2 CREATING ALGORITHMS

In this section you will learn how to create algorithms to solve particular problems using the constructs of sequence, selection and iteration. You will also practise displaying algorithms in flowcharts and pseudocode.

LEARNING OBJECTIVES

- Understand how to create an algorithm to solve a particular problem
- Make use of programming constructs (sequence, selection and iteration) and use appropriate conventions (flowchart, pseudocode, written description, draft program code)

ALGORITHMS FOR COMPUTERS

SUBJECT VOCABULARY

construct a smaller part from which something is built. Letters and numbers (i.e. a to z and 0 to 9) are the constructs we use to build our language and convey meaning. Bricks and cement are the basic constructs of a building

selection a construct that allows a choice to be made between different alternatives

iteration a construct that means a process is repeated. An action is repeated until a **condition** is met or a particular outcome is reached. It is often referred to as a 'loop'

GENERAL VOCABULARY

ambiguous when a statement or command does not have one obvious meaning but can be interpreted in different ways

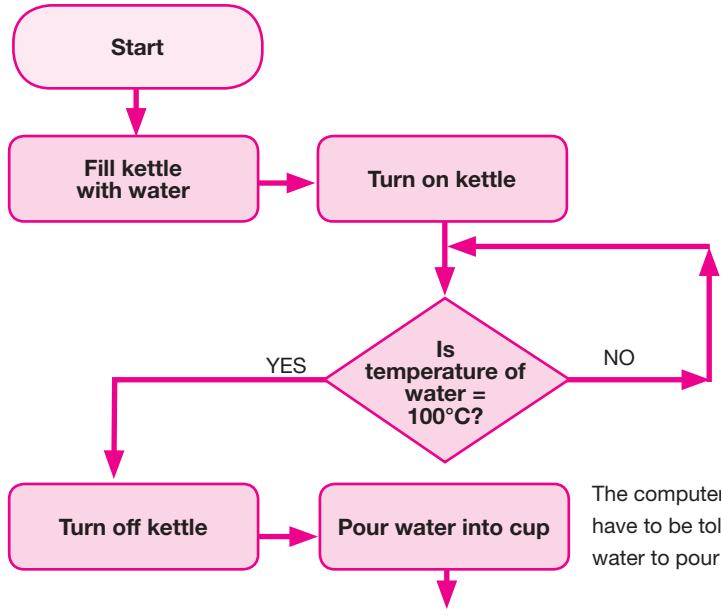
condition something that must happen before something else can happen

There was an **ambiguous** statement in the algorithm for making a cup of coffee. After filling the kettle with water and adding coffee to the cup, the next instruction was 'Wait for water to boil'.

A human can understand that this instruction means they have to keep checking the kettle over and over again until the water is boiling, but a computer is unable to understand an instruction like this in the same way. It would just wait. And wait. Forever.

Even worse, the algorithm didn't state clearly how to tell the water was boiling. Through experience, we humans assume the water is boiling when there is lots of steam, sound and bubbles; or, even better, when the kettle turns itself off. An algorithm for a computer would have to state that it must wait until the water reached 100°C.

A version of this part of the algorithm, suitable for a computer, is shown in Figure 1.6. This example introduces two new **constructs** from which algorithms are created. We have already met the construct sequence – step-by-step instructions in the correct order. To add to this, we now have **selection** and **iteration**.

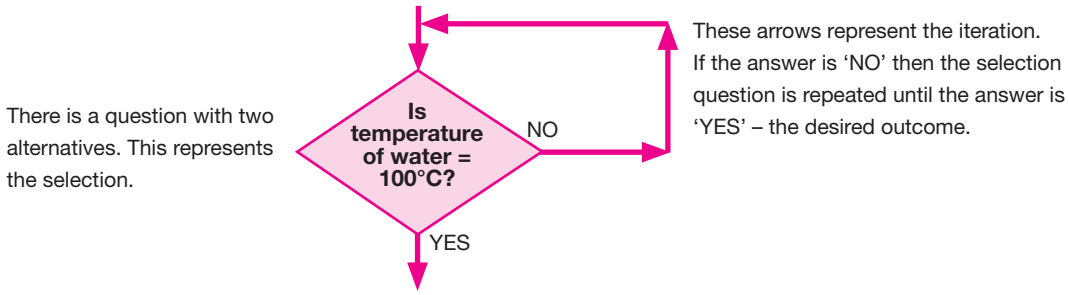


The computer would also have to be told how much water to pour into the cup!

▶ **Figure 1.6** Part of an algorithm suitable for a computer for making coffee

REPRESENTING SELECTION AND ITERATION IN A FLOWCHART

Selection and iteration are represented in a flowchart as shown in Figure 1.7.



▶ **Figure 1.7** Selection and iteration in a flowchart

SKILLS REASONING, PROBLEM SOLVING

DID YOU KNOW?

We use iteration in our daily lives whenever we carry out an action over and over again. For example, at mealtimes we keep on eating until our plate is empty or we have had enough to eat.

When we're travelling by car and the traffic lights are red, we have to keep waiting until they change to green.

An actor repeats their lines over and over again, learning more each time until they know them all.

ACTIVITY 7

GUESSING GAMES

A student is creating a guessing game. A player has to enter a number no greater than 10. If it is too high, they are informed that they have made an error. But if it is within the range 1 to 10, they are told whether or not they have guessed the correct number. (Assume that the correct number is 3.)

Can you make an algorithm to solve this problem and express it as a written description and a flowchart?

Compare your solution to others in your group.

Check that they are correct and would produce the correct outcome.

Are some of the algorithms more efficient than others? Do they use fewer commands?

