CHAPTER 1
Count the Dots - Binary Numbers

Learning Objectives:

- This lesson aims to introduce students to binary numbers and binary code as a computer's "language" of storing information.
- Representation of data in a computer

Introduction

Have you ever wished you could learn to talk to a computer? Computers use binary coding to work. Binary is a specific code that represents numbers following a series of 0s and 1s. The Codes are like a secret means of communication that computers speak. I am sure that you always thought in your minds about the computer being the smartest thing in the world, but computers are just super awesome at following specific and detailed instructions. Binary codes are used to tell our computers what we want them to do for us.

In this module, we'll learn all about binary numbers, how to read them, and how we use them to communicate with computers. Read on to learn 'computer talk!'

Base Ten and Place Value

There is what we call the "Decimal Number System" or DNS for short. The decimal number system is based on ten, sometimes called base ten. This means that in our number system popularly called the counting numbers, we represent amounts using ten numerals i.e 0 - 9 : 0, 1, 2, 3, 4, 5, 6, 7, 8, and 9. When we need to write a larger number, we reuse the same numerals and make use of place value.

Place value means that each place or column, in a number has a particular value. Each place to the left increases by a power of ten.

What does binary mean?

The binary number system, popularly referred to as base two, works similarly to the base ten systems. Because it is base two, we can only use two numerals: 0 and 1. Just as in base 10, we also use place value in base two: each place to the left increases in size by a factor of 2.

How Data is Represented in A Computer

To explain how data is represented in a computer, we will need a set of five cards, as shown below, with dots on one side and nothing on the other. The cards can be made out of waste papers or cardboards. Five students will be chosen at random to hold the binary paper cards where everyone will see it conveniently. The cards should be in the following order:

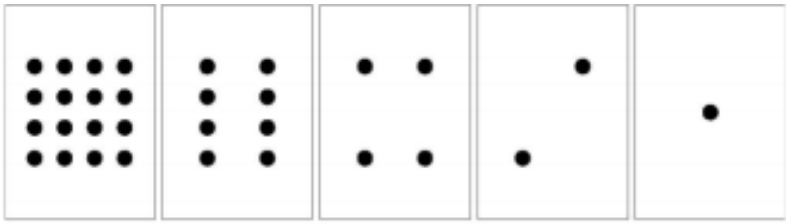

Figure *1.0 Binary Paper Card.*

Activity

As you receive the cards (from right to left), see if you can guess how many dots are on the next card. What do you notice about the number of dots on the cards? (Each card has twice as many as the card to its right.)

How many dots would the next card have if we carried on to the left? (32) The next…? (64)

We can use these cards to make numbers by turning some of them face down and adding up the dots that are showing. Ask the students to show 6 dots (4-dot and 2-dot cards), then 15 (8-, 4-, 2- and 1-dot cards), then 21 (16, 4 and 1)… The only rule is that a card has to be completely visible, or completely hidden. What is the smallest number of dots possible? (They may answer one, but it's zero). Now try counting from zero onwards.

We need to look closely at how the cards change to see if they can see a pattern in how the cards flip (each card flips half as often as the one to its right). You may like to try this with your friends by forming a group. When a binary number card is not showing, it is represented by a zero. When it is showing, it is represented by a one. This is the binary number system.

Exercise

1. As an exercise you are required to make 01001 using your binary cards. What number is this in decimal?
2. What would 17 be in binary?

From exercise 1 above, if you get a binary card representation like the one below then you can continue to the next paragraph.

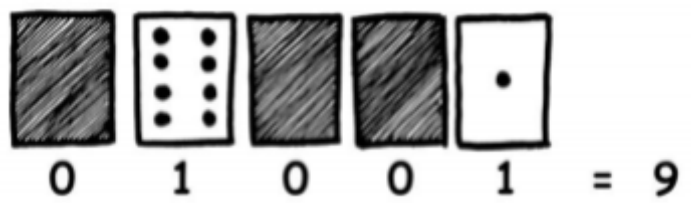

Figure *1.1 Solution To Exercise.*

In a computer, data is stored and transmitted as a series of zeros and ones. Each binary paper card has a number of bits on it. Each binary digit is called a *bit.* We just need to read off the number of dots that are showing to determine what number is being represented by the binary paper card. When a binary paper card is not showing, it is represented by a zero otherwise, it is represented by a one. This is the binary number system.

Learning how to count

So, you thought you knew how to count? Well, here is a new way to do it! Did you know that computers use only zero and one? Everything that you see or hear on the computer—words, pictures, numbers, movies and even sound is stored using just those two numbers! In this chapter, we will learn how you can send messages to your friends using exactly the same method as a computer.

