Preparation
Read through the Student Book for this chapter before you begin the topic. Make sure you understand what students should already know, and that you understand the key words and concepts introduced in this chapter.
You may want to start a paper folder for individual students so they can collate their assessed written work for each lesson, and the outcomes of their ‘Test yourself…’ exercises.

Learning outcomes
In this chapter students use search algorithms and sort algorithms to design a route planning system (e.g. a satellite navigation system for a car). They are guided, step by step, through every part of this activity. They are introduced to five types of algorithms: exhaustive algorithms, insertion and bubble sort algorithms, and serial and binary search algorithms. Most importantly, students apply their developing computational skills to understanding a real-world problem. They choose the correct algorithms for solving that problem.
By completing this chapter students will be able to:
- use computational thinking to solve problems and explain their thinking to others
- use pseudocode to show their computational thinking
- design exhaustive algorithms
- use insertion and bubble sort algorithms
- use serial and binary search algorithms
- write search algorithms as pseudocode
- use an exhaustive algorithm and a greedy algorithm to find the fastest route.

Students will also develop their understanding of computer systems. They will be able to explain:
- the definition of a model
- the differences between the different types of algorithm, including their strengths and weaknesses.

Design a route planner
Offline activity
The offline activity introduces students to a key idea in the chapter without using computers. You could use this activity to introduce the project. Students choose ten small objects from around the classroom to take back to their tables. Working in pairs, students are asked to sort their items into a sequence. For example, they could sort the items by size from smallest to biggest, or by density from heaviest to lightest.
Students are then asked to sort the items into groups. For example, they could group the items by use, such as placing all pencils together, all rulers together. They could group the items by size.

Talk about...
The discussion is also an activity you can do offline. You could use this activity any time to vary the pace of lessons and encourage students to reflect on their learning.

Although the Student Book does not specify a discussion activity, you can encourage students to explain to each other why they have chosen to place items in each group. Encourage students to talk about and reflect on their thinking and decision-making processes. This is also an opportunity for those with English as an additional language to practise talking about their ideas. You could encourage students to understand and use words such as: sort, order, sequence, pattern, choice, because, set, group, organise and manage.

FACT
Some students find it challenging to understand the rate of change we have seen in computing over the past 70 years. Use the fact provided to discuss how rapidly technology has changed in students’ lifetime. They may talk about improved gaming devices or mobile technologies, or more widespread access to technology. If you have access to the Internet, you could show students images of computers through the ages, such as those shown here.

http://www.computerhistory.org/timeline/computers/

Encourage students to talk about the images by asking these questions.
- How big are the computers?
- Who is using the computers?
- How powerful are the computers in the pictures compared to the power of a modern mobile phone?

Word cloud
The Word cloud contains all the key words that are highlighted and defined in Key words boxes in the lessons. The key words for this chapter are: algorithm, iteration, selection, pseudocode, abstraction, exhaustive search, bubble sort, insertion sort, sort algorithm, binary search algorithm, search algorithm, serial search algorithm and greedy algorithm.

---

1.1 What do I know? pages 8–11

Learning outcomes
When they have completed this lesson students should be able to:
- explain their computational thinking to others
- use computational thinking to solve problems.

More-confident students will:
- consider a more complex variation on the problem presented in the lesson.

Overview
In this lesson students review the key words and concepts introduced in Matrix 1, Chapter 1, Computational Thinking. This lesson is an opportunity to ensure that any students who have not completed Matrix 1 understand the foundations of computational thinking. Students will begin the project of the chapter by decomposing the problem of route navigation.
Language development
The lesson reminds students that computational thinking offers a set of tools they can use to solve real-world problems. To find a solution, students will use a set of rules or instructions called algorithms. They will use:
- decomposition to break problems down into smaller, solvable parts
- pattern recognition to find patterns in a problem that could help to make the solution more effective
- selection to show a step in an algorithm where there is more than one possible path
- iteration to show repeating steps in an algorithm
- flow charts to represent their computational thinking.

Before the lesson
Make sure you are confident with the key words and concepts. Students will move quickly through the material and you may need to explain terms as the lesson progresses.

The key words for this lesson are: algorithm, iteration and selection. The words are highlighted when they first appear in the text. Their definitions are included in the Key words box at the end of the lesson. You may want to review these words before the lesson.

If you want to play the game in the ‘Learn about…’ part of the lesson, you will need a hat, gloves and any other suitable outer layers of clothing.

Learn about...
You will lead the first part of the lesson. Make sure students understand these ideas. You may ask them to make notes. You may use directed questioning to check understanding.
- Computational thinking is a way of understanding a problem, and finding the best solution for the problem.
- In computational thinking, problems are broken down into small steps called decomposition. The rules or instructions we develop to solve problems are called algorithms.
- Sequencing, selection and iteration are the most important basic tools used to design algorithms. Explain the difference between these three terms to students.

