INSPIRE COMPUTING

International

Student Book

YEAR 9

Pearson

INSPIRE COMPUTING

International

Student Book YEAR 9

Paul Clowrey

Pearson

Published by Pearson Education Limited, 80 Strand, London, WC2R 0RL.
www.pearson.com/international-schools

Copies of official specifications for all Pearson Edexcel qualifications may be found on the website:
https://qualifications.pearson.com

First published 2022

24 23 22
10 9 8 7 6 5 4 3 2 1

British Library Cataloguing in Publication Data
A catalogue record for this book is available from the British Library

ISBN 978 1 292 40429 5

Printed in Slovakia by Neografia

The author and publisher would like to thank the following individuals and organisations for permission to reproduce photographs, illustrations, and text:
KEY (t – top, c – center, b – bottom, tl – top left, tr – top right)

Getty Images: Darren Robb 2, James Woodson 28, Peter Dazeley 50, EmirMemedovski 72, pixhook 98, Thomas M. Scheer/EyeEm 122; **LibreOffice:** LibreOffice 103 cl, 103cr, 105t, 105b, 109, 113tl, 113tr, 113cl, 115, 117cl, 117cr, 117b; **Our World in Data:** Our World in Data 95; **Peter Higginson:** Peter Higginson 56; **Shutterstock:** Bakhtiar Zein 4, Metamorworks 5tr, Captain Wang 5c, SurfsUp 12, 32pixels 13, BallBall14 16, BananyakoSensei 23, William Potter 26, AzmanMD 35l, Aleksandar Grozdanovski 35c, Eshma 35r, Carlos andre Santos 35br, Vladimir Pisarenko 38, Mariia Korneeva 40c, KorradolYamsatthm 40b, Emerald_media 42, Howcolour 44, Production Perig 47t, KirinKhotcharak 47b, Evgeniy pavlovski 48, Akura Yochi 58t, Adam Jan Figel 58b, Xtock 60, Thvideostudio 62, Anake Seenadee 63, Crydo 64cr, Fer Gregory 64br, Aleksei Lazukov 65, Roman Samborskyi 68, Prostock-studio 70, PR Image Factory 84, Iconic Bestiary 92, Zern Liew 94, Gorodenkoff 96, Fboudrias 102, Mixov 104, Bay015 110, Bakhtiar Zein 112, 114, Nanausop 120, Filiz buber 124, Trismegist san 127, Lovelypeace 129, Jiw Ingka 130, Kheng Guan Toh 132, Nmedia 133, Ronstik 134, Art. em.po 136, 0beron 141, Ira Yapanda 142, Lane V. Erickson vii; **UNICEF:** United Nations Children's Fund 94; **YouTube:** 137

Published by Pearson Education Limited, 80 Strand, London, WC2R 0RL.
www.pearson.com/international-schools

Copies of official specifications for all Pearson Edexcel qualifications may be found on the website:
https://qualifications.pearson.com

First published 2022

24 23 22
10 9 8 7 6 5 4 3 2 1

British Library Cataloguing in Publication Data
A catalogue record for this book is available from the British Library

ISBN 978 1 292 40429 5

Printed in Slovakia by Neografia

The author and publisher would like to thank the following individuals and organisations for permission to reproduce photographs, illustrations, and text:
KEY (t – top, c – center, b – bottom, tl – top left, tr – top right)

Getty Images: Darren Robb 2, James Woodson 28, Peter Dazeley 50, EmirMemedovski 72, pixhook 98, Thomas M. Scheer/EyeEm 122; **LibreOffice:** LibreOffice 103 cl, 103cr, 105t, 105b, 109, 113tl, 113tr, 113cl, 115, 117cl, 117cr, 117b; **Our World in Data:** Our World in Data 95; **Peter Higginson:** Peter Higginson 56; **Shutterstock:** Bakhtiar Zein 4, Metamorworks 5tr, Captain Wang 5c, SurfsUp 12, 32pixels 13, BallBall14 16, BananyakoSensei 23, William Potter 26, AzmanMD 35l, Aleksandar Grozdanovski 35c, Eshma 35r, Carlos andre Santos 35br, Vladimir Pisarenko 38, Mariia Korneeva 40c, KorradolYamsatthm 40b, Emerald_media 42, Howcolour 44, Production Perig 47t, KirinKhotcharak 47b, Evgeniy pavlovski 48, Akura Yochi 58t, Adam Jan Figel 58b, Xtock 60, Thvideostudio 62, Anake Seenadee 63, Crydo 64cr, Fer Gregory 64br, Aleksei Lazukov 65, Roman Samborskyi 68, Prostock-studio 70, PR Image Factory 84, Iconic Bestiary 92, Zern Liew 94, Gorodenkoff 96, Fboudrias 102, Mixov 104, Bay015 110, Bakhtiar Zein 112, 114, Nanausop 120, Filiz buber 124, Trismegist san 127, Lovelypeace 129, Jiw Ingka 130, Kheng Guan Toh 132, Nmedia 133, Ronstik 134, Art. em.po 136, 0beron 141, Ira Yapanda 142, Lane V. Erickson vii; **UNICEF:** United Nations Children's Fund 94; **YouTube:** 137

Overview contents

Overview contents

Detailed contents

Detailed contents

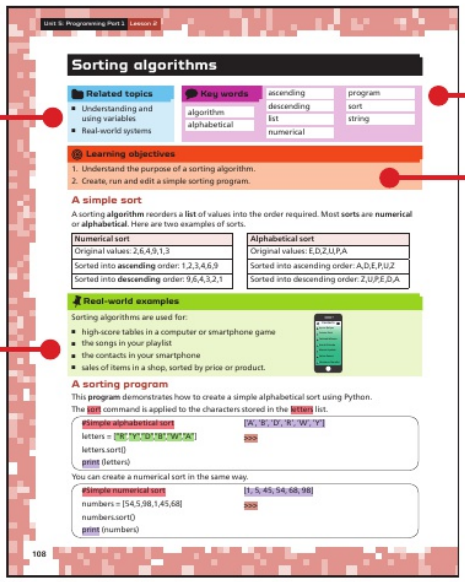
Welcome to Inspire Computing

Whether for school, fun, work or staying in touch with relatives around the world digital technology is all around us.