SKILLS PROBLEM SOLVING, ANALYSIS

ACTIVITY 8

CALCULATING GRADES

A school uses this algorithm to calculate the grade that students achieve in end-of-topic tests.

```
RECEIVE testScore FROM KEYBOARD
IF testScore >= 80 THEN
    SEND 'A' TO DISPLAY
ELSE
    IF testScore >= 70 THEN
        SEND 'B' TO DISPLAY
    ELSE
        IF testScore >= 60 THEN
            SEND 'C' TO DISPLAY
        ELSE
            IF testScore > 0 THEN
                SEND 'D' TO DISPLAY
            ELSE
                SEND 'FAIL' TO DISPLAY
```

```
            END IF
        END IF
    END IF
END IF
```

What would be the output of this algorithm for these test scores: 91, 56 and 78?

ITERATION

When writing programs, it is often necessary to repeat the same set of statements several times. Instead of making multiple copies of the statements, you can use iteration to repeat them. The algorithm for making a cup of coffee includes an instruction to keep waiting until the water in the kettle boils.

CHECKPOINT

Strengthen

SKILLS CRITICAL THINKING, DECISION MAKING

S1 How are sequence, selection and iteration used in algorithms? Give examples to justify your answer.

Challenge

SKILLS PROBLEM SOLVING

C1 Develop an algorithm using a flowchart that asks the user to enter their height (in metres) and weight (in kilograms) and displays their body mass index (BMI). The formula for calculating BMI is weight/height^2.

SKILLS PROBLEM SOLVING

C2 Develop an algorithm expressed as a flowchart to control the heating in a house. A thermostat monitors the temperature within the house. During the week the temperature should be 20°C between 06.00 and 08.30 in the morning and between 17.30 and 22.00 at night. At weekends it should be 22°C between 08.00 and 23.00. If the temperature in the house falls below 10°C at any time the boiler is switched on.

How confident do you feel about your answers to these questions? If you're not sure you answered them well, try the following activities again.

- For S1 have a look at the Subject vocabulary sections in 'Understanding algorithms' and 'Creating algorithms'.

SUMMARY

- The constructs sequence, selection and iteration are the basic building blocks of algorithms.

3 SORTING AND SEARCHING ALGORITHMS

Whenever we want to find information, we carry out a search. Just imagine the number of people around the world who are using the same search engine at the same time. Without efficient searching algorithms, we would have to wait a long time to be shown the results.

We are on all sorts of lists (for example, school and college, clubs and groups, voting registers) and all these must be searched to find relevant information. Sorting information is also important to facilitate efficient searches.

LEARNING OBJECTIVES

- Understand how standard algorithms work (bubble sort, merge sort, linear search, binary search)
- Understand how the choice of algorithm is influenced by the data structures and data values that need to be manipulated
- Evaluate the fitness for purpose of algorithms in meeting specified requirements efficiently, using logical reasoning and test data

Two of the most common tasks in computer programs are sorting data into a particular order and searching for particular items of information.

There might be millions of items of stored data and searching for information wouldn't be efficient if the data was not sorted. Imagine the confusion and difficulty of having to find something in a dictionary that wasn't in alphabetical order. Or planning a trip with train timetables that weren't sorted into time order. Even small lists such as football league tables or the Top 20 music charts are much more useful if they are sorted into order.

SORTING ALGORITHMS

As sorting is such a widely used procedure, many algorithms have been created to carry it out. As with all algorithms, some are more efficient than others.

SUBJECT VOCABULARY

ascending order this is arranging items from smallest to largest (e.g. 1, 2, 3)

descending order this is arranging items from largest to smallest (e.g. 3, 2, 1)

traversal travel across or through something.

BUBBLE SORT

When data is sorted, different items must be compared with each other and moved so that they are in either **ascending order** or **descending order**.

The bubble sort algorithm starts at one end of the list and compares pairs of data items. If they are in the wrong order, they are swapped. The comparison of pairs continues to the end of the list, each complete **traversal** of the list being called a 'pass'. This process is repeated until there have been no swaps during a pass. This indicates that the items must all be in the correct order.

The algorithm can be described as follows.

BUBBLE SORT (ASCENDING ORDER)

1. Start at the beginning of the list.
2. Compare the values in position 1 and position 2 in the list – if they are not in ascending order then swap them.
3. Compare the values in position 2 and position 3 in the list and swap if necessary.
4. Continue to the end of the list.
5. If there have been any swaps, repeat steps 1 to 4.

WORKED EXAMPLE

Here is an example of a bubble sort in action.

Pass 1

(4)	(2)	6	1	3	Items 1 and 2 must be swapped.
2	4	6	1	3	Items 1 and 2 are swapped.
2	(4)	(6)	1	3	Items 2 and 3 are already in ascending order.
2	4	(6)	(1)	3	Items 3 and 4 must be swapped.
2	4	1	6	3	Items 3 and 4 have been swapped.
2	4	1	(6)	(3)	Items 4 and 5 must now be swapped.
2	4	1	3	6	Items 4 and 5 have been swapped.

Pass 2

(2)	(4)	1	3	6	Items 1 and 2 are in correct order.
2	(4)	(1)	3	6	Items 2 and 3 must be swapped.
2	1	4	3	6	Items 2 and 3 have been swapped.
2	1	(4)	(3)	6	Items 3 and 4 must be swapped.
2	1	3	4	6	Items 3 and 4 have been swapped.
2	1	3	(4)	(6)	Items 4 and 5 do not need to be swapped.

Pass 3

(2)	(1)	3	4	6	Items 1 and 2 must be swapped.
1	2	3	4	6	Items 1 and 2 have been swapped.
1	2	3	4	6	All items are now in the correct order.