If students struggle to understand the difference between sequencing, selection and iteration, play this game.
- Ask for a volunteer. Explain that the volunteer needs to put on the items of clothing you have brought into the lesson.
- Ask the class to suggest a sequence, or order, in which the volunteer could put on the clothing. Encourage them to explain their reasoning. Ask the volunteer to try this sequence and discuss its strengths and weaknesses.
- Explain that the class, for example, only gave instructions for one glove to be put on. The instruction needs to be iterated so that both gloves can be put on.
- Explain that, at each stage, the volunteer could choose to put on one or another item of clothing. The volunteer needed to select the most sensible item to put on next.

How to...
In the second part of the lesson students complete an exercise under your guidance. This part of the lesson models the way in which we develop flow charts.

Students can work as a class to create a flow chart for a robot hammering a nail into a piece of wood. You may want them to do this before looking at the example in the Student Book. Compare the flow chart they have created with the one in the book.

Point out these important features of the example flow chart.
- Start and stop: Ask students what would happen if these steps in the sequence were left out. The program would not start, or it would not stop.
- Selection is shown in a decision point, which asks the question: Is the nail fully in the wood? There are two possible answers. If the nail is not fully in the wood, the algorithm iterates, or loops, back to Step 2. This makes the hammer hit the nail again. If the nail is fully in the wood, then the algorithm can stop.
- The algorithm stops when the nail is fully in the wood.

Now you do it...
Students will begin the chapter project, which is to design a route navigation system based on computational thinking processes.

The project begins with a simple map. Students are asked to decompose the problem of planning a route from point A to point C on the map.
The Student Book provides a set of simple questions students can use to decompose the problem.

- How many points are there on the map? Answer: There are four points on the map. Make sure students are not counting all landmarks or named locations on the map as points in the route.
- How many routes can you see between point A and point C? Answer: Most students should be able to identify two routes.

The activity also encourages students to talk with one another about their decompositions. This is an opportunity for them to provide helpful feedback to one another on their computational thinking.

**What success looks like:** Students identify all the points on the map and begin to identify routes.

**If you have time...**

The extension activity asks more-able students to think about what would happen if they were walking from point A to point C on the map, rather than driving.

**What success looks like:** Students should see that they could walk directly from point A to point C without intersecting with any other point, across Green Park.

Discuss the importance of understanding the context of problems we are trying to solve using computational thinking.

### Test yourself...

**FOUNDATION QUESTIONS**

1. What is a loop? Answer: To loop means to go back to a previous step.

2. A loop is used in which construct or tool? Answer: Iteration.

**EXTENSION QUESTIONS**

Create a flow chart of something you do every day, such as brushing your teeth, getting dressed or going to bed.

3. If you have used a loop in your flow chart, draw an arrow to it and label it. Answer: You can expect to see a loop in the flow chart of any activity where there is a repeating step or action. For the examples suggested, you would see a loop in an activity about brushing hair or teeth, or putting on two socks or shoes. The most-able students will include a decomposition as well as the flow chart without being prompted. Students who have done this should receive extra credit.

4. If you have not used a loop, write a sentence explaining why you do not need a loop in your flow chart. Answer: There are no repeating actions or steps in the problem.

---

**1.2 Using pseudocode**

**Learning outcomes**

When they have completed this lesson students should be able to:

- use pseudocode to show their computational thinking.

More-confident students will:

- compare using pseudocode and flow charts, and consider their personal preferences.

**Overview**

In this lesson students are introduced to pseudocode. Pseudocode is not a programming language. It is a different way of showing computational thinking and communicating this type of thinking to others. Pseudocode is used extensively in the real world. For example, software engineers and computer scientists use it. This is an important foundation lesson. During the lesson students will have opportunity to compare flow charts and pseudocode. Students then apply their understanding to the route navigation system problem.
Language development

The key word for this lesson is pseudocode. This is specialist technical vocabulary. The term will be unfamiliar to most students, regardless of language background. Take care to explain the word during the lesson.

As the Fact box at the end of the lesson explains, the word ‘pseudocode’ is made up of two words: pseudo and code. ‘Pseudo’ comes from a Greek word meaning false or lies. Pseudocode is ‘false code’. Pseudocode is a way to describe algorithms. Students will use pseudocode to show their computational thinking for a program, and to show their problem-solving.

Before the lesson

Practise the lesson in advance so you are confident. To support students with English as an additional language, prepare strips of paper or card with each line of pseudocode in the ‘Now you do it...’ activity. Students could then rearrange these into the correct order.

Learn about...

You will lead the first part of the lesson. Make sure students understand these ideas. You may ask them to make notes. You may use directed questioning to check understanding.

- **Pseudocode**: Pseudocode uses words instead of a diagram to describe algorithms, show planning, show problem-solving and show multiple solutions for a problem.

- **Students need to know how to use pseudocode**: As students become more confident, they may find they prefer it to using flow charts. Some people prefer to use flow charts.