Through coverage of ICT and Computer Science you will discover how this amazing technology works, how it connects the world together and it has revolutionised the classroom, workplace, and home.

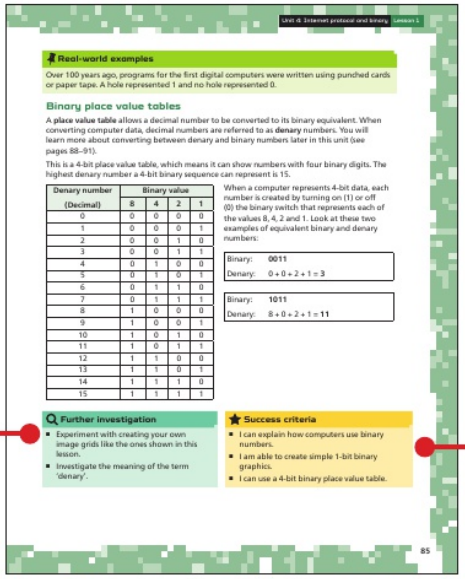
Related topics
Other topics linked to the subject that can also be explored.

Real-world examples
How the learning applies to the world outside the classroom.



Key words
These are important words to know!

Learning objectives
This is what you will know or be able to do by the end of the lesson.



Further investigation
Take your learning from the lesson further!

Success criteria
What has been understood from the lesson.

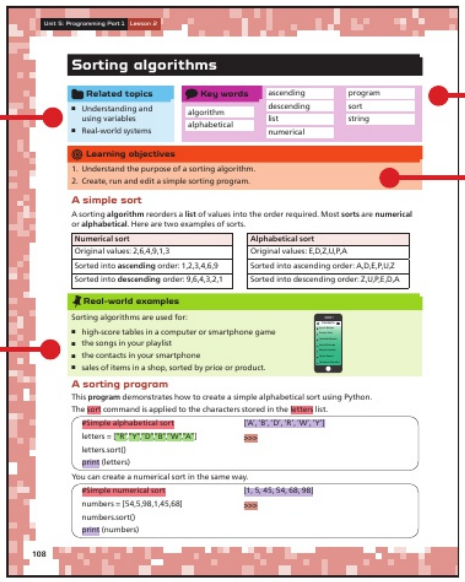
Welcome to Inspire Computing

Whether for school, fun, work or staying in touch with relatives around the world digital technology is all around us.

Through coverage of ICT and Computer Science you will discover how this amazing technology works, how it connects the world together and it has revolutionised the classroom, workplace, and home.

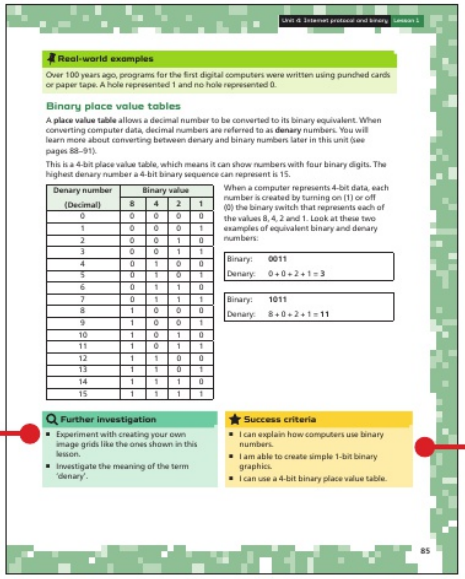
Related topics
Other topics linked to the subject that can also be explored.

Real-world examples
How the learning applies to the world outside the classroom.



Key words
These are important words to know!

Learning objectives
This is what you will know or be able to do by the end of the lesson.