Instructions

Lay your binary cards on your table with the 16-dot card on the left as shown below:

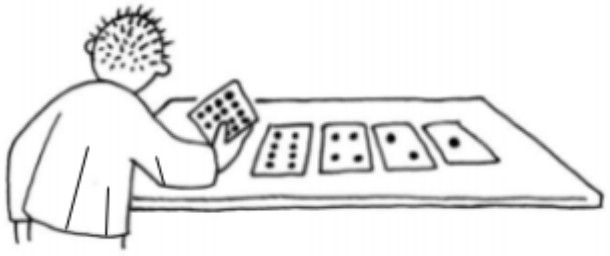

Figure *1.2 How to lay your binary cards.*

Make sure the cards are placed in exactly the same order.

Now flip the cards so exactly 5 dots show—keep your cards in the same order!

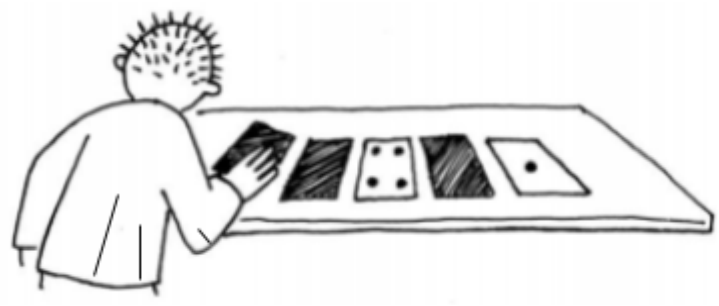

Figure *1.3 How to flip your binary cards.*

As an exercise, do the following:

1. Find out how to get 3, 12, 19. Is there more than one way to get any number?
2. What is the biggest number you can make?
3. What is the smallest?
4. Is there any number you can't make between the smallest and biggest numbers?

The binary system uses zero and one to represent whether a card is face up or not. 0 shows that a card is hidden, and 1 means that you can see the dots.

For example:

Figure *1.4 An example of a binary representation of the number 9 with some cards faced up and down.*

Sending Messages To Your Friends

John is trapped on the top floor of his tree house. It's christmas and he wants to to his friend's place with his presents. What can he do? He has tried calling, even yelling, but there is no one around. Across the street he can see some computer person still working away late into the night. How could he attract her attention? John looks around to see what he could use. Then he has a brilliant idea—he can use the Christmas tree lights to send her a message! He finds all the lights and plugs them in so he can turn them on and off. He uses a simple binary code, which he knows the woman across the street is sure to understand. Can you work it out?

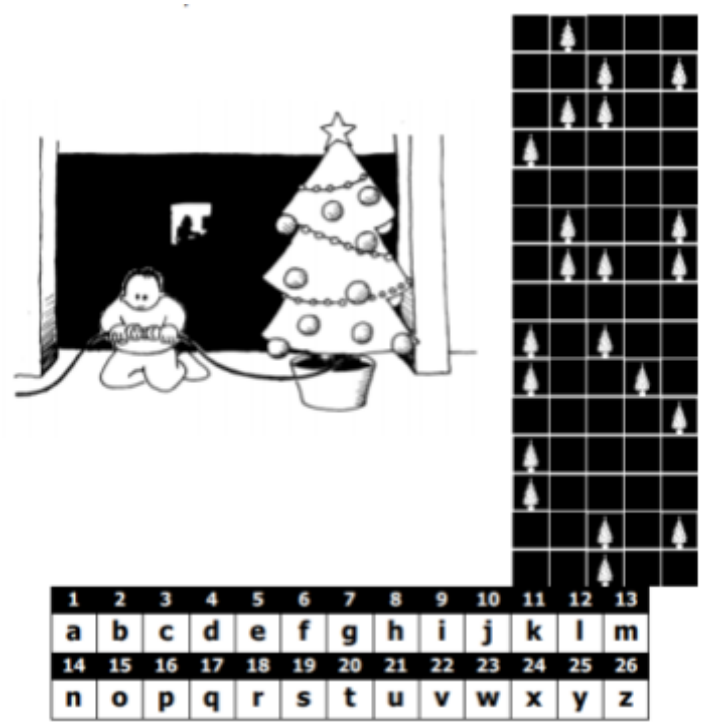

Figure ***1.5 Image of John in his tree house trying to send a message using the christmas tree***

E-mail and Modems

Computers connected to the internet through a modem also use the binary system to send messages. The only difference is that they use beeps. A high-pitched beep can be used for a one and a low-pitched beep for a zero. These tones go very fast—so fast, in fact, that all we can hear is a horrible continuous screeching sound. If you have never heard it, listen to a modem connecting to the Internet, or try calling a fax machine—fax machines also use modems to send information.