- **Fit-for-purpose algorithm**: A fit-for-purpose algorithm is decomposed, efficient, correct and elegant.

- There are generally no articles (‘a’, ‘the’) in pseudocode.

How to...

In the second part of the lesson students complete an exercise under your guidance. In the first part of the exercise students are reminded of the flow chart they worked on in Lesson 1.1. In the flow chart a robot hammers a nail into a piece of wood. Students are then shown how to recreate the same algorithm using pseudocode. Two examples are given. The first is simple. The second adds an output instruction to complete the algorithm more thoroughly. Highlight the following points to students.

- We use indentations to show sub-tasks or instructions inside the algorithm. An indentation means we include a white space before the start of the line, for example:

  ```plaintext
  REPEAT
  Hit nail with hammer
  ```

- We use `REPEAT` to indicate a loop in the algorithm. The loop is indented.

- In the *Matrix* series, we have put words that are computational constructs in capitals. These are: `FOR`, `IF`, `INPUT`, `OUTPUT`, `REPEAT`, `SET`, `STORE`, `THEN`, `UNTIL`, `WHILE`, `DEFINE` and `RETURN`. However, pseudocode does not follow a rigid syntax. Different programmers have different styles of writing pseudocode. Therefore, students should not be penalised if they use capitalisation differently.

- Point out that in the Student Book the output is shown in quotation marks.

Now you do it...

Most students should be able to do the learning activity, which asks students to write pseudocode to instruct a driver to move around a route. The route begins and ends at point A on the map.

Students who have English as an additional language may need help in finding the right instruction words. For these students, provide the sentences in pseudocode you have printed out on strips of paper or card. Ask students to rearrange the pseudocode into the correct order.

What success looks like: There are many possible answers. This is an example of the kind of algorithm students might produce:

```
Start
REPEAT
Drive to next point
IF back at A
THEN stop
Go back to REPEAT
```

If you have time...

More-confident students are asked to work with a partner to discuss whether they prefer pseudocode or flow charts for showing their computational thinking, and to justify their preferences.
What success looks like: Students will explore the strengths and weaknesses of both types of structured notation. For example, they will notice that the pseudocode is quicker to write, while the flow chart shows more detail.

Test yourself...

FOUNDATION QUESTIONS

1. Describe what pseudocode is using your own words. Answer: Most answers should show that students understand that pseudocode is not a programming language. They will know that pseudocode is a way of describing algorithms, and/or showing our computational thinking.

2. How would you show a loop in pseudocode? Answer: Pseudocode can be whatever students want to use to show their thinking. Students can use words such as FOR, WHILE or REPEAT to show loops. You can show the loop in the algorithm by indenting it.

EXTENSION QUESTIONS


4. Create a short algorithm in pseudocode for cutting up an apple. Answer: There are several possible answers. This is an example of the kind of algorithm students may produce. Make first cut of apple is not essential in the pseudocode, but will encourage students to think about each step in the algorithm.

Start
Make first cut of apple
REPEAT
  Choose largest piece
  Cut largest piece
  Are all pieces small enough?
  THEN stop
  Go back to REPEAT

1.3 Exhaustive search algorithms

Learning outcomes

When they have completed this lesson students should be able to:

- design an exhaustive search algorithm.

More-confident students will:

- think about the strengths and weaknesses of exhaustive search algorithms.

Overview

Students who have completed Lessons 1.1 and 1.2 will understand the basic foundation concepts of computational thinking. They will also be able to use simple pseudocode. In this lesson students will apply their understanding to one particular type of algorithm, called an ‘exhaustive search’. Students are introduced to abstraction as a way of showing problems in a clear way. They use abstraction and an exhaustive search in their route navigation system problem.

Language development

There are two key words for this lesson. Abstraction means to take away any unnecessary detail to help us to see and understand problems more clearly.

Understanding problems more clearly helps us to create better, more fit-for-purpose solutions. Abstraction is a useful thinking tool for many aspects of life, and it is important in computational thinking.

Exhaustive search algorithms are related to the word ‘exhaustion’, which means to use something up or to be in a state of extreme tiredness. In computer science this type of search algorithm uses up all of the possible solutions to a problem, checking whether each solution is possible.

Before the lesson

Make sure you are confident with the concepts of abstraction and exhaustive search algorithms. Make sure you understand the maps, which are used to
illustrate abstraction and exhaustive searches in the ‘How to…’ part of the lesson. Be prepared to create new routes and times to reinforce learning if you find that students are struggling.

Learn about...
You will lead the first part of the lesson. Make sure students understand these ideas. You may ask them to make notes. You may use directed questioning to check understanding.

- **Abstraction:** We can use abstraction to take away detail on the map that is not needed. We do not need to know street names, or where green space or landmarks are for this version of the problem. We can show the abstracted problem in a simple way that anyone can understand. Encourage students to compare the two abstractions on pages 16 and 17. Ask them what the two diagrams have in common and where they are different.