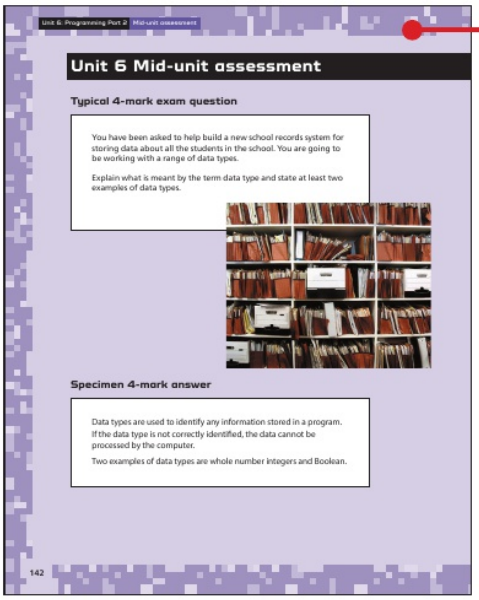


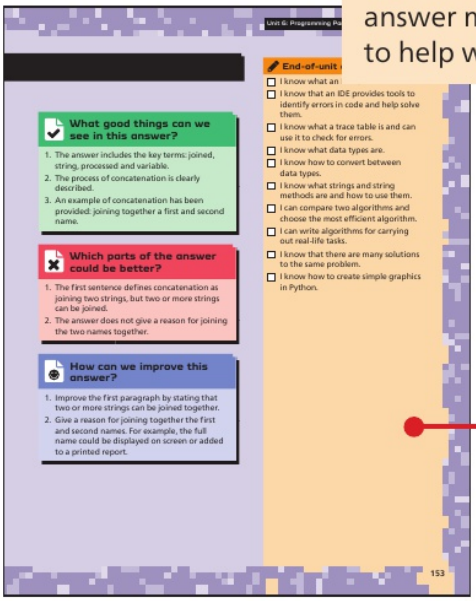
Further investigation
Take your learning from the lesson further!

Success criteria
What has been understood from the lesson.

We hope you will find this book useful in developing your knowledge of digital technology, its effective use of applications and in supporting future learning.

Each topic includes easy to understand theory, real-world examples, and ideas for further investigation. You can also test your knowledge of keywords and regular exam-quality questions with supported answers. A checkpoint at the end of each lesson is a quick and easy way to check your own understanding.





Assessment pages

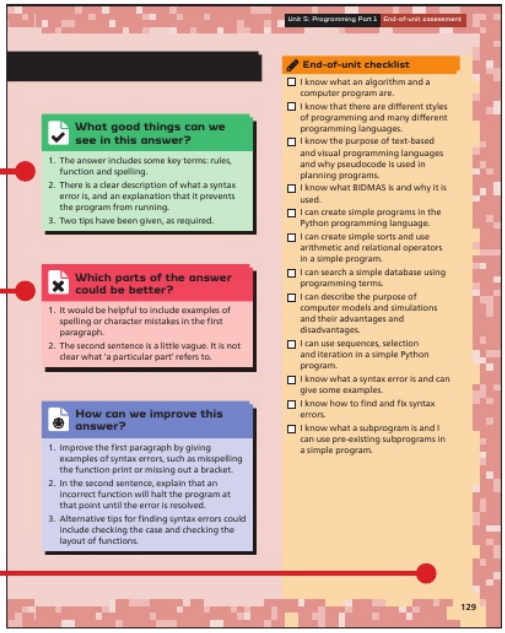
Example of exam question and answer midway and end of unit to help with exam preparation.

Analysis

Evaluation of example answer at mid-unit assessment and end-of-unit assessment to hone analytical skills and provide useful guidance.

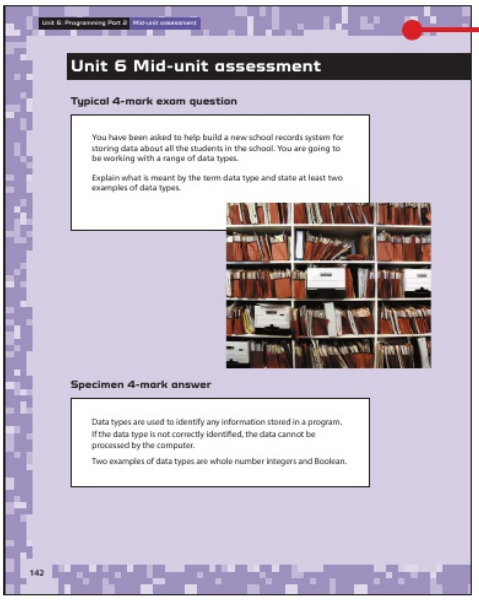
End-of-unit checklist

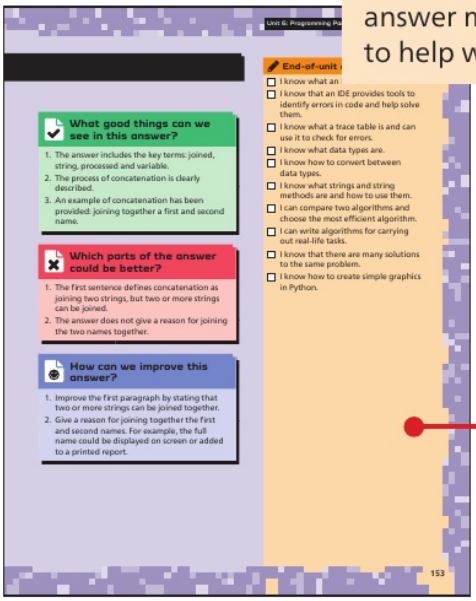
Checklist at the end of every unit to quickly assess your understanding and progress.



We hope you will find this book useful in developing your knowledge of digital technology, its effective use of applications and in supporting future learning.

Each topic includes easy to understand theory, real-world examples, and ideas for further investigation. You can also test your knowledge of keywords and regular exam-quality questions with supported answers. A checkpoint at the end of each lesson is a quick and easy way to check your own understanding.