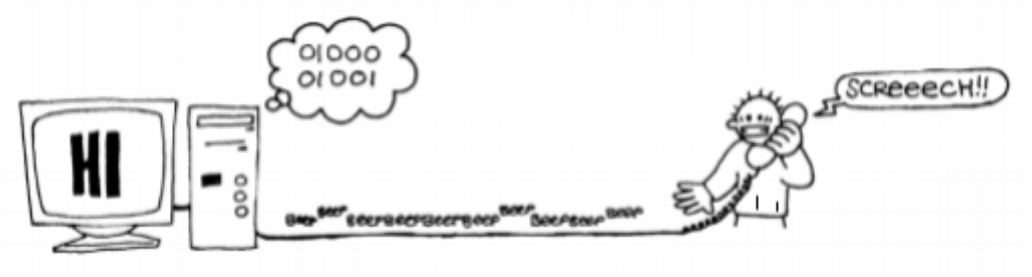

Figure *1.6 Listening to a modem connection to the internet*

Using the same code that John used in the tree house, try sending an email message to your friend using the high-pitched and low-pitched beep. Make it easy for yourself and your friend though— you don't have to be as fast as a real modem!

Figure *1.7 Sending a message to your friends using the high-pitched and low-pitched beep*

Each of the cards we have used so far represents a ‘bit’ on the computer (‘bit’ is short for ‘binary digit’). So our alphabet code we have used so far can be represented using just five cards, or ‘bits’. However, a computer has to know whether letters are capitals or not, and also recognise digits, punctuation and special symbols such as $ or ~.

Go and look at a keyboard and work out how many characters a computer has to represent. So how many bits does a computer need to store all the characters?

Most computers today use a representation called ASCII (American Standard Code for Information Interchange), which is based on using this number of bits per character, but some non-English speaking countries have to use longer codes.

Figure *1.8 How the computer feels after hearing ASCII*

Image Representation Using Numbers In A Computer

Learning Objectives:

- This lesson aims to introduce students to how images are represented in a computer
- Students get to learn how a computer stores and transmits data

---

Introduction

Computers store drawings, photographs and other pictures using only numbers. A typical example of a device that makes this happen is the Fax Machine.

Fax (short for facsimile), is the transmission of a scanned printed material (in both text and images), to a telephone number connected to a printer or other types of output device. The original document is scanned with a fax machine (or a telecopier), which converts the contents (text or images) into a bitmap, Then the bitmap is sent through the telephone system. At the receiving end, the fax machine reconverts the coded image, printing a paper copy.

Interactive Question

One quick question that we should ask ourselves is In what situations would computers need to store pictures?

Answer: A drawing program, a game with graphics, or a multi-media system.

Interactive Question

This leads us to another question.

How can computers store pictures when they can only use numbers?

Images are displayed on our computer screen for all sorts of reasons. Just like images, videos are just still images shown quickly after one another. Surprised? You can attach and view images in your email, and you can browse ***Pj Masks*** pictures on Google's image search.

Today, we'll explore how images are represented in a computer. To keep our understanding of how images are stored in a computer simple, we'll limit our discussion to black and white images.

Computer screens are divided up into a grid of small dots called pixels (**picture elements**). In a black and white picture, each pixel is either black or white.
From the figure below, the letter "a" has been magnified to show the pixels. When a computer stores a picture, all that it needs to store is which dots are black and which are white. Take a look.

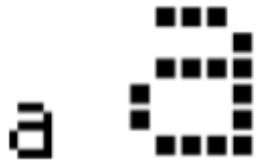

Figure *1.9 The letter 'a' magnified to show the pixels.*

The picture below shows us how a picture can be represented by numbers.

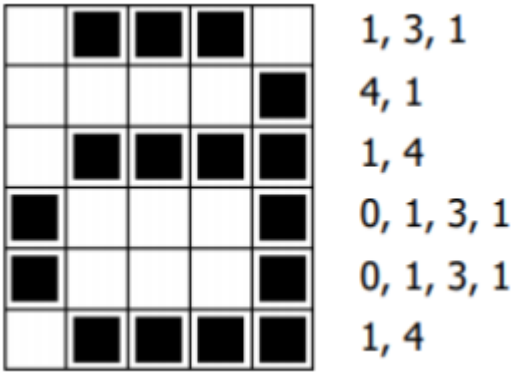

Figure *1.10 Representation of a picture on the pixel grid with numbers.*

The first line on the grid or 6x5 Matrix consists of one white pixel, then three black pixels, then one white pixel. Thus the first line is represented as 1, 3, 1. The first number always relates to the number of white pixels. If the first pixel is black the line will begin with a zero.

A fax machine is really just a simple computer that scans a black and white page into about 1000 × 2000 pixels, which are sent using a modem to another fax machine, which prints the pixels out on a page. Often, fax images have large blocks of white (e.g. margins) or black pixels (e.g. a horizontal line). Colour pictures also have a lot of repetitions in them. To save on the amount of storage space needed to keep such images programmers can use a variety of compression techniques. The method used in this activity is called 'run-length coding',

and is an effective way to compress images. If we didn't compress images it would take much longer to transmit pictures and require much more storage space. This would make it infeasible to send faxes or put photos on a web page. For example, fax images are generally compressed to about a seventh of their original size. Without compression they would take seven times as long to transmit!