▲ **Figure 1.8** A bubble sort

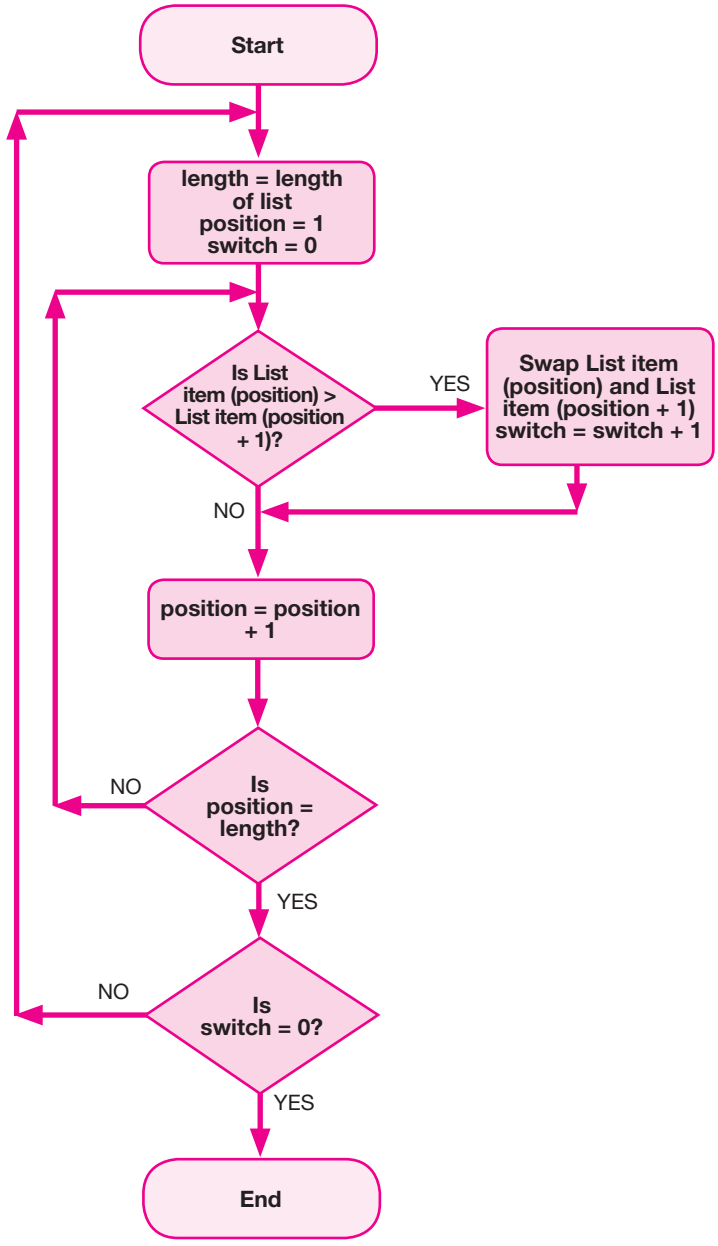
DID YOU KNOW?

Do you know why it is called 'bubble sort'? If you look carefully, you can see that the largest items gradually move to the end, like bubbles rising in water. After the first pass, the largest number is in its correct position. Then after the second pass, the next largest is in its correct position. This happens on each pass and so if the algorithm is to be made more efficient the last set of comparisons can be left out.

It would take a human three passes to carry out this bubble sort. A computer would need four passes because it must continue until there have been no swaps; it cannot just look at all of the numbers at once and see that they are all in order.

The bubble sort algorithm can be represented as a flowchart as shown in Figure 1.9.

Figure 1.9 Bubble sort algorithm written as a flowchart



SKILLS CRITICAL THINKING

ACTIVITY 9

HOW DOES BUBBLE SORT WORK?

Study the flowchart of the bubble sort algorithm.

Using the variables declared, can you explain the logic behind the algorithm? How does it function to sort a list?

GENERAL VOCABULARY

repeatedly many times

SUBJECT VOCABULARY

recursion a process that is repeated. For example, a document can be checked and edited, checked and edited and so on until it is perfect

MERGE SORT

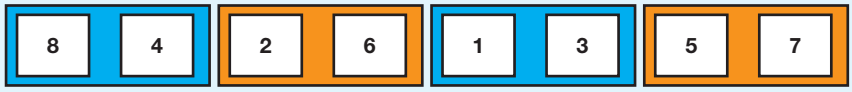
Merge sort is a sorting algorithm that divides a list into two smaller lists and then divides these until the size of each list is one. **Repeatedly** applying a method to the results of a previous application of the method is called **recursion**.

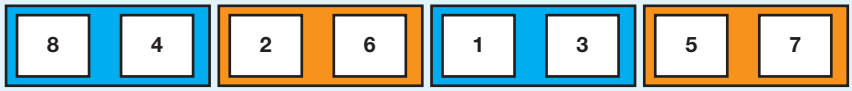
In computing, a problem is solved by repeatedly solving smaller parts of the problem. A part of a program can be run and rerun on the results of the previous run (e.g. repeatedly dividing a number by 2).

WORKED EXAMPLE

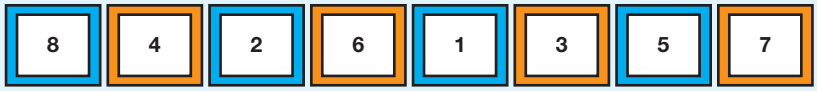
Here is an example of a merge sort which will sort the following list into ascending order.

The list is split into half with recursion to produce a left list and a right list each time.

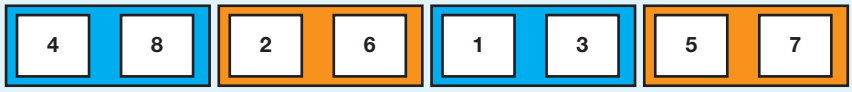




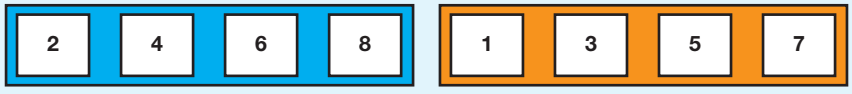
This continues until there is only one item in each list. Therefore, each list is sorted into order.