- **Exhaustive search:** A search, or searching, algorithm is used to find whatever data you need to solve a problem. The algorithm means that you can instruct a computer to search quickly through large amounts of data. There are many different types of search algorithm. Exhaustive search is one type of search algorithm. An exhaustive search finds all of the possible solutions to a problem, and checks whether each solution is possible.

How to...
In the second part of the lesson students complete an exercise under your guidance.

- **Draw a more complicated abstraction:** The route possibilities given on the map are now more complicated. There are more possible routes and the idea of adding time in minutes is introduced. Students can now work out which is the quickest route between points. If students are struggling with the additional information, you may want to create a new route with them on the map, add in your own time in minutes, and reinforce their understanding by going through the process again.

- **Exhaustive search algorithm:** Using the abstraction, show students how to write all of the possible routes from point A to point C. It is most logical to start with point A.

  - Show students that they can go from point A to points B and F. We would write this as AB or AF.
  - From AB we can then go to ABC or ABE.
  - At ABC, we have reached our destination.
  - From ABE we can go to ABEF (which would take us back to A, so does not solve the problem) or ABED, and then ABEDC to reach our destination.

If students do not understand, work through another example, such as travelling from point A to point E.

Now introduce students to the time in minutes. Show them that from point A to point B takes three minutes. From point B to point C takes five minutes. This means that to travel from point A to point C takes $3 + 5$ minutes (eight minutes) in total.

If students do not understand, work through another example.

Now you do it...
Students are asked to use the exhaustive algorithm to answer questions on the map. Students can use the same time in minutes as for the ‘How to…’ exercise.

What success looks like:

- Work out the exhaustive search algorithm to get from point A to point D. Answer: The outcomes of the algorithm are: ABCD, ABED, AFED, AFEBCD.
- Which is the quickest route or routes? Answer: ABED ($6 + 3 + 4 = 13$ minutes)
- Which is the slowest route or routes? Answer: AFEBCD ($9 + 1 + 3 + 5 + 5 = 23$ minutes)
- Which routes would take the driver past Buckingham Palace? Answer: AFGD, AFED and AFEBCD

If you have time...
More-confident students are asked to make a list of the strengths and weaknesses of exhaustive algorithms.

What success looks like:

- **Strengths:** Exhaustive algorithms can be simple. They are used in a wide range of settings and are thorough.
- **Weaknesses:** Exhaustive algorithms might not be efficient since they go through every possible solution, making them slow.
1.4 Sort algorithms

Learning outcomes

When they have completed this lesson students should be able to:

- use insertion and bubble sort algorithms.

More-confident students will:

- explore two other sort algorithms.

Overview

In Lesson 1.3 students were introduced to a type of search algorithm. In this lesson they find out about two types of sort algorithm. Students sort famous London landmarks to practise their skills and reinforce their understanding. They then apply their understanding to sorting a list for their route navigation systems.

Language development

By the end of the lesson students should be comfortable with the words ‘sort’ and ‘sort algorithm’. To sort means to arrange items into a particular order or groups, and it means the same thing in computational thinking.

There are many different types of sort algorithm. The names of the algorithms will give students an idea about how the algorithms work. Point the names out to students during the lesson.

- Insertion sort algorithms ‘insert’ items into a list, one by one, to get the list into order.
- Bubble sort algorithms compare items on a list, and then swap items into the correct order. The correct items ‘bubble’ to their place on the list.

Before the lesson

Make sure you are confident with the way the insertion sort and bubble sort algorithms work. Insertion sorts have been represented as dance formations on several YouTube videos. If it is appropriate for your setting, you may want to have these videos ready to show during the ‘How to...’ part of the lesson.

https://www.youtube.com/watch?v=ROalU37913U
https://www.youtube.com/watch?v=lyZQPjUT5B4

If these videos are not available in your setting, look for alternatives by searching for ‘insertion sort dance video’ or ‘bubble sort dance video’ using a browser.
Learn about...

You will lead the first part of the lesson. Make sure students understand these ideas. You may ask them to make notes. You may use directed questioning to check understanding.

- Sort algorithms put items into correctly ordered lists. The lists can be alphabetical or numerical. Programs run more smoothly when lists have been sorted. It is also easier to implement a search algorithm on a sorted list.
- There are different types of sort algorithm.
- One type of sort algorithm is an insertion sort. In an insertion sort we insert items into the list in the right order.
- Another type of sort algorithm is a bubble sort. In a bubble sort we compare and swap items on a list until they are in the correct order.

How to...

In the second part of the lesson students complete an exercise under your guidance. This part of the lesson will demonstrate how to carry out alphabetical insertion and bubble sorts.

This is an opportunity to show the YouTube dance videos (mentioned in the ‘Before the lesson’ section). The videos could help you present the concept of insertion sorts in another way.