Text Compression

Computers have to store and transmit a lot of data. So that they don't have to use up too much storage space, or take too long to send information through a network connection or to an external storage device.

Since computers only have a limited amount of space to hold information, they need to represent information as efficiently as possible. This is process is called compression. By coding data before it is stored, and decoding it when it is retrieved, the computer can store more data, or send it faster through the Internet or an external storage device.

Information Theory

A celebrated American mathematician (and juggler, and unicyclist) called ***Claude Shannon*** did a lot of experiments with this game. He measured the amount of information in bits—each yes/no answer is equivalent to a 1/0 bit. He found that the amount of "information" contained in a message depends on what you already know. Sometimes we can ask a question that eliminates the need to ask a lot of questions. In this case the information content of the message is low.

For example, the information in a single toss of a coin is normally one bit: heads or tails. But if the coin happens to be a biased one that turns up heads nine times out of ten, then the information is no longer one bit—believe it or not, it's less. How can you find out what a coin toss was with less than one yes/no question? Simple—just use questions like "are the next two coin tosses both heads?" For a sequence of tosses with the biased coin, the answer to this will be "yes" about 80%, of the time. On the 20% of occasions where the answer is "no," you will have to ask two further questions. But on average you will be asking less than one question per coin toss! ***Shannon*** called the information content of a message "entropy". Entropy depends not only on the number of possible outcomes—in the case of a coin toss, two—but also on the probability of it happening. Improbable events, or surprising information, need a lot more questions to guess the message because they tell us more information we didn't already know—just like the situation of taking a helicopter to school.

The entropy of a message is very important to computer scientists. You cannot compress a message to occupy less space than its entropy, and the best compression systems are equivalent to a guessing game. Since a computer program is making the 'guesses', the list of questions can be reproduced later, so as long as the answers (bits) are stored, we can reconstruct the information! The best compression systems can reduce text files to about a quarter of their original size—a big saving on storage space! The guessing method can also be used to build a computer interface that predicts what the user is going to type next! This can be very useful for physically disabled people who find it difficult to type. The computer suggests what it thinks they are likely to type next, and they just indicate what they want. A good system needs an average of only two yes/no answers per character, and can be of great assistance to someone who has difficulty making the fine movements needed to control a mouse or keyboard. This sort of system is also used in a different form to 'type' text on some cellphones.

CHAPTER 2

Algorithms

Learning Objectives:

- This lesson aims to introduce students to how the Searching Algorithms works in a computer
- Students will also get to learn three different search methods: linear searching, binary searching and hashing through an activity
- Students will also learn Searching and Sorting Algorithms, Sorting Networks

---

Introduction

Computers operate by following a list of instructions set for them. These instructions enable them to sort, find send or even manipulate information. To do these things as quickly as possible, you need good methods for finding things in large collections of data, and for sending information through networks.

An algorithm is a set of instructions for completing a task. The idea of an algorithm is central to computer science. Algorithms are how we get computers to solve problems. Some algorithms are faster than others, and many of the algorithms that have been discovered have made it possible to solve problems that previously took an infeasible length of time—for example, finding all the pages that contain your name on the World-Wide Web, or finding out the best way to pack parcels into a container, or finding out whether or not very large (100-digit) numbers are prime.

The word “algorithm” is derived from the name of Mohammed ibn Musa AlKhowarizmi—Mohammed, son of Moses, from Khowarizm—who joined an academic centre known as the House of Wisdom in Baghdad around 800AD. His works transmitted the Hindu art of reckoning to the Arabs, and thence to Europe. When they were translated into Latin in 1120AD, the first words were “Dixit Algorismi”—“thus said Algorismi”.

Battleships

Introductory Activity

1. Choose about 15 students to line up at the front of the classroom. Give each student a card with a number on it (in random order). Keep the numbers hidden from the rest of the class.

2. Give another student a container with four or five sweets in it. Their job is to find a given number. They can “pay” to look at a particular card. If they find the correct number before using all their sweets, they get to keep the rest.

3. Repeat if you wish to.

4. Now shuffle the cards and give them out again. This time, have the students sort themselves into ascending order. The searching process is repeated.

If the numbers are sorted, a sensible strategy is to use just one “payment” to eliminate half the students by having the middle student reveal their card. By repeating this process they should be able to find the number using only three sweets. The increased efficiency will be obvious. Activity The students can get a feel for how a computer searches by playing the battleship game. As they play the game, get them to think about the strategies they are using to locate the ships.

Activity

Computers are often required to find information in large collections of data. They need to develop quick and efficient ways of doing this. This activity demonstrates three different search methods: linear searching, binary searching and hashing. The students can get a feel for how a computer searches by playing the battleship game. As they play the game, get them to think about the strategies they are using to locate the ships.