The left and right lists are now merged through recursion with the items in the correct order.



The leftmost items in each list are the lowest items of those lists and the algorithm compares them – in this case 4 with 2. The 2 is inserted in the new list and the 4 is then compared with the second number of the right list – 6. The 4 is inserted and the 6 is compared with the second number of the left list.



The algorithm now merges these two lists in the same way to produce the final sorted list. 1 is compared with 2 and then 2 with 3, 3 with 4, etc.

KEY POINT

In the exam, you will be expected to show the intermediate stages when the algorithms are applied to data.

SKILLS PROBLEM SOLVING

ACTIVITY 10

USING MERGE SORT

Using a table like the one in the worked example on page 18, show how the following list would be sorted into descending order using merge sort.

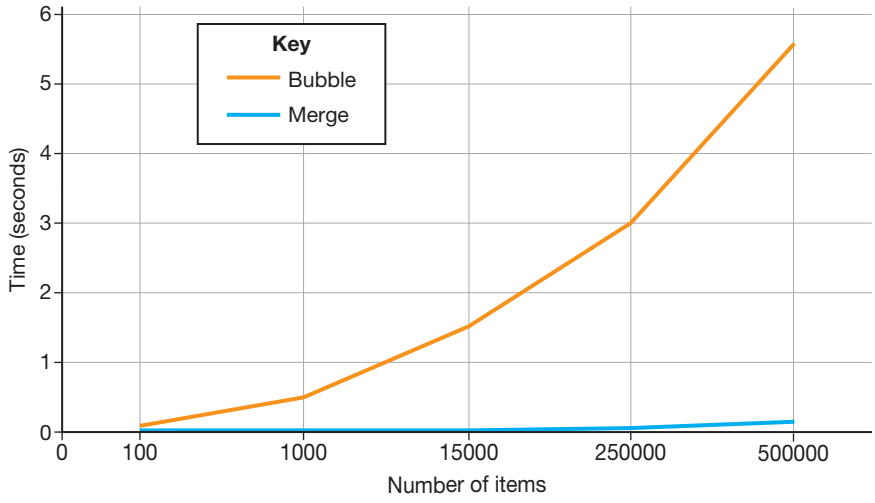
48, 20, 9, 17, 13, 21, 28, 60

EXTEND YOUR KNOWLEDGE

Only two sorting algorithms are required for the specification: bubble sort (the slowest) and merge sort (one of the most efficient). There are far more, and many of them are relatively easy to code. Research the insertion and selection sorts.

EFFICIENCY OF SORTING ALGORITHMS

This graph compares the performance of the bubble and merge sort algorithms.



▶ **Figure 1.10** A graph comparing the performance of bubble and merge sort algorithms

SUBJECT VOCABULARY

brute force an algorithm design that does not include any techniques to improve performance, but instead relies on computing power to try all possibilities until the solution to a problem is found

divide and conquer an algorithm design that works by dividing a problem into smaller and smaller sub-problems, until they are easy to solve. The solutions to these are then combined to give a solution to the complete problem

The bubble and merge sort algorithms demonstrate two alternative approaches to algorithm design.

The bubble sort algorithm is said to be using **brute force** because it starts at the beginning and completes the same task over and over again until it has found a solution.

The merge sort uses the **divide and conquer** method because it repeatedly breaks down the problem into smaller sub-problems, solves those and then combines the solutions.

The graph shows that a bubble sort is far slower at sorting lists of more than 1000 items, but for smaller lists the time difference is too small to be of importance.

As the bubble sort algorithm is easier to code, it could be beneficial to use it for smaller lists of less than 1000 items.

SEARCHING ALGORITHMS

To find a specific item in a list involves carrying out a search. Like sorting, some methods of searching are more efficient than others.

LINEAR SEARCH

A linear search is a simple algorithm and not very sophisticated. It simply starts at the beginning of the list and goes through it, item by item, until it finds the item it is looking for or reaches the end of the list without finding it.

GENERAL VOCABULARY

sequential following in order, one after the other

A linear search is **sequential** because it moves through the list item by item.

LINEAR SEARCH

1. Start at the first item in the list.
2. Compare the item with the search item.
3. If they are the same, then stop.
4. If they are not, then move to the next item.
5. Repeat 2 to 4 until the end of the list is reached.

BINARY SEARCH

Like a merge sort, a binary search uses a 'divide and conquer' method.

In a binary search the middle or **median** item in a list is repeatedly selected to reduce the size of the list to be searched – another example of recursion. If the selected item is too high or too low, then the items below or above that selected item can be searched.

To use this method, the list must be sorted into ascending or descending order. It will not work on an unsorted list.

SUBJECT VOCABULARY

median the middle number when the numbers are put in ascending or descending order (e.g. if there are 13 numbers, then the 7th number is the median). If there are an even number of items in a list, the median is the mean of the middle two numbers (e.g. if there are 10 numbers, add the 5th and 6th numbers together and divide the result by 2). In a binary search, the higher of the two numbers would be chosen

DID YOU KNOW?

You have probably used a binary search method when trying to guess a number between two limits. If you are asked to guess a number between 1 and 20 you will probably start at 10, the middle number. If you are told this is too high, you will then guess 5, the middle number between 1 and 10, and then repeat this method until you find the correct one.

BINARY SEARCH (ITEMS IN ASCENDING ORDER)

1. Select the median item of the list.
2. If the median item is equal to the search item, then stop.
3. If the median is too high, then repeat 1 and 2 with the sub-list to the left.
4. If the median is too low, then repeat 1 and 2 with the sub-list to the right.
5. Repeat steps 3 and 4 until the item has been found or all of the items have been checked.