Assessment pages

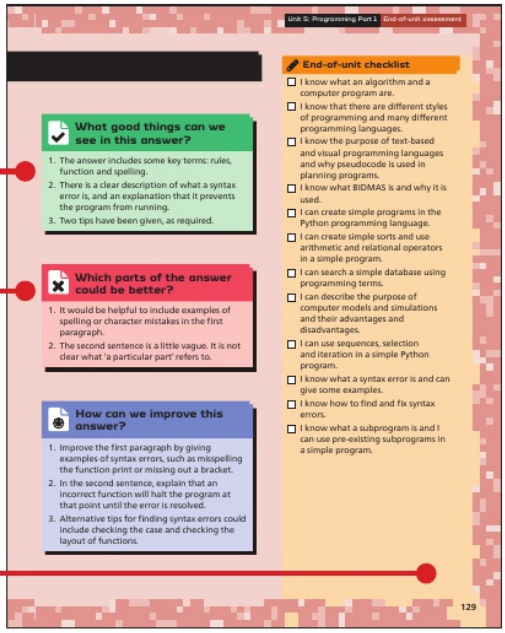
Example of exam question and answer midway and end of unit to help with exam preparation.

Analysis

Evaluation of example answer at mid-unit assessment and end-of-unit assessment to hone analytical skills and provide useful guidance.

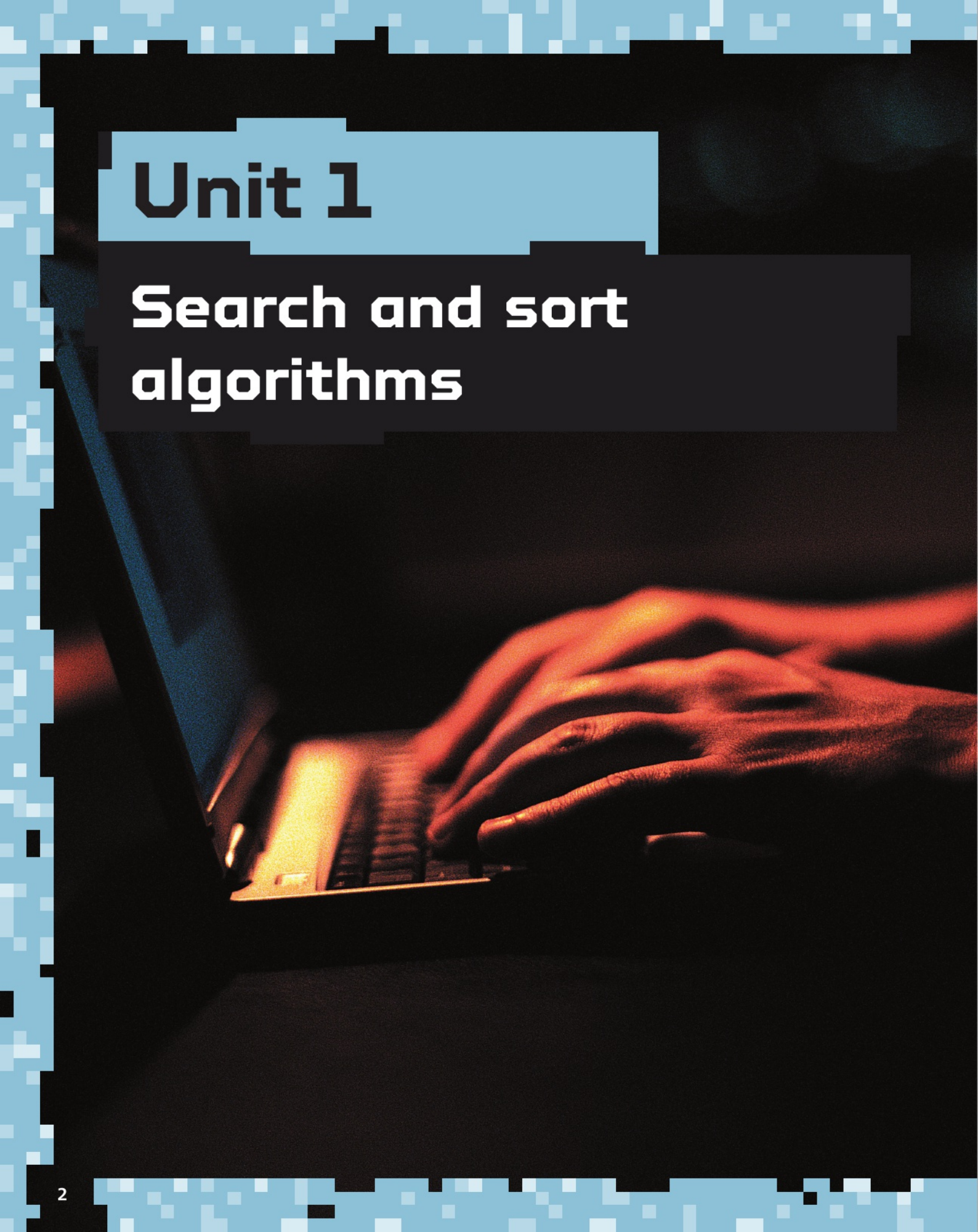
End-of-unit checklist

Checklist at the end of every unit to quickly assess your understanding and progress.