Battleships—A Linear Searching Game

Read the following instructions to the students

1. Organise yourselves into pairs. One of you has sheet 1A, the other sheet 1B. Don’t show your sheet to your partner!

2. Both of you circle one battleship on the top line of your game sheet and tell your partner its number.

3. Now take turns to guess where your partner’s ship is. (You say the letter name of a ship and your partner tells you the number of the ship at that letter.)

4. How many shots does it take to locate your partner’s ship? This is your score for the game. (Sheets 1A' and 1B' are extras provided for students who would like to play more games or who “inadvertently” see their partner’s sheet. Sheets 2A', 2B' and 3A', 3B' are for the later games.)

Follow Up Discussion

1. What were the scores?

2. What would be the minimum and maximum scores possible? (They are 1 and 26 respectively, assuming that the students don’t shoot at the same ship twice. This method is called ‘linear search’, because it involves going through all the positions, one by one.)

Battleships—A Binary Searching Game

Instructions

The instructions for this version of the game are the same as for the previous game but the numbers on the ships are now in ascending order. Explain this to the students before they start.

1. Organise yourselves into pairs. One of you has sheet 2A, the other sheet 2B. Don't show your sheet to your partner!

2. Both of you circle one battleship on the top line of your game sheet and tell your partner its number.

3. Now take turns to guess where your partner's ship is. (You say the letter name of a ship and your partner tells you the number of the ship at that letter.)

4. How many shots does it take to locate your partner's ship? This is your score for the game.

Follow Up Discussion

1. What were the scores?

2. What strategy did the low scorers use?

3. Which ship should you choose first? (The one in the middle tells you which half of the line the chosen ship must be in.) Which location would you choose next? (Again the best strategy is always to choose the middle ship of the section that must have the selected ship.)

4. If this strategy is applied how many shots will it take to find a ship? (Five at most). This method is called 'binary search', because it divides the problem into two parts.

Battleships—A Hashing Search Game

Instructions

1. Each take a sheet as in the previous games and tell your partner the number of your chosen ship.

2. In this game you can find out which column (0 to 9) the ship is in. You simply add together the digits of the ship's number. The last digit of the sum is the column the ship is in. For example, to locate a ship numbered 2345, add the digits 2+3+4+5, giving 14. The last digit of the sum is 4, so that ship must be in column 4. Once you know the column you need to guess which of the ships in that column is the desired one. This technique is called 'hashing', because the digits are being squashed up ("hashed") together.

3. Now play the game using this new searching strategy. You may like to play more than one game using the same sheet—just choose from different columns.

(Note that, unlike the other games, the spare sheets 3A' and 3B' must be used as a pair, because the pattern of ships in columns must correspond.)

Follow Up Discussion

1. Collect and discuss scores as before.

2. Which ships are very quick to find? (The ones that are alone in their column.) Which ships may be harder to find? (The ones whose columns contain lots of other ships.)

3. Which of the three searching processes is fastest? Why?

What are the advantages of each of the three different ways of searching? (The second strategy is faster than the first, but the first one doesn't require the ships to be sorted into order. The third strategy is usually faster than the other two, but it is possible, by chance, for

it to be very slow. In the worst case, if all the ships end up in the same column, it is just as slow as the first strategy.)

Summary

Computers store a lot of information, and they need to be able to sift through it quickly. One of the biggest search problems in the world is faced by Internet search engines, which must search billions of web pages in a fraction of a second. The data that a computer is asked to look up, such as a word, a bar code number or an author's name, is called a *search key*.

Computers can process information very quickly, and you might think that to find something they should just start at the beginning of their storage and keep looking until the desired information is found. This is what we did in the Linear Searching Game. But this method is very slow—even for computers. For example, suppose a supermarket has 10,000 different products on its shelves. When a bar code is scanned at a checkout, the computer must look through up to 10,000 numbers to find the product name and price. Even if it takes only one thousandth of a second to check each code, ten seconds would be needed to go through the whole list. Imagine how long it would take to check out the groceries for a family!

A better strategy is *binary search*. In this method, the numbers are sorted into order. Checking the middle item of the list will identify which half the search key is in. The process is repeated until the item is found. Returning to the supermarket example, the 10,000 items can now be searched with fourteen probes, which might take two hundredths of a second—hardly noticeable.

A third strategy for finding data is called *hashing*. Here the search key is manipulated to indicate exactly where to find the information. For example, if the search key is a telephone number, you could add up all the digits in the number and take the remainder when divided by 11. In this respect, a hash key is a little like the check digits discussed in Activity 4—a small piece of data whose value depends on the other data being processed. Usually the computer will find what it is looking for straight away. There is a small chance that several keys end up in the same location in which case the computer will need to search through them until it finds the one it is seeking.

Computer programmers usually use some version of the hashing strategy for searching, unless it is important to keep the data in order, or unless an occasional slow response is unacceptable.