WORKED EXAMPLE

In this list, the search item is the number 13.

3	13	24	27	(31)	39	45	60	69	SELECT THE MEDIAN NUMBER.

As this is too high, the sub-list to the left of the median must be searched.

3	13	(24)	27	THE MEDIAN NUMBER OF THIS SUB-LIST IS NOW SELECTED.

This is again too high and so the sub-list to the left must be searched.

3	(13)	THE MEDIAN NUMBER IS NOW THE SEARCH ITEM.

▲ **Figure 1.11** Binary search including sub-lists

In this example, it took three attempts to find the search item. A linear search would have accomplished this with only two attempts.

SKILLS PROBLEM SOLVING

ACTIVITY 11

USING BINARY SEARCH

Display the stages of a binary search, as in the worked example above, to find the number 13 in this list.

3 9 13 15 21 24 27 30 36 39 42 54 69

Compare your results with those of others in your group. Are all your answers the same?

EFFICIENCY OF SEARCHING ALGORITHMS

In the example on page 20, the linear search was more efficient because it only had to carry out two comparisons instead of the three for a binary search. But is this always the case?

Searching algorithms can be compared by looking at the 'worst case' and the 'best case' for each one.

WORKED EXAMPLE

If you wanted to find a particular item in a list of 1000 items, these are the best- and worst-case scenarios for the linear search and binary search algorithms.

Linear search

A linear search starts at the first item and then works through sequentially.

The best case would be if the item is first in the list.

The worst case would be if it is last in the list.

Therefore, in this example the average would be 500 comparisons.

Binary search

The best case would be if the item is in the median position in the list. The search would require only one comparison.

For the worst case it would have to choose the following medians until it finally hit the target.

(This assumes that the target is always smaller than the median.)

Attempt	Median
1	500
2	250
3	125
4	63
5	32
6	16
7	8
8	4
9	2
10	1

Therefore, the worst case for the binary search is ten comparisons.

The binary search is therefore far more efficient than the linear search.

So, should a binary search be used every time? That depends on the circumstances. The binary search has one great disadvantage. The list must be already sorted into ascending or descending order. Therefore, a sorting algorithm must be applied before the search.

If the list is to be searched just once then a linear search would be better, but if there is a large list that will be searched many times then sorting the list and using a binary search would be better. Once the list has been sorted, new items can be inserted into the correct places.

CHECKPOINT

Strengthen

SKILLS CRITICAL THINKING

S1 What are the differences between the 'bubble sort' and 'merge sort' algorithms?

SKILLS DECISION MAKING

S2 How does a binary search algorithm find the search item?

Challenge

SKILLS CRITICAL THINKING

C1 When might a linear search be preferable to a binary search, even if the binary search algorithm is more efficient?

How confident do you feel about your answers to these questions? If you're not sure you answered them well, try the following activities again.

- For S1 have a look at the 'Sorting algorithms' section.
- For S2 have a look at the 'Binary search' section.

SUMMARY

- There are many algorithms for sorting and searching data.
- The choice of algorithm depends on the data that is to be processed.
- If only a small amount of data needs to be processed, then a simpler, but less efficient search algorithm may be the best choice. The time difference of the search or sort time will be negligible.

4 DECOMPOSITION AND ABSTRACTION

When computer scientists attempt to solve problems by producing algorithms and coding them into programs, they approach the problems in a particular way. This method has been given the name 'computational thinking'. Before they create a structured solution or algorithm and code it into a program, they must define and analyse the problems. This section will introduce two of the **thought processes** they use – decomposition and abstraction.

GENERAL VOCABULARY

thought process the act of using your mind to consider or think about something

LEARNING OBJECTIVES

- Analyse a problem, investigate requirements (inputs, outputs, processing, initialisation) and design solutions
- Decompose a problem into smaller sub-problems
- Understand how abstraction can be used effectively to model aspects of the real world
- Program abstractions of real-world examples

PROBLEM SOLVING

The tasks of a computer scientist include defining and analysing problems; creating structured solutions – algorithms; and coding the solutions into a form that can be implemented by a computer. These tasks are part of what is known as **computational thinking**.

One of the skills required for computational thinking is algorithm design (which we've covered in detail in this unit). If there is a fault in the algorithm design, then the program will not work, however good a coder you are. Two other skills are **decomposition** and **abstraction**.

SUBJECT VOCABULARY

computational thinking the thought processes involved in formulating problems and their solutions so that the solutions are represented in a form that can be effectively carried out by a computer

decomposition breaking a problem down into smaller, more manageable parts, which are then easier to solve

abstraction the process of removing or hiding unnecessary detail so that only the important points remain

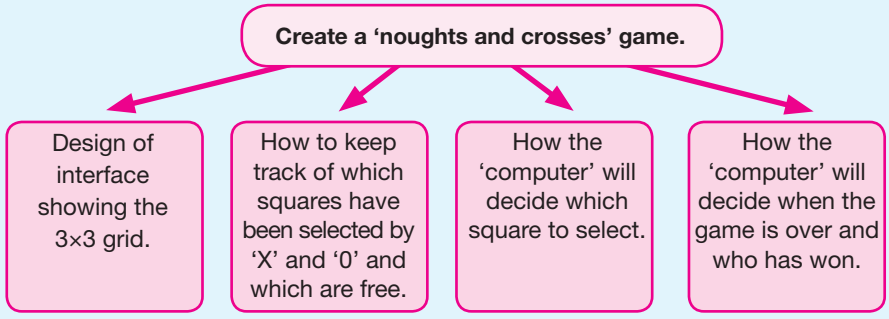
DECOMPOSITION

Decomposition is usually the first step in the problem-solving process. Once a problem has been broken down and the sub-problems have been identified, algorithms can be developed to solve each of them.