Insertion sort

First, students carry out an insertion sort. You may need to explain the term ‘alphabetical order’ if you have students in the class with English as an additional language. Students begin by placing Big Ben and Tower Bridge in the correct order. They then choose each name on the list in turn, putting each name in the correct place on the list. Students do this until all of the names are in alphabetical order.

- The unsorted list is:
  Big Ben, Tower Bridge, Buckingham Palace, London Eye, Trafalgar Square, Tower of London.

  - The first two names are compared. Big Ben comes before Tower Bridge in the alphabet, so these names stay in the order given.
  - We compare the next pair (Tower Bridge and Buckingham Palace). ‘B’ comes before ‘T’ in the alphabet, so we swap these names around. The list looks like this:
    Big Ben, Buckingham Palace, Tower Bridge, London Eye, Trafalgar Square, Tower of London
  - We compare the next pair (Tower Bridge and London Eye). ‘L’ comes before ‘T’ in the alphabet, so we swap these names around. The list now looks like this:
    Big Ben, Buckingham Palace, Tower Bridge, Trafalgar Square, Tower of London
  - We compare the next pair (Trafalgar Square and Tower of London). ‘To’ comes before ‘Tr’ in the alphabet, so these names stay in the order given.
  - We compare the next pair (Trafalgar Square and London Eye). ‘Tr’ comes before ‘To’ in the alphabet, so we swap these names around. The list now looks like this:
    Big Ben, Buckingham Palace, London Eye, Tower Bridge, Tower of London, Trafalgar Square.

If you have time you could go through the list again to make sure all of the names are in the right order. A bubble sort will go through a list again and again until it is correct.

Now you do it...

Most students will now understand the difference between an insertion sort and a bubble sort. They should be able to carry out the exercises.

What success looks like: The first exercise asks students to compare the advantages and disadvantages of the two types of sort algorithm. Answers might include:

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Advantages</th>
<th>Disadvantages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Insertion</td>
<td>Insertion sorts are easy to understand</td>
<td>It takes a long time to write each step</td>
</tr>
<tr>
<td>sort</td>
<td>They are thorough</td>
<td></td>
</tr>
<tr>
<td></td>
<td>You can easily add new items to the list</td>
<td></td>
</tr>
<tr>
<td>Bubble</td>
<td>Bubble sorts are easy to understand</td>
<td>Bubble sorts are slower than insertion sorts</td>
</tr>
<tr>
<td>sort</td>
<td>They are thorough</td>
<td>You cannot easily add new items to the list</td>
</tr>
</tbody>
</table>

The second exercise asks students to organise items into an ordered list. The list can be alphabetical or numerical. Students can use any type of sort algorithm they like.
Test yourself...

FOUNDATION QUESTIONS

1 Explain how each of these algorithms work in your own words: insertion sort, bubble sort. Answer: Insertion sort: Students should write about inserting items into a list. Bubble sort: Students should write about comparing pairs of items, and swapping where needed until the list is sorted.

2 Why are sort algorithms important in computer science? Answer: Sort algorithms place data items in an ordered list. Computer programs can run more smoothly using sort algorithms. It is easier to search for an item in an ordered list.

If you have time...

More-confident students can use the Internet to explore two other sort algorithms. Students could use the BBC Bitesize website.

What success looks like: Students could explore bucket sort, merge sort, shell sort or quick sort.

What success looks like: The alphabetical list should look like this:
A to B, A to F, B to C, B to E, C to D, E to D, F to E, F to G, G to D

The numerical list should look like this:
F to E = 1 minute, G to D = 2 minutes, B to E = 3 minutes, F to G = 3 minutes, E to D = 4 minutes, B to C = 5 minutes, C to D = 5 minutes, A to B = 6 minutes, A to F = 9 minutes

EXTENSION QUESTIONS

3 Look at these six images of a bus. Use an insertion sort to place these buses in order from smallest to largest. Answer: The final list should be: e, d, a, b, c, f.

4 Use a bubble sort to place these buses in order from largest to smallest. Answer: The final list should be: f, c, b, a, d, e.

More search algorithms

Learning outcomes

When they have completed this lesson students should be able to:
- use serial and binary search algorithms
- write search algorithms as pseudocode.
More-confident students will:
- explore other search algorithms.

Overview

Different elements from previous lessons come together in this lesson. In Lesson 1.3 students were introduced to exhaustive search algorithms. This lesson introduces students to two more search algorithms. Students also bring together their understanding of different sorts of algorithm using their early pseudocoding skills. They use these skills to begin to solve the route navigation problem.

Language development

There are many different types of search algorithm. The algorithm names will give students a clue about how the algorithms work. Make sure you point out algorithm names during the lesson.

- Binary search algorithms divide an ordered list in half repeatedly until the item is found. We say ‘binary’ because we keep dividing the list in two, and binary is a base-2 number system.
Serial search algorithms look at each item on a list in turn, until the item is found. Serial search refers to the word ‘series’, which means related items coming one after another.