Lightest and Heaviest—Sorting Algorithms

Discussion

Computers often have to sort lists of things into order. Brainstorm all the places where putting things into order is important. What would happen if these things were not in order?

Computers usually only compare two values at once.

Sorting Weights

Aim: To find the best method of sorting a group of unknown weights into order.

You will need: Sand or water, 8 identical containers, a set of balance scales

What to do:

1. Fill each container with a different amount of sand or water. Seal tightly.

2. Mix them up so that you no longer know the order of the weights.

3. Find the lightest weight. What is the easiest way of doing this?

Note: You are only allowed to use the scales to find out how heavy each container is. Only two weights can be compared at a time.

4. Choose 3 weights at random and sort them into order from lightest to heaviest using only the scales. How did you do this? What is the minimum number of comparisons you can make? Why?

5. Now sort all of the objects into order from lightest to heaviest. When you think you have finished, check your ordering by re-weighing each pair of objects standing together.

Selection Sort

One method a computer might use is called *selection sort*. This is how selection sort works. First find the lightest weight in the set and put it to one side. Next, find the lightest of the weights that are left, and remove it. Repeat this until all the weights have been removed.

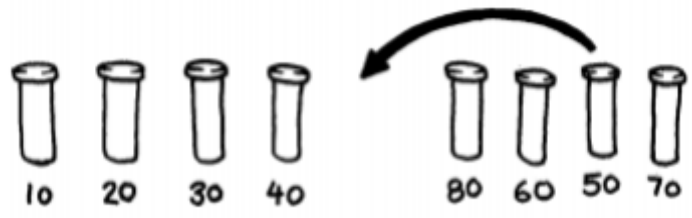

Figure *2.0 Selection Sort as performed by a computer.*

Count how many comparisons you made.

Quicksort

Quicksort is a lot faster than *selection sort*, particularly for larger lists. In fact, it is one of the best methods known. This is how quicksort works.

Choose one of the objects at random, and place it on one side of the balance scales.

Now compare each of the remaining objects with it. Put those that are lighter on the left, the chosen object in the middle, and the heavier ones on the right. (By chance you may end up with many more objects on one side than on the other.)

Choose one of the groups and repeat this procedure. Do the same for the other group. Remember to keep the one you know in the centre.

Keep repeating this procedure on the remaining groups until no group has more than one object in it. Once all the groups have been divided down to single objects, the objects will be in order from lightest to heaviest.

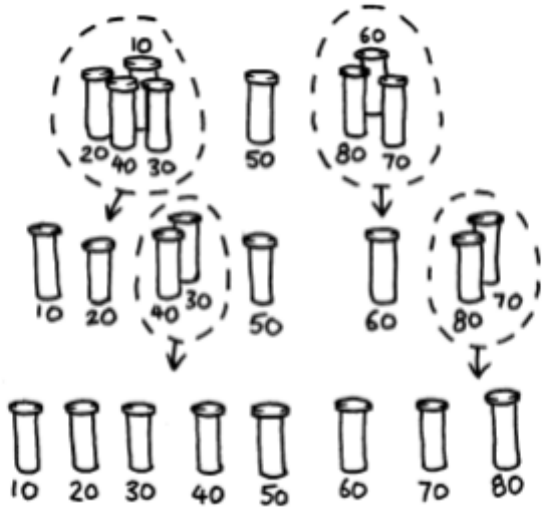

Figure *2.1 Quicksort activity.*

How many comparisons did this process take?

You should find that quicksort is a more efficient method than selection sort unless you happen to have chosen the lightest or heaviest weight to begin with. If you were lucky enough to have chosen the middle weight, you should have taken only 14 comparisons, compared with the 28 for selection sort. At any rate the quicksort method will never be any worse than selection sort and may be much better!

Variations and extensions

Many different methods for sorting have been invented. You could try sorting your weights using these:

Insertion sort works by removing each object from an unsorted group and inserting it into its correct position in a growing list (see figure below). With each insertion the group of unsorted objects shrinks and the sorted list grows, until eventually the whole list is sorted. Card players often use this method to sort a hand into order.

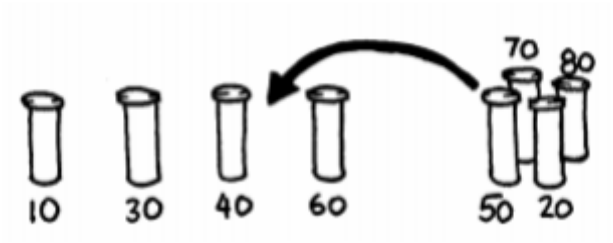

Figure *2.1 Insertion Sort*

Bubble sort involves going through the list again and again, swapping any objects side-by-side that are in the wrong order. The list is sorted when no swaps occur during a pass through the list. This method is not very efficient, but some people find it easier to understand than the others.