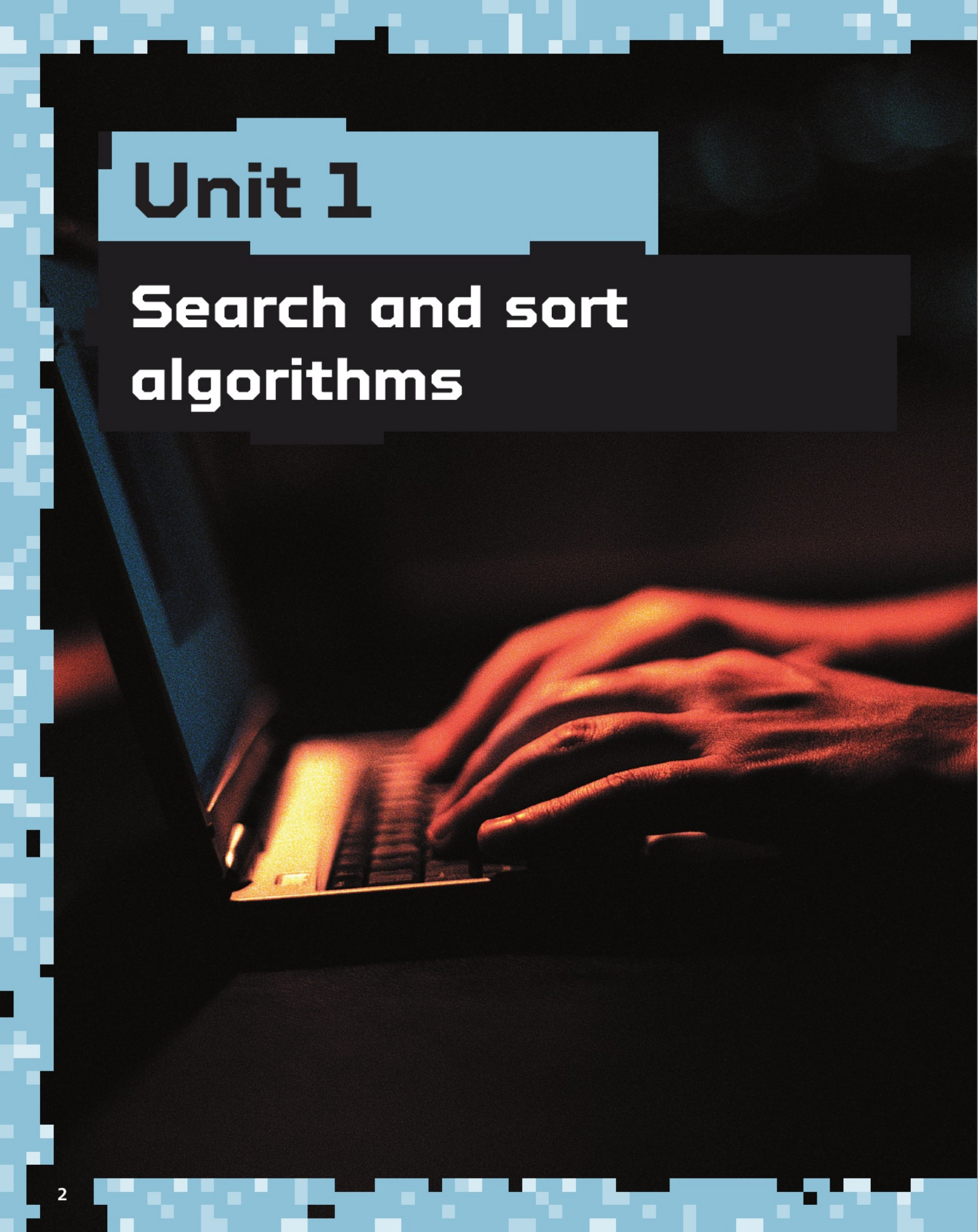
Unit 1

Search and sort algorithms



Unit 1

Search and sort algorithms



You are already familiar with creating simple algorithms and programs, but do you know what abstraction and decomposition are? Do you know the difference between them? In this unit, you will also look at complex search and sorting algorithms and become adept at testing and resolving problems.

Key objectives

1. Practise algorithmic and computational thinking.
2. Know what abstraction and decomposition are.
3. Understand how bubble and merge sorts work.
4. Understand how binary and linear searches work.
5. Know how to check for errors in a program and test an error fix.

By the end of this unit, you will:

- apply algorithmic and computational thinking to a problem
- create a simple search algorithm using pseudocode and Python programming
- create a simple sort algorithm using pseudocode and Python programming
- be able to recommend an appropriate algorithm for a particular problem
- check a program for errors and carry out testing of an error fix.

You are already familiar with creating simple algorithms and programs, but do you know what abstraction and decomposition are? Do you know the difference between them? In this unit, you will also look at complex search and sorting algorithms and become adept at testing and resolving problems.

Key objectives

1. Practise algorithmic and computational thinking.
2. Know what abstraction and decomposition are.
3. Understand how bubble and merge sorts work.
4. Understand how binary and linear searches work.
5. Know how to check for errors in a program and test an error fix.

By the end of this unit, you will:

- apply algorithmic and computational thinking to a problem
- create a simple search algorithm using pseudocode and Python programming
- create a simple sort algorithm using pseudocode and Python programming
- be able to recommend an appropriate algorithm for a particular problem
- check a program for errors and carry out testing of an error fix.

Algorithms and computational thinking

Related topics

- Programming
- Impact of technology on society

Key words

abstraction

algorithmic thinking

computational thinking

decomposition

Learning objectives

1. Understand the terms algorithmic and computational thinking.
2. Understand the terms abstraction and decomposition.
3. Know how to apply abstraction to a simple problem.

What is algorithmic thinking?

As you have seen in Years 7 and 8, an algorithm is a simple set of instructions that follows a logical sequence. **Algorithmic thinking** is when we apply this logical sequence to a problem and come up with a step-by-step solution.

What is computational thinking?

Computational thinking is when we think like a computer scientist. We can analyse a problem and plan a solution that uses computers. Two key elements of computational thinking are:

- **decomposition**
- **abstraction**.