Decomposition means that sub-problems can be worked on by different teams at the same time. As smaller algorithms are developed for each sub-problem, it is easier to spot and correct errors. When the algorithm is developed into a program, code can be used again.

WORKED EXAMPLE

A student has been set the task of creating a computer version of the game 'noughts and crosses' (also known as 'tic-tac-toe') where a user plays against the computer.



▶ **Figure 1.12** Sub-problems to be solved to create a noughts and crosses computer program

The diagram shows some of the sub-problems that must be solved in order to solve the complete problem and create a version of the game.

ABSTRACTION

We use abstraction all the time in our daily lives. We **abstract** the essential features of something so that we can understand what people are trying to communicate.

GENERAL VOCABULARY

abstract to remove something from somewhere

Somebody might say, 'I was walking down the street when I saw a cat'. You immediately understand what they mean by 'street' – probably a road with a pavement and houses or shops along the side of it. Similarly, you can picture the cat – a small animal with fur, four legs and a tail. An animal that is basically 'cattish'. You have extracted the basic properties of animals called cats so that you can recognise one when you see one, or imagine one when somebody talks about a cat.

▲ Is this the street and cat you imagined?

What you imagine is very unlikely to match the actual street and cat that the person experienced. But, because of our ability to abstract, the person did not

have to go into unnecessary detail about exactly where they were and what they saw. They wouldn't get very far with the story if they did.

When we create algorithms, we abstract the basic details of the problem and represent them in a way that a computer is able to process.

WORKED EXAMPLE

Yasmin is designing a computer version of a game in which users have to throw a die to find out their number of moves.

In the computer game, the users can't have an actual die, so she will have to design a 'pretend' or virtual die that behaves in exactly the same way as a real-life die.

Yasmin will have to use her powers of abstraction to work out the essential features of a die and then represent them in computer code.

To represent the die, she will have to create a routine that will select a random number from 1 to 6 because that's what a die does.

Yasmin has used abstraction to **model** a real-life event.

GENERAL VOCABULARY

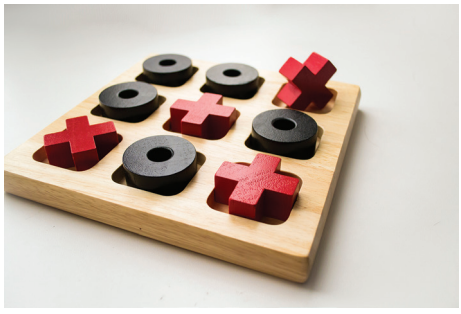
model to make a simple version of something to show how it works

LEVELS OF ABSTRACTION

There are different levels or types of abstraction. The higher the level of abstraction, the less detail is required. We use abstraction all the time in accomplishing everyday tasks.

When programmers write the 'print' command they do not have to bother about all of the details of how this will be accomplished. They are removed from them. They are at a certain level of abstraction.

A driver turning the ignition key to start a car does not have to understand how the engine works or how the spark to ignite the petrol is generated. It just happens and they can simply drive the car. That is abstraction.



▲ A game of noughts and crosses

AN EXAMPLE – NOUGHTS AND CROSSES

Figure 1.12 showed some of the sub-problems that the problem of creating a noughts and crosses game could be divided into. The following could be written at a high level of abstraction.

- The computer goes first. Then the user. This continues until either one wins, or all of the squares have been used.

Immediately a pattern can be recognised – a loop will be needed.

Inputs and outputs

The following inputs from the user will be needed.

- Start the game.
- Entries for the user.
- Select a new game or finish.

The following outputs will be needed.

- A message to inform the user when it is their turn.
- A message to inform the user if they try to select a square that has already been used.

- A message to inform the user if the game is a draw.
- A message to inform the user if they or the computer has won.
- A message to ask the user if they want to play another game or want to finish.

Processing and initialisation

The following processing will be needed.

- Set up the grid with the nine squares.
- Initialise all variables to a start value.
- Decide which square the computer will select.
- Allow the user to select a square.
- Check if the user has selected an already used square.
- Check if the computer or the user has won.
- Check if all squares have been used and the game is a draw.
- Allow the user to select a new game or finish.

The solution is still at a high level of abstraction and more details will need to be added.

For example, the programmer will need to decide how the game will record which player has selected each square; how the computer will decide which square to select; how the game will decide if the computer or the user has won.

The programmer will have to go into more and more detail or move to lower levels of abstraction.

Eventually, the programmer will be able to design an algorithm for the game and code it using a high-level programming language such as Python or Java. Even before they start to implement the game, they will need to plan how they will test the finished program to make sure that it works correctly, what test data they will use and what outcomes it should produce.

CODING AN ALGORITHM

High-level programming languages make it easier for a programmer to write code. Unfortunately, the processor that has to execute the program cannot understand the language it is written in. It therefore needs a translator to translate the code into the only language it does understand – a stream of 1s and 0s.

These high-level languages are therefore at a high level of abstraction – very far removed from the actual language of a computer.

The processing can be split into parts. For example, in the example of the noughts and crosses game there could be separate algorithms for:

- deciding where the computer should make its next selection – it could be called ‘computer entry’
- checking if the computer or the player has won – it could be called ‘check if won’
- checking if there are any empty squares left – it could be called ‘check draw’.

These separate algorithms could be used when they are needed. It is efficient because it means that the same code doesn’t have to be rewritten whenever it is needed.

SUBJECT VOCABULARY

subprogram a self-contained module of code that performs a specific task. It can be 'called' by the main program when it is needed

These items of code are called **subprograms**.

In Unit 2, we'll look in detail at how subprograms are used to reduce the complexity of programs and to make them easier to understand.