**Before the lesson**

Make sure you are confident with binary search algorithms and serial search algorithms.

**Learn about...**

You will lead the first part of the lesson. Make sure students understand these ideas. You may ask them to make notes. You may use directed questioning to check understanding.

- **Search algorithms**: We use search algorithms in any problem where the computer needs to find something. For example, in the route navigation system problem we need to find a single landmark or address among thousands of street names. We must then find a good route from our current position to that point. There are many different types of search algorithm. In this lesson students will find out about serial and binary searches.

- Programmers often use an underscore to join two words together in pseudocode. This is because programming languages don’t allow spaces in variable names. You will notice that underscores are used in the Student Book for this reason.

- **Question for students to consider**: What other problems could be solved with a search algorithm? Example answers include: finding websites in an Internet browser, finding errors in a program, finding information about a customer who is using an online store.

**How to...**

In the second part of the lesson students complete an exercise under your guidance. We want the computer to search for a criterion (singular) or criteria (plural). In the example, the street name Pall Mall is the criterion.

- **Serial search**: In a serial search the computer goes through the list, one item at a time, and checks whether the item matches the criterion. The example explains how a serial search could be written in pseudocode. Point out that:
  - the first line is defining what the process, or procedure, will be for the program
  - the FOR loop is indented
  - **RETURN** ends the procedure.

**Binary search**: You could play a game to reinforce students’ understanding of a binary search. Ask a volunteer to come to the front of the class and think of a number between 0 and 100. The volunteer must keep the number secret. The rest of the class should try to guess the number in as few guesses as possible. After each guess the volunteer must say whether the guess is too high or too low. Point out that if students make their first guess 50, they are eliminating the half of the list that is not relevant to the volunteer’s number. They could then guess 25 or 75, and so on until they identify the correct number.

**Now you do it...**

Most students will now understand the difference between a serial search and a binary search. They should be able to do the activity.

**What success looks like**: The first part of the activity asks students to compare the advantages and disadvantages of the two types of search algorithm. Answers might include:

<table>
<thead>
<tr>
<th></th>
<th>Advantages</th>
<th>Disadvantages</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Serial search</strong></td>
<td>Serial searches are to understand</td>
<td>Serial searches are slower than binary searches</td>
</tr>
<tr>
<td></td>
<td>They work on any type of list, even an unsorted list</td>
<td></td>
</tr>
<tr>
<td><strong>Binary search</strong></td>
<td>Binary searches are quick</td>
<td>The list needs to be sorted before you can search it</td>
</tr>
</tbody>
</table>

The second part of the activity asks students to write pseudocode for a serial search of the list they sorted in Lesson 1.4. There are several possible answers. Students may produce something like this. Give extra credit for the last line, as this is not essential, but shows good programming practice:

```plaintext
SET route_name as “A to F”
FOR key_name in (list of routes)
    IF route_name equals key_name
        OUTPUT route length for key_name
    STOP
OUTPUT route_name not found
```

**If you have time...**

More-confident students can use the Internet to explore a further search algorithm. Students could use the BBC Bitesize website. Remind students that serial searches are sometimes called linear searches.
What success looks like: Students are most likely to find tree search algorithms.

Test yourself...

FOUNDATION QUESTIONS
1. What is the difference between a serial search and a binary search? Answer: A serial search searches a list item by item, checking whether each item matches a criterion. A binary search divides the list in half, works out which half the item is in, and repeats the process until the item is found.
2. What are the advantages and disadvantages of a serial search? Answer: A serial search is easy to understand, and works on any type of list, even an unsorted list. It is slower than a binary search.

EXTENSION QUESTIONS
3. Why are search algorithms important in computer science? Answer: Search algorithms make it possible for a computer to search for data.
4. When could a search algorithm be useful in a real-life problem? Answer: There are many possible answers. Examples are finding a contact in a mobile phone, looking up an address, finding a file on a computer, finding an online banking transaction.

1.6 The fastest route

Learning outcomes
When they have completed this lesson students should be able to:

- use a greedy algorithm
- compare a greedy algorithm with an exhaustive algorithm.

More-confident students will:

- find out information about important women in computer science.

Overview
Students have been introduced to a number of sort and search algorithms in previous lessons. Students have applied these algorithms to the route navigation problem. In this lesson students learn one more type of algorithm and compare it to the first algorithm they learned.

Language development
This lesson gives students the chance to revise some of the words they have used throughout this chapter. For example, they might want to review: algorithm, efficient and elegant. This lesson also asks students to use language to compare two algorithms. It introduces students to the key word greedy algorithm. ‘Greedy’ is an English language word. In computational thinking, a greedy algorithm is one that is short-sighted. A greedy algorithm does not see the whole problem, but simply solves one small part of a problem at a time. Greedy algorithms were first described by Fibonacci, who is famous for describing a special sequence of numbers that is often found in nature.