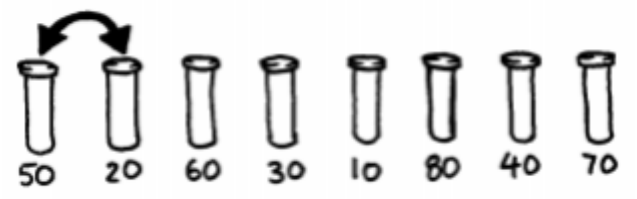

Figure *2.2 Bubble Sort*

Mergesort is another method that uses 'divide and conquer' to sort a list of items. First, the list is divided at random into two lists of equal size (or nearly equal if there are an odd number of items). Each of the two half-size lists is sorted, and the two lists are merged together. Merging two sorted lists is easy—you repeatedly remove the smaller of the two items at the front of the two lists. In the figure below, the 40 and 60-gram weights are at the front of the lists, so the next item to add is the 40-gram weight. How do you sort the smaller lists?

Simple—just use mergesort! Eventually, all the lists will be cut down into individual items, so you don't need to worry about knowing when to stop.

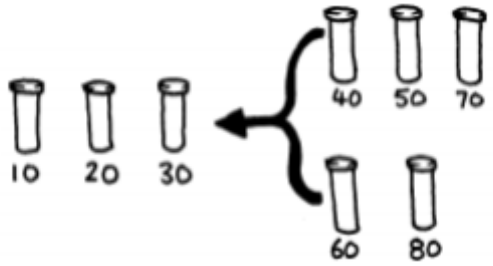

Figure *2.2 MergeSort*

Summary

Information is much easier to find in a sorted list. Telephone directories, dictionaries and book indexes all use alphabetical order, and life would be far more difficult if they didn't. If a list of numbers (such as a list of expenses) is sorted into order, the extreme cases are easy to see because they are at the beginning and end of the list. Duplicates are also easy to find, because they end up together.

Computers spend a lot of their time sorting things into order, so computer scientists have to find fast and efficient ways of doing this. Some of the slower methods such as insertion sort, selection sort and bubble sort can be useful in special situations, but the fast ones such as quicksort and mergesort are usually used because they are much faster on large lists – for example, for 100,000 items, quicksort is typically about 2,000 times as fast as selection sort, and for 1,000,000 items, it is about 20,000 times as fast. Computers often have to deal with

a million items (lots of websites have millions of customers, and even a single photo taken on a cheap camera has over a million pixels); the difference between the two algorithms is the difference between taking 1 second to process the items, and taking over 5 hours to do exactly the same task. Not only would the delay be intolerable, but it will have used 20,000 times as much power (which not only impacts the environment, but also reduces battery life in portable devices), so choosing the right algorithm has serious consequences.

Quicksort uses an approach called Divide and Conquer. In quicksort, you keep dividing a list into smaller parts, and then perform a quicksort on each of the parts. The list is divided repeatedly until it is small enough to conquer. For quicksort, the lists are divided until they contain only one item. It is trivial to sort one item into order! Although this seems very involved, in practice it is dramatically faster than other methods. This is an example of a powerful idea called Recursion where an algorithm uses itself to solve a problem – this sounds weird but it can work very well.

CHAPTER 3

Telling Computers What To Do

Learning Objectives:

- This lesson aims to make students understand how to give sets of instructions to a computer
- Students will also be exposed to two activities to give us some idea of what it is like to communicate with a computer.

Computers follow instructions—millions of instructions every second. To tell a computer what to do, all you have to do is give it the right instructions. But that's not as easy as it sounds!

When we are given instructions we use common sense to interpret what is meant. If someone says "go through that door," they don't mean to actually smash through the door—they mean go through the doorway, if necessary opening the door first! Computers are different. Indeed, when they are attached to mobile robots you need to be careful to take safety precautions to avoid them causing damage and danger by interpreting instructions literally—like trying to go through doors. Dealing with something that obeys instructions exactly, without "thinking", takes some getting used to.

The two activities in this chapter give us some idea of what it is like to communicate to literal-minded machines using a fixed set of instructions.

The first will teach us about a "machine" that computers use to recognise words, numbers or strings of symbols that the computer can work with. These "machines" are called *finite-state automata*.

The second activity introduces us to how we can communicate with computers. A good programmer has to learn how to tell the computer what to do using a fixed set of instructions that are interpreted literally. The list of instructions is the program. There are lots of different programming languages a programmer can choose to write these instructions in, but we will be using a simple language that can be used without a computer.