In the die example above, the designer could write a subprogram called 'die' that generates a random number from 1 to 6. In the main program the designer could just call the 'die' subprogram without having to think about how to implement it each time.

SKILLS REASONING, PROBLEM SOLVING

ACTIVITY 12

CREATING A QUIZ GAME

In a game, each player spins a wheel that is divided into four colours: red, blue, green and yellow. Each player has to answer a question on a particular topic depending on the colour next to a pointer when the wheel stops. Red is for science, blue for history, green for general knowledge and yellow for geography. A player scores two points if they answer correctly on the first attempt and one point for being correct on the second attempt. The first player to reach 30 points is the winner.

Your task is to design a computer version of the game for up to four players. You must analyse the problem and list all of the requirements; **decompose** the problem, list all the sub-problems and write a brief description of each; list all of the input, output and processing requirements.

One of the requirements that will have to be modelled is the spinning of the wheel. Using a written description and pseudocode shows how this could be done.

GENERAL VOCABULARY

decompose to divide something into smaller parts

EXTEND YOUR KNOWLEDGE

Complete the noughts and crosses game by coding it in the language you are studying. See if you can end up with a working game.

CHECKPOINT

Strengthen

SKILLS CRITICAL THINKING

S1 What is meant by 'decomposition'? What are the benefits it provides for programmers?

SKILLS CRITICAL THINKING

S2 What is meant by 'abstraction'?

SKILLS CRITICAL THINKING, PROBLEM SOLVING

S3 A student is creating a model of the cost of a car journey, a real-world problem. Write down the important items she will have to include in her model and how they interact to calculate the cost of the journey.

Challenge

SKILLS CRITICAL THINKING

C1 Can you think of some examples when 'decomposition' and 'abstraction' are used when solving a problem?

SKILLS PROBLEM SOLVING

C2 In your own words, can you explain what is meant by 'computational thinking'?

SKILLS DECISION MAKING

C3 Explain how we use abstraction in our daily lives when we are communicating with others.

DID YOU KNOW?

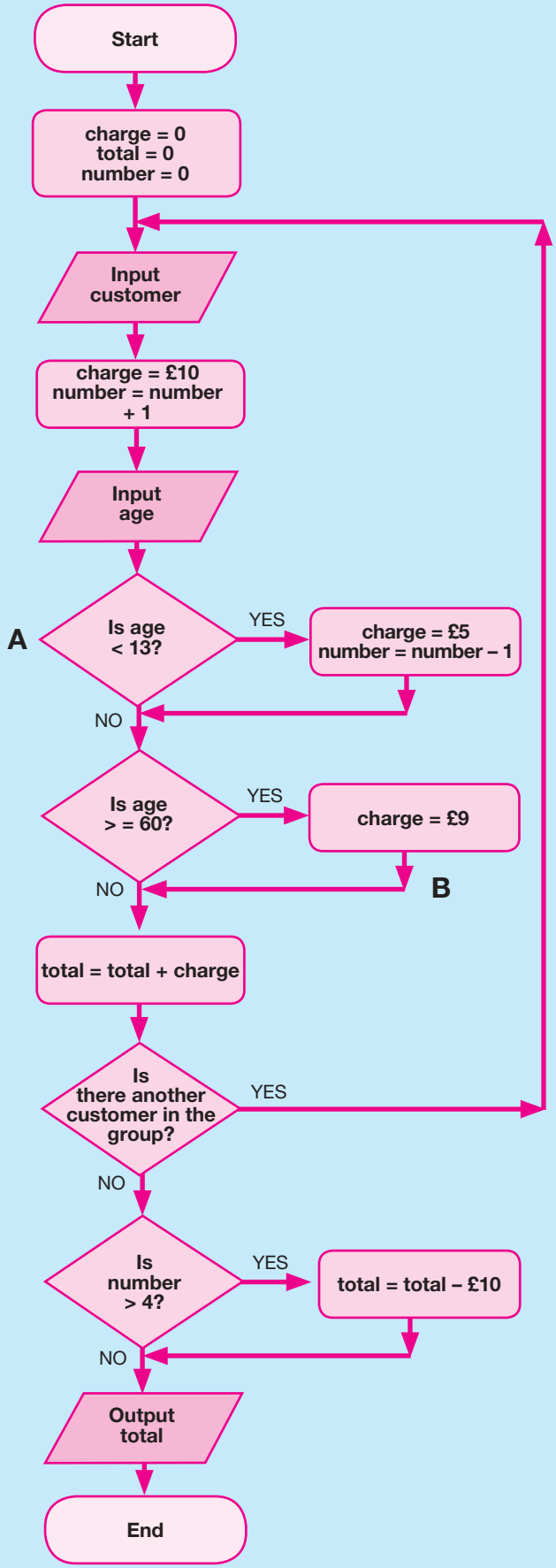
Computer models or 'simulations' of real life are widely used. It is far cheaper and safer to train pilots on flight simulators than on real aircraft. They are also used in weather forecasting, designing and testing new cars and bridges and even teaching people to drive. Computer models are used by all governments around the world to experiment with the short and long-term effects of changing variables such as tax rates on the economy.

How confident do you feel about your answers to these questions? If you're not sure you answered them well, re-read the sections about 'Decomposition' and 'Abstraction'.

SUMMARY

- Computational thinking is an approach to solving problems, such as traffic flow in a city, or how many products a business needs to make and sell to produce a profit. It includes techniques such as decomposition and abstraction.
- Problems are easier to solve if they are decomposed into smaller sub-problems.
- Abstraction is used to remove unnecessary detail to make a problem easier to understand and solve. For example, when modelling traffic flow in a city, unnecessary details could include the colours of the vehicles or the ages of the drivers.
- When designing a solution to a problem the inputs, outputs and processing requirements should be identified at the outset.