Decomposition and abstraction

- Decomposition is the breaking down of a problem into smaller parts that can be individually solved. For example, to understand how a games console works, we can break it down into parts, such as how the laser reads the disc.
- Abstraction is the process of filtering out any information that is not essential. *For example, to turn on a games console, you need to know how to turn on the power button. You don't need to know about how this supplies power to the components, or how it loads*.

Decomposition helps us break down a problem in order to understand it. Abstraction lets us focus on the most important elements to solve the problem.

Algorithms and computational thinking

Related topics

- Programming
- Impact of technology on society

Key words

abstraction

algorithmic thinking

computational thinking

decomposition

Learning objectives

1. Understand the terms algorithmic and computational thinking.
2. Understand the terms abstraction and decomposition.
3. Know how to apply abstraction to a simple problem.

What is algorithmic thinking?

As you have seen in Years 7 and 8, an algorithm is a simple set of instructions that follows a logical sequence. **Algorithmic thinking** is when we apply this logical sequence to a problem and come up with a step-by-step solution.

What is computational thinking?

Computational thinking is when we think like a computer scientist. We can analyse a problem and plan a solution that uses computers. Two key elements of computational thinking are:

- **decomposition**
- **abstraction**.

Decomposition and abstraction

- Decomposition is the breaking down of a problem into smaller parts that can be individually solved. For example, to understand how a games console works, we can break it down into parts, such as how the laser reads the disc.
- Abstraction is the process of filtering out any information that is not essential. *For example, to turn on a games console, you need to know how to turn on the power button. You don't need to know about how this supplies power to the components, or how it loads*.

Decomposition helps us break down a problem in order to understand it. Abstraction lets us focus on the most important elements to solve the problem.

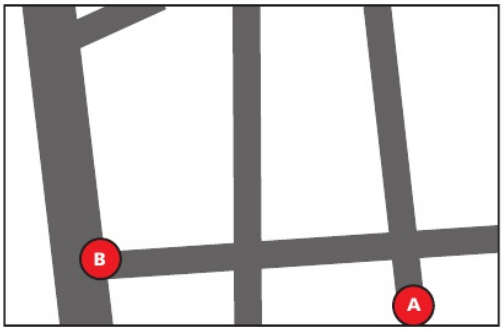
Real-world examples

Lots of jobs might need you to think analytically, but some use computational thinking every day. These include:

- website developer
- software engineer
- database designer
- app or video game developer
- hardware engineer.

Applying abstraction to a problem

Route planning is a good example of a problem that benefits from abstraction. Consider the map below and the route from point A to point B. A set of instructions for this route could be very complicated.



Point A to point B:

- Travel along the highway with the trees until you reach the tall building.
- Turn left and drive until you see the gap between the buildings.
- Turn onto the main road…

These instructions are confusing and difficult to follow. There is too much non-essential information.

Now consider an abstracted version. The buildings, cars and trees have been removed.

The instructions become:

- From Point A, take the first left.
- Turn left at the T-junction.
- Then take the next right.
- Point B is at the end of the road.

These instructions are much clearer because they only contain the essential details.

Map apps use decomposition like this to make directions easy to understand and follow.

Further investigation

- Create your own abstraction of a route you know, such as around your school.
- Make a list of different things a driverless car will need to observe and react to- for example, traffic lights.

Success criteria

- I can define the terms algorithmic and computational thinking.
- I can define the terms abstraction and decomposition.
- I can apply abstraction to a simple navigation problem.

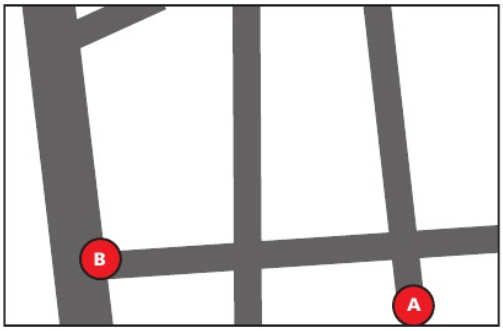
Real-world examples

Lots of jobs might need you to think analytically, but some use computational thinking every day. These include:

- website developer
- software engineer
- database designer
- app or video game developer
- hardware engineer.

Applying abstraction to a problem

Route planning is a good example of a problem that benefits from abstraction. Consider the map below and the route from point A to point B. A set of instructions for this route could be very complicated.



Point A to point B:

- Travel along the highway with the trees until you reach the tall building.
- Turn left and drive until you see the gap between the buildings.
- Turn onto the main road…

These instructions are confusing and difficult to follow. There is too much non-essential information.

Now consider an abstracted version. The buildings, cars and trees have been removed.

The instructions become:

- From Point A, take the first left.
- Turn left at the T-junction.
- Then take the next right.
- Point B is at the end of the road.

These instructions are much clearer because they only contain the essential details.

Map apps use decomposition like this to make directions easy to understand and follow.

Further investigation

- Create your own abstraction of a route you know, such as around your school.
- Make a list of different things a driverless car will need to observe and react to- for example, traffic lights.

Success criteria

- I can define the terms algorithmic and computational thinking.
- I can define the terms abstraction and decomposition.
- I can apply abstraction to a simple navigation problem.

Sorting algorithms

Related topics

- Programming
- Impact of technology on society
- Data analysis
- Logical operators

Key words

algorithm
bubble sort
greater than
less than
order
pass

Learning objectives

1. Understand the purpose of sorting algorithms.
2. Know what a bubble sort is and how it works.
3. Know how to create a simple representation of a bubble sort.