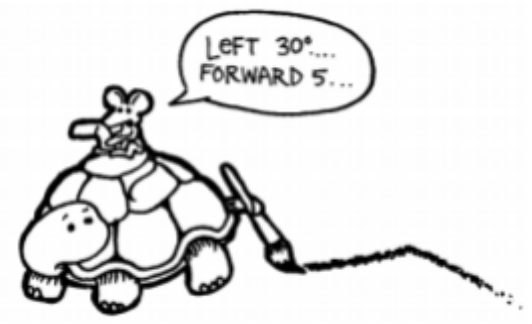

Figure *3.0 Giving fixed set of instructions*

Treasure Hunt—Finite-State Automata

Computer programs often need to process a sequence of symbols such as letters or words in a document, or even the text of another computer program. Computer scientists often use a finite-state automaton to do this. A finite-state automaton (FSA) follows a set of instructions to see if the computer will recognise the word or string of symbols. We will be working with something equivalent to a FSA—treasure maps!

Treasure Island

Introduction

Your goal is to find Treasure Island. Friendly pirate ships sail along a fixed set of routes between the islands in this part of the world, offering rides to travellers. Each island has two departing ships, A and B, which you can choose to travel on. You need to find the best route to Treasure Island. At each island you arrive at you may ask for either ship A or B (not both). The person at the island will tell you where your ship will take you to next, but the pirates don't have a map of all the islands available. Use your map to keep track of where you are going and which ship you have travelled on.

Demonstration

(Note: This is a different map from the actual activity.) Using a board, draw a diagram of three islands as shown here:

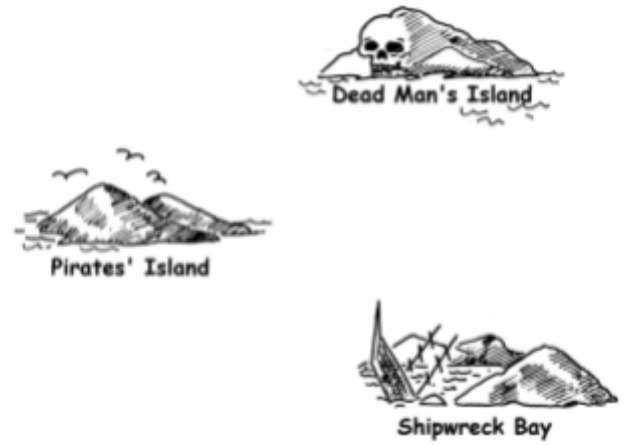

Figure *3.1 Diagram of three islands*

Copy the three cards on the next two pages, and have one student hold each card. Note that the routes on these cards are different from those in the main activity. Starting at Pirates' Island ask for ship A. The student should direct you to Shipwreck Bay. Mark the route in on the map. At Shipwreck Bay ask for ship A again. You will be directed back to Pirates' island. Mark this on the map. This time ask for ship B. Mark this on the map. This route goes to Dead Man's Island, at which stage you will be stuck!

Your final map should look like this:

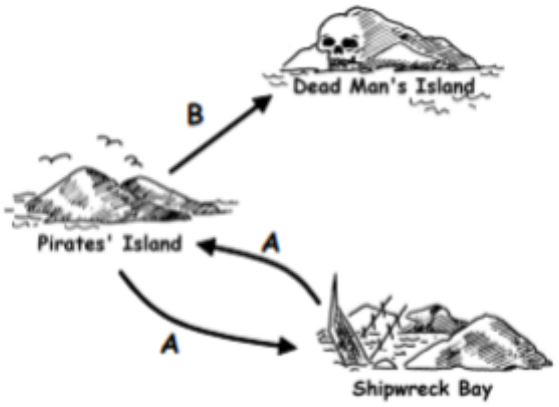

Figure *3.2 Final Map*

What's it all about?

Finite-state automata are used in computer science to help a computer process a sequence of characters or events.

A simple example is when you dial up a telephone number and you get a message that says "Press 1 for this … Press 2 for that … Press 3 to talk to a human operator." Your key presses are inputs for a finite state automaton at the other end of the phone. The dialogue can be quite simple, or very complex. Sometimes you are taken round in circles because there is a peculiar loop in the finite-state automaton. If this occurs, it is an error in the design of the system— and it can be extremely frustrating for the caller!

Another example is when you get cash from a bank cash machine. The program in the machine's computer leads you through a sequence of events. Inside the program all the possible sequences are kept as a finite-state automaton. Every key you press takes the automaton to another state. Some of the states have instructions for the computer on them, like "dispense ₦1000 of cash" or "print a statement" or "eject the cash card".

Some computer programs really do deal with English sentences using maps. They can both generate sentences themselves, and process sentences that the user types in. In the 1960s a computer scientist wrote a famous program called "Eliza" (after Eliza Dolittle) that had conversations with people. The program pretended to be a psychotherapist, and came out with leading questions like "Tell me about your family" and "Do go on." Although it didn't "understand" anything, it was sufficiently plausible—and its human users were sufficiently gullible—that some people really did think they were talking to a human psychotherapist.

Summary

Although computers are not really very good at understanding natural language, they can readily process artificial languages. One important type of artificial language is the programming language. Computers use finite-state automata to read in programs and translate them into the form of elementary computer instructions, which can then be "executed" directly by the computer.