phrase, 'You should', indicates that what follows is the method for solving the problem. The result is followed by the declaration, 'This is the procedure', which signifies the end of the algorithm.

The Old Babylonian algorithm is far from simple. It divides the volume by the height to obtain the area of the base of the cistern.

Simply taking the square root of this area would give the length and width of a square base. An adjustment must be made to create the desired rectangular base. Since a square has minimum area for a given perimeter, the desired rectangle must have a slightly larger area than the square base. The additional area is calculated as the area of a square with sides equal to half the difference between the desired length and width. The algorithm adds this additional area to the area of the square base. The width of a square with this combined area is calculated. The desired rectangle is formed by stretching this larger square. The lengths of two opposite sides are increased by half of the desired length–width difference. The length of the other two sides is decreased by the same amount. This produces a rectangle with the correct dimensions.

Decimal numbers are used the description above. In the original, the Babylonians utilized *sexagesimal* numbers. A sexagesimal number system possesses sixty unique digits (0–59). In contrast, decimal uses just ten digits (0–9). In both systems, the weight of a digit is determined by its position relative to the fractional (or decimal) point. In decimal, moving right-to-left, each digit is worth ten times the preceding digit. Thus, we have the units, the tens, the hundreds, the thousands, and so on. For example, the decimal number 421 is equal to four hundreds plus two tens plus one unit. In sexagesimal, moving right-to-left from the fractional point, each digit is worth sixty times the preceding one. Conversely, moving left-to-right, each column is worth a sixtieth of the previous one. Thus, sexagesimal 1,3.20, means one sixty plus three units plus twenty sixtieths, equal to $63\frac{20}{60}$ or 63.333 decimal. Seemingly, the sole advantage of the Old Babylonian system is that thirds are much easier to represent than in decimal.

To the modern reader, the Babylonian number system seems bizarre. However, we use it every day for measuring time. There are sixty seconds in a minute and sixty minutes in an hour. The time 3:04 am is 184 ($3 \times 60 + 4 \times 1$) minutes after midnight.

Babylonian mathematics contains three other oddities. First, the fractional point wasn't written down. Babylonian scholars had to infer its position based on context. This must have been problematic—consider a price tag with no distinction between dollars and cents! Second, the Babylonians did not have a symbol for zero. Today, we highlight the gap left for zero by drawing a ring around it (0). Third, division was performed by multiplying by the reciprocal of the divisor. In other words, the Babylonians didn't divide by two, they multiplied by a half.

In practice, students referred to precalculated tables of reciprocals and multiplications to speed up calculations.

A small round tablet shows the breathtaking extent of Babylonian mathematics. The tablet—YBC 7289—resides in Yale University's Old Babylonian Collection (Figure 1.2). Dating to around 1,800 to 1,600 BCE, it depicts a square with two diagonal lines connecting opposing corners. The length of the sides of the square are marked as thirty units. The length of the diagonal is recorded as thirty times the square root of two.

The values indicate knowledge of the Pythagorean Theorem, which you may recall from school. It states that, in triangles with a right angle, the square (a value multiplied by itself) of the length of the hypotenuse (the longest side) is equal to the sum of the squares of the lengths of the other two sides.

What is truly remarkable about the tablet is that it was inscribed 1,000 years before the ancient Greek mathematician Pythagoras was born. For mathematicians, the discovery is akin finding an electric light bulb in a Viking camp! It raises fundamental questions about the history of mathematics. Did Pythagoras invent the algorithm, or did he learn of it during his travels? Was the Theorem forgotten and independently reinvented by Pythagoras? What other algorithms did the Mesopotamians invent?

YBC 7289 states that the square root of two is 1.41421296 (in decimal). This is intriguing. We now know that the square root of two is 1.414213562, to nine decimal places. Remarkably, the value on the tablet

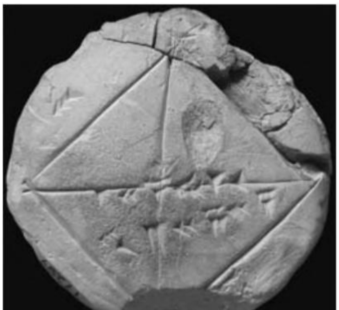

Figure 1.2 Yale Babylonian Collection tablet 7289. (*YBC 7289, Courtesy of the Yale Babylonian Collection.*)

is accurate to almost seven digits, or 0.0000006. How did the Babylonians calculate the square root of two so precisely?