Why do we need sorting algorithms?

Think about the stored data we access in our daily lives. There is probably more than you realise! Consider the following:

- music streaming services
- film and television streaming services
- online shopping sites
- contacts or address books in our smart devices
- messaging and email applications.

The examples above all contain huge amounts of stored data. When we access them, we use sorting to help us find what we want. For example:

- searching for the most recently released music, television or film
- comparing shopping items based on price
- scrolling through a list of contacts based on surname
- looking through emails from particular senders.

All of these examples will use sorting **algorithms**.

What is a bubble sort?

A **bubble sort** is a method of sorting a jumbled sequence of values into **order**. It works in the following way:

1. The first two values are compared and swapped if not in order.
2. The next two values are compared and swapped if not in order.
3. This is repeated until the end of the sequence is reached.

This is known as the first **pass**.

4. The process is then repeated until no values need to be swapped and the sequence is in order.

Sorting algorithms

Related topics

- Programming
- Impact of technology on society
- Data analysis
- Logical operators

Key words

algorithm
bubble sort
greater than
less than
order
pass

Learning objectives

1. Understand the purpose of sorting algorithms.
2. Know what a bubble sort is and how it works.
3. Know how to create a simple representation of a bubble sort.

Why do we need sorting algorithms?

Think about the stored data we access in our daily lives. There is probably more than you realise! Consider the following:

- music streaming services
- film and television streaming services
- online shopping sites
- contacts or address books in our smart devices
- messaging and email applications.

The examples above all contain huge amounts of stored data. When we access them, we use sorting to help us find what we want. For example:

- searching for the most recently released music, television or film
- comparing shopping items based on price
- scrolling through a list of contacts based on surname
- looking through emails from particular senders.

All of these examples will use sorting **algorithms**.

What is a bubble sort?

A **bubble sort** is a method of sorting a jumbled sequence of values into **order**. It works in the following way:

1. The first two values are compared and swapped if not in order.
2. The next two values are compared and swapped if not in order.
3. This is repeated until the end of the sequence is reached.

This is known as the first **pass**.

4. The process is then repeated until no values need to be swapped and the sequence is in order.

Applying a bubble sort

The example below applies a bubble sort to put the sequence 2, 8, 1, 4, 9 into ascending order (smallest number first).

First pass

2, **8**, 1, 4, 9 becomes **2**, **8**, 1, 4, 9 (2 is **less than** 8, so nothing changes)

2, **8**, **1**, 4, 9 becomes 2, **1**, **8**, 1, 9 (8 is **greater than** 1, so they swap)

2, 1, **8**, **4**, 9 becomes 2, 1, **4**, **8**, 9 (8 is greater than 4, so they swap)

2, 1, 4, **8**, **9** becomes 2, 1, 4, **8**, **9** (8 is less than 9, so nothing changes)

Second pass

2, **1**, 4, 8, 9 becomes **1**, **2**, 4, 8, 9 (2 is greater than 1, so they swap)

1, **2**, **4**, 8, 9 becomes 1, **2**, **4**, 8, 9 (2 is less than 4, so nothing changes)

1, 2, **4**, **8**, 9 becomes 1, 2, **4**, **8**, 9 (4 is less than 8, so nothing changes)

1, 2, 4, **8**, **9** becomes 1, 2, 4, **8**, **9** (8 is less than 9, so nothing changes)

Third pass

We can see that the numbers are now in order, but the algorithm needs to check this. In the third pass, no swaps are needed. This tells the algorithm that the data is now sorted: 1, 2, 4, 8, 9.

Real-world examples

The term bubble sort is used because of its similarity to a fizzy drink. Larger bubbles make their way to the top in the same way that the larger values move to the end of the sequence.

Further investigation

- Experiment with using a bubble sort with your own set of values.

Success criteria

- I know why sorting algorithms are needed.
- I know what a bubble sort is and how it works.
- I can create my own simple bubble sort.

Applying a bubble sort

The example below applies a bubble sort to put the sequence 2, 8, 1, 4, 9 into ascending order (smallest number first).

First pass

2, **8**, 1, 4, 9 becomes **2**, **8**, 1, 4, 9 (2 is **less than** 8, so nothing changes)

2, **8**, **1**, 4, 9 becomes 2, **1**, **8**, 1, 9 (8 is **greater than** 1, so they swap)

2, 1, **8**, **4**, 9 becomes 2, 1, **4**, **8**, 9 (8 is greater than 4, so they swap)

2, 1, 4, **8**, **9** becomes 2, 1, 4, **8**, **9** (8 is less than 9, so nothing changes)

Second pass

2, **1**, 4, 8, 9 becomes **1**, **2**, 4, 8, 9 (2 is greater than 1, so they swap)

1, **2**, **4**, 8, 9 becomes 1, **2**, **4**, 8, 9 (2 is less than 4, so nothing changes)

1, 2, **4**, **8**, 9 becomes 1, 2, **4**, **8**, 9 (4 is less than 8, so nothing changes)

1, 2, 4, **8**, **9** becomes 1, 2, 4, **8**, **9** (8 is less than 9, so nothing changes)

Third pass

We can see that the numbers are now in order, but the algorithm needs to check this. In the third pass, no swaps are needed. This tells the algorithm that the data is now sorted: 1, 2, 4, 8, 9.