UNIT QUESTIONS



▶ **Figure 1.13** Flowchart of an algorithm showing charges in a theme park

The flowchart in Figure 1.13 displays an algorithm used by Holiday Theme Parks Limited.

SKILLS PROBLEM SOLVING A02

1 Explain how the algorithm calculates the total amount that should be paid. **(4)**

SKILLS PROBLEM SOLVING A02

2 Give two variables that are used in the algorithm. **(2)**

SKILLS PROBLEM SOLVING A02

3 In the flowchart, two of the constructs are labelled A and B. State the type of each construct. **(2)**

SKILLS PROBLEM SOLVING A02

4 The Lim family is visiting the park. The family consists of two children, one aged 8 and one aged 10, their two parents and their grandfather, who is aged 65. Use the algorithm to calculate how much the family should have to pay for entry. **(4)**

HINT

Spend some time studying the algorithm to ensure that you fully understand it.

In **1** you are asked to 'explain' how the algorithm works. A longer answer is required, which includes all of the stages of the algorithm. Use the correct terms to explain the constructs.

In **2** and **3** short answers are sufficient.

In **4** calculate the charge for each person using the rules of the algorithm. Then calculate the overall charge and check to see if the family qualifies for a group discount.

SKILLS PROBLEM SOLVING A02

5 A teacher has stored learner surnames as shown below.

Marek	Jackson	Bachchan	Wilson	Abraham	French	Smith

Identify the stages of a bubble sort when applied to this data. **(5)**

SKILLS PROBLEM SOLVING A02

6 The teacher has a sorted list of names from another class as shown below.

Azikiwe	Bloom	Byrne	Davidson	Gateri	Hinton	Jackson	Linton	Smith	Wall

Identify the stages of a binary search to find the name 'Jackson' when applied to this list. **(4)**

HINT

These questions are testing knowledge of the sort and search algorithms.

In question **5**, the answer should be set out to show how the data is progressively sorted using the bubble sort and the result of each pass should be shown.

Question **6** is to check that you know that this is a **recursive method** where the median is repeatedly selected.

SUBJECT VOCABULARY

recursive method a recursive method calls a function over and over until a required goal is met

SKILLS PROBLEM SOLVING A03

7 Create an algorithm to calculate the cost of sending a parcel.

If the weight of the parcel is 2 kg or under then the standard charge is $3. There is then a charge of $2 for each extra kilogram up to 10 kg. After 10 kg the charge per extra kilogram is $3.

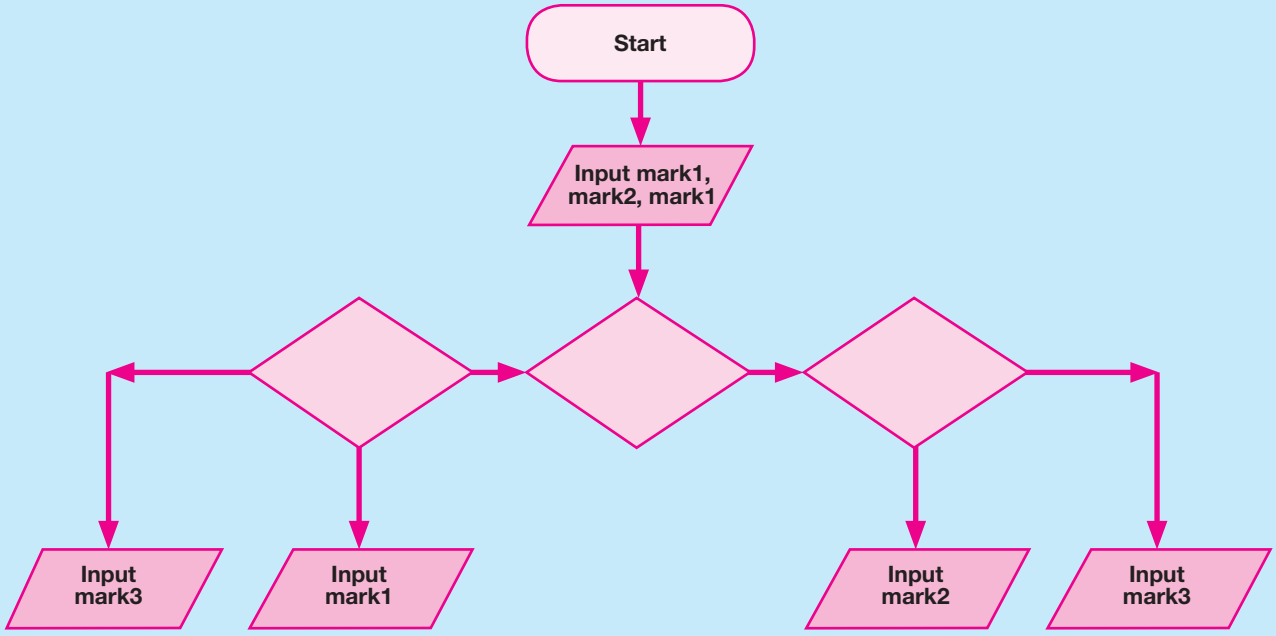
a Display your algorithm as a flowchart. **(5)**

b Construct your algorithm as pseudocode. **(5)**

SKILLS PROBLEM SOLVING A03

8 A learner hands in three homework assignments, which were each given a mark out of 10. All of the marks were different. The following is part of an algorithm to find the highest mark but some of the decision symbols are empty.

Complete the decision symbols and add 'YES' and 'NO' labels where required. **(6)**



SKILLS PROBLEM SOLVING A02

9 A list is made up of the numbers 4, 1, 2, 6, 3, 5.
Identify the steps involved when sorting this list using a bubble sort algorithm. **(2)**