Computing the square root of two is not trivial. The simplest method is Heron of Alexandria's approximation algorithm. There is, of course, the slight difficulty that Heron lived 1,500 years (*c.* 10–70 CE) after YBE 7289 was inscribed! We must assume that the Babylonians devised the same method.

Heron's algorithm reverses the question. Instead of asking 'What is the square root of two?', Heron enquires 'What number multiplied by itself gives two?' Heron's algorithm starts with a guess and successively improves it over a number of iterations:

```
Make a guess for the square root of two.
Repeatedly generate new guesses as follows:
    Divide two by the current guess.
    Add the current guess.
    Divide by two to obtain a new guess.
Stop repeating when the two most recent guesses are almost
  equal.
The latest guess is an approximation for the square root of two.
```

Let's say that the algorithm begins with the extremely poor guess of:

2.

Dividing 2 by 2 gives 1. Adding 2 to this, and dividing by 2 gives:

1.5.

Dividing 2 by 1.5 gives 1.333. Adding 1.5 to this and dividing by 2 again gives:

1.416666666.

Repeating once more gives:

1.41421568.

which is close to the true value.

How does the algorithm work? Imagine that you know the true value for the square root of two. If you divide two by this number, the result is exactly the same value—the square root of two.

Now, imagine that your guess is greater than the square root of two. When you divide two by this number, you obtain a value less than the square root of two. These two numbers frame the true square root—one is too large, the other is too small. An improved estimate can be obtained by calculating the average of these two numbers (i.e. the sum divided by two). This gives a value midway between the two framing numbers.

This procedure—division and averaging—can be repeated to further refine the estimate. Over successive iterations, the estimates converge on the true square root.

It is worth noting that the process also works if the guess is less than the true square root. In this case, the number obtained by means of division is too large. Again, the two values frame the true square root.

Even today, Heron's method is used to estimate square roots. An extended version of the algorithm was utilized by Greg Fee in 1996 to confirm enumeration of the square root of two to ten million digits.

Mesopotamian mathematicians went so far as to invoke the use of memory in their algorithms. Their command 'Keep this number in your head' is an antecedent of the data storage instructions available in a modern computer.

Curiously, Babylonian algorithms do not seem to have contained explicit decision-making ('if–then–else') steps. 'If–then' rules were, however, used by the Babylonians to systematize non-mathematical knowledge. The Code of Hammurabi, dating from 1754 BCE, set out 282 laws by which citizens should live. Every law included a crime and a punishment:[8]

> If a son strike a father, they shall cut off his fingers.
>
> If a man destroy the eye of another man, they shall destroy his eye.

If–then constructs were also used to capture medical knowledge and superstitions. The following omens come from the library of King Ashurbanipal in Nineveh around 650 BCE:[9]

> If a town is set on a hill, it will not be good for the dweller within that town.
>
> If a man unwittingly treads on a lizard and kills it, he will prevail over his adversary.

Despite the dearth of decision-making steps, the Mesopotamians solved a wide variety of problems by means of algorithms. They

image
not
available

Euclid's Elements was copied, translated, recopied, and retranslated. What is now known as Euclid's algorithm is contained in Book VII.

Euclid's algorithm calculates the greatest common divisor of two numbers (also called the GCD, or the largest common factor). For example, 12 has six divisors (i.e. integers that divide evenly into it). The divisors are 12, 6, 4, 3, 2, and 1. The number 18 also happens to have six divisors: 18, 9, 6, 3, 2, and 1. The greatest common divisor of both 12 and 18 is therefore 6.

The GCD of two numbers can be found by listing all of their divisors and searching for the largest value common to both lists. This approach is fine for small numbers, but is very time consuming for large numbers. Euclid came up with a much faster method for finding the GCD of two numbers. The method has the advantage of only needing subtraction operations. Cumbersome divisions and multiplications are avoided.

Euclid's algorithm operates as follows:

```
Take a pair of numbers as input.
Repeat the following steps:
    Subtract the smaller from the larger.
    Replace the larger of the pair with the value obtained.
Stop repeating when the two numbers are equal.
The two numbers are equal to the GCD.
```

As an example, take the following two inputs:

12, 18.

The difference is 6. This replaces 18, the larger of the pair. The pair is now:

12, 6.

The difference is again 6. This replaces 12, giving the pair:

6, 6.

Since the numbers are equal, the GCD is 6.

It is not immediately obvious how the algorithm works. Imagine that you know the GCD at the outset. The two starting numbers must both be multiples of the GCD since the GCD is a divisor of both. Since both inputs are multiples of the GCD, the difference between them must also be a multiple of the GCD. By definition, the difference between the