Real-world examples

The term bubble sort is used because of its similarity to a fizzy drink. Larger bubbles make their way to the top in the same way that the larger values move to the end of the sequence.

Further investigation

- Experiment with using a bubble sort with your own set of values.

Success criteria

- I know why sorting algorithms are needed.
- I know what a bubble sort is and how it works.
- I can create my own simple bubble sort.

Creating a bubble sort program

Related topics

- Impact of technology on society
- Data analysis
- Logical operators

Key words

algorithm
bubble sort
high-level programming
program
pseudocode

Learning objectives

1. Know how to create a bubble sort using pseudocode.
2. Know how to create a bubble sort using Python.

Creating a bubble sort using pseudocode

In Lesson 2, we applied a **bubble sort** to a sequence of numbers in a series of passes.

The example below is a **pseudocode** version of a bubble sort.

```
WHILE list is unsorted DO
FOR n items in list DO
    IF n > n+1
    THEN swap items
    END IF
END FOR
END WHILE
```

This tells the **algorithm** to run until the list of values is sorted.

For each of the items in the list, carry out the instructions below.

If the first value in the list is larger than the following item, swap the items.

Continue until all items have been compared.

End the process.

Real-world advice

In Year 8, you learned that pseudocode is **program**-like code that programmers use to plan programs. It cannot be run by a computer, but a programmer will use it as a starting point to create a version of the code in a **high-level programming** language such as Python, Java or C++.

Creating a bubble sort using Python

Building on the text-based version and the pseudocode version, a bubble sort can be created using the Python programming language.

Creating a bubble sort program

Related topics

- Impact of technology on society
- Data analysis
- Logical operators

Key words

algorithm
bubble sort
high-level programming
program
pseudocode

Learning objectives

1. Know how to create a bubble sort using pseudocode.
2. Know how to create a bubble sort using Python.

Creating a bubble sort using pseudocode

In Lesson 2, we applied a **bubble sort** to a sequence of numbers in a series of passes.

The example below is a **pseudocode** version of a bubble sort.

```
WHILE list is unsorted DO
FOR n items in list DO
    IF n > n+1
    THEN swap items
    END IF
END FOR
END WHILE
```

This tells the **algorithm** to run until the list of values is sorted.

For each of the items in the list, carry out the instructions below.

If the first value in the list is larger than the following item, swap the items.

Continue until all items have been compared.

End the process.

Real-world advice

In Year 8, you learned that pseudocode is **program**-like code that programmers use to plan programs. It cannot be run by a computer, but a programmer will use it as a starting point to create a version of the code in a **high-level programming** language such as Python, Java or C++.

Creating a bubble sort using Python

Building on the text-based version and the pseudocode version, a bubble sort can be created using the Python programming language.

```
#Python Bubble Sort

#This creates a sub program called bubbleSort
def bubbleSort(list):
    for number in range(len(list)-1,0,-1):
        #set a variable to test if the list changes in this iteration
            listChanged = False
            for n in range(number):
                if list[n]>list[n+1]:
                    temp = list[n]
                    list[n] = list[n+1]
                    list[n+1] = temp
                    #set variable to indicate list was changed
                    listChanged = True
            #If the list was not changed then further checking can stop
                if not listChanged:
                    break

#This is the list to be sorted
list = [7,3,8,5,6,4]
#The bubbleSort is applied to the list
bubbleSort(list)
print(list)
```

This program would generate the following output:

```
[3, 4, 5, 6, 7, 8]
>>>
```

Real-world examples

In the early 2000s, music streaming services such as Apple iTunes, Pandora and Myspace introduced digital music streaming. Before that, digital music was accessed from locally stored audio files (such as mp3).

Now, music streaming platforms provide unlimited access to millions of songs. Search algorithms have to search tens of millions of files effectively.

Further investigation

- You can experiment with the Python program shown. Try changing the following:
  - the names of variables
  - the list to be sorted
  - the length of the list.

Success criteria

- I know how to create a simple pseudocode bubble sort algorithm.
- I know how to create a Python bubble sort program and apply it to a short list of numbers.

```
#Python Bubble Sort

#This creates a sub program called bubbleSort
def bubbleSort(list):
    for number in range(len(list)-1,0,-1):
        #set a variable to test if the list changes in this iteration
            listChanged = False
            for n in range(number):
                if list[n]>list[n+1]:
                    temp = list[n]
                    list[n] = list[n+1]
                    list[n+1] = temp
                    #set variable to indicate list was changed
                    listChanged = True
            #If the list was not changed then further checking can stop
                if not listChanged:
                    break

#This is the list to be sorted
list = [7,3,8,5,6,4]
#The bubbleSort is applied to the list
bubbleSort(list)
print(list)
```

This program would generate the following output:

```
[3, 4, 5, 6, 7, 8]
>>>
```

Real-world examples

In the early 2000s, music streaming services such as Apple iTunes, Pandora and Myspace introduced digital music streaming. Before that, digital music was accessed from locally stored audio files (such as mp3).

Now, music streaming platforms provide unlimited access to millions of songs. Search algorithms have to search tens of millions of files effectively.

Further investigation

- You can experiment with the Python program shown. Try changing the following:
  - the names of variables
  - the list to be sorted
  - the length of the list.

Success criteria

- I know how to create a simple pseudocode bubble sort algorithm.
- I know how to create a Python bubble sort program and apply it to a short list of numbers.

Creating a merge sort program

Related topics

- Impact of technology on society
- Data analysis
- Logical operators

Key words