Before the lesson
Assess students’ progress. Prepare reinforcement activities for any students who struggle during the ‘Now you do it...’ activity. You might include using different-sized, or different material items students can use to help their understanding. For example, you could provide blocks of different sizes.
Learn about...

You will lead the first part of the lesson. Make sure students understand these ideas. You may ask them to make notes. You may use directed questioning to check understanding.

- **Revision:** Remind students of the sort and search algorithms they have already studied in this chapter. Ask students to work in pairs. They should explain different types of algorithm to one another, then share questions with the class to correct any mistakes.
- **Compare and contrast:** Explain that sometimes the algorithm that seems to be the best one for solving the problem can turn out to be the wrong type. We need to compare and contrast algorithms to make sure we have chosen the right type of algorithm for the problem.

If you have students with English as an additional language, make a list of important words to help students compare and contrast. The list might include words such as: similar to, different from, advantage, disadvantage, instead, unlike, both, the same as, on the other hand, however, yet, or, but.

How to...

In the second part of the lesson students complete an exercise under your guidance.

The Student Book works through a greedy algorithm step by step. The example uses a similar format to the other route navigation problems set during this chapter.

Show students that the greedy algorithm, although simple to write, does not always solve the problem correctly. A greedy algorithm will only find the best solution to one small, decomposed part of the problem. It does not look at the problem as a whole. Computational thinking techniques help break a problem down into smaller parts, and solve the whole problem in the most fit-for-purpose, elegant and efficient way.

Now you do it...

The activity asks students to solve a route navigation problem using a greedy algorithm.

What success looks like:

- Can you work out the route? Answer: No, because the greedy algorithm takes us in the wrong direction for the destination in the first step (AB = 5 minutes, AD = 1 minute, and it is not possible to get from point D to point C).
- Students are then asked to work out the fastest route using an exhaustive algorithm. Answer: ABC = 7 minutes.

If you have time...

More-confident students find out information about three important women in computer science. Encourage students to be critical of the information they find online.

What success looks like: Students will check more than one source for their findings. They might share their work with the rest of the class or display it in the classroom.

Test yourself...

**FOUNDATION QUESTIONS**

1. A serial search algorithm looks at each ........ in a ......... in turn until it finds the ........ it is looking for. Answer: item, list, item.
2. A binary search algorithm repeatedly ........ a ........ list in ........ to find an item. Answer: halves, sorted, order.

**EXTENSION QUESTIONS**

3. Using your own words, write how a greedy algorithm works. Answer: Answers should include information about greedy algorithms choosing which next step will most likely help to solve a part of a problem.
4. Name one strength and one weakness of a greedy algorithm. Answer: Strength: A greedy algorithm works inside a decomposed problem to find a good solution for one part of the problem. Weakness: A greedy algorithm does not always lead to the correct answer, because it does not work on the whole problem.
The test questions and assessment activities in the Student Book give you an opportunity to evaluate students’ understanding. The questions are shown here with possible answers.

**Model answers to test questions**

1. You have a number of tools in your computational thinking toolbox. For each of these tools, explain what they are, and why they are important in computational thinking. Answer:
   a) **Pseudocode:** This is a way to describe algorithms. Using pseudocode can help you show your planning for a program and how you would solve a problem. It can also help you see that there might be more than one way to solve a problem. Pseudocode uses words instead of drawings to show your computational thinking.
   
b) **Iteration:** Iteration is a repeating step in an algorithm or computer program. Iteration is useful because it means we do not have to write out or code the same instruction many times over. Give extra credit if students explain that in algorithms we show iteration in loops.
   
c) **Algorithm:** An algorithm is a set of instructions or rules we create to carry out a task. We can use algorithms to get computers to carry out tasks for us.
   
d) **Sequence:** A sequence is the order of instructions in an algorithm. If we get a sequence wrong, it can mean the algorithm does not work.
   
e) **Selection:** Selection is a step in an algorithm where there are two or more possibilities. We use selection in algorithms to show what the possibilities are, and to show what will happen in the algorithm if any of the choices are taken. Give extra credit if students explain that selection can be represented by a question or a decision.

2. What is a sort algorithm? Answer: A sort algorithm puts items into a particular order. Give extra credit if students explain that there are many different kinds of sort algorithm and offer names of sort algorithms.

3. What is the name of one type of sort algorithm? Answer: Students have learned about insertion and bubble sorts during this chapter. They could also write the name of a sort algorithm they found out about during an extension activity, such as a bucket sort, quick sort or merge sort.

4. What is a search algorithm? Answer: A search algorithm finds items of information inside any type of data structure.

5. What does a binary search do? Answer: A binary search repeatedly divides an ordered list in half to find an item.

6. What is a greedy algorithm? Answer: A greedy algorithm looks for a simple solution to a complicated problem that has many steps. The algorithm works by choosing which next step will give the most benefit for that part of the problem.

7. Use a bubble sort to arrange these children in height order from shortest to tallest. Answer:
   - Compare AB: A is shorter, so the order remains the same.
   - Compare BC: C is shorter so swap these two. The order is now ACBDEF
   - Compare BD: D is shorter so swap these two. The order is now ACDBEF
   - Compare BE: B is shorter so keep these the same. The order does not change.
   - Compare EF: F is shorter so swap these two. The order is now ACDBFE
   - Go back to the start of the list.
   - Compare AC: C is shorter, so swap these two. The order is now CADBFE
   - Compare AD: D is shorter so swap these two. The order is now CDABFE
   - The sort is now complete.
8. Compare the advantages and disadvantages of the two sort algorithms you know about.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Advantages</th>
<th>Disadvantages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Insertion sort</td>
<td>Insertion sorts are easy to understand, thorough, you can easily add new items to the list</td>
<td>It takes a long time to write each step</td>
</tr>
<tr>
<td>Bubble sort</td>
<td>Bubble sorts are easy to understand, thorough</td>
<td>Bubble sorts are slower than insertion sorts, you can't easily add new items to the list</td>
</tr>
</tbody>
</table>

9. Compare the advantages and disadvantages of two search algorithms you know about.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Advantages</th>
<th>Disadvantages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Serial search</td>
<td>Serial searches are easy to understand, thorough, work on any type of list, even an unsorted list</td>
<td>Serial searches are slower than a binary search</td>
</tr>
<tr>
<td>Binary search</td>
<td>Binary searches are quick</td>
<td>The list needs to be sorted before you can search it</td>
</tr>
<tr>
<td>Exhaustive search</td>
<td>Exhaustive searches are easy to understand</td>
<td>Exhaustive searches are slower than serial or binary searches</td>
</tr>
</tbody>
</table>

10. What are the strengths and weaknesses of greedy algorithms?

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Advantages</th>
<th>Disadvantages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Greedy algorithm</td>
<td>Greedy algorithms are quicker than many other algorithms, work best when the problem has been decomposed clearly</td>
<td>A greedy algorithm does not always give you the right solution to the problem</td>
</tr>
</tbody>
</table>

Model answers to assessment activities

Starter activity

All students should be able to complete this activity. They are asked to write pseudocode to show what the avatar will need to do to move from point A to point C.

What success looks like: There are several possible answers. Most students should produce pseudocode that looks similar to this:

Start
Exit ballroom
Move forward
Turn left
Move forward
Turn right
Move forward
UNTIL point C

Intermediate activity

Students are asked to amend their pseudocode to show what would happen if the avatar is instructed to pick up a casket of treasure at point B on the map.

What success looks like: There are several possible answers. Students could produce pseudocode similar to the pseudocode in the starter activity, adding in PICK UP treasure and an extra MOVE forward line. Additional credit should be given to students who show they understand how to represent a loop in pseudocode, for example:

Start
Exit ballroom
REPEAT
  Move forward
UNTIL choice of paths
  Move forward
  TURN left
REPEAT
  Move forward
UNTIL choice of paths
  Move forward
UNTIL point B
Pick up treasure
REPEAT
  Move forward
REPEAT
UNTIL point C

Extension activity

There are two extension activities for confident students. Students can complete these unassisted as all the guidance they will need is in the Student Book. You can concentrate on supporting students who need more help.

- Move forward
- Turn left
- Move forward
- Turn right
- Move forward
- UNTIL point C
There is more than one door in the passageway. All the doors are locked except the door to point C. Show this in your pseudocode.

**What success looks like:** There are many possible answers. Most students should produce pseudocode that looks similar to this:

```plaintext
Start
Exit ballroom
REPEAT
    Move forward
UNTIL choice of paths
Turn left
REPEAT
    Move forward
UNTIL choice of paths
Turn left
REPEAT
    Move forward
UNTIL choice of paths
Turn right
REPEAT
    Move forward
UNTIL point B
Pick up treasure
REPEAT
    Move forward
UNTIL point C
REPEAT
    Try next door
    IF unlocked
        Go through door
UNTIL all doors tried
```

What is behind each door? Show what the avatar would encounter behind each door in your pseudocode.

**What success looks like:** There are many possible answers. Most students should produce pseudocode that looks similar to this:

```plaintext
Start
Exit ballroom
REPEAT
    Move forward
UNTIL choice of paths
Turn left
REPEAT
    Move forward
UNTIL choice of paths
Turn left
REPEAT
    Move forward
UNTIL choice of paths
Turn right
REPEAT
    Move forward
UNTIL point B
Pick up treasure
REPEAT
    Move forward
UNTIL point C
REPEAT
    Try next door
    IF unlocked
        Go through door
        Record what you observe
        Go back to point C
UNTIL all doors tried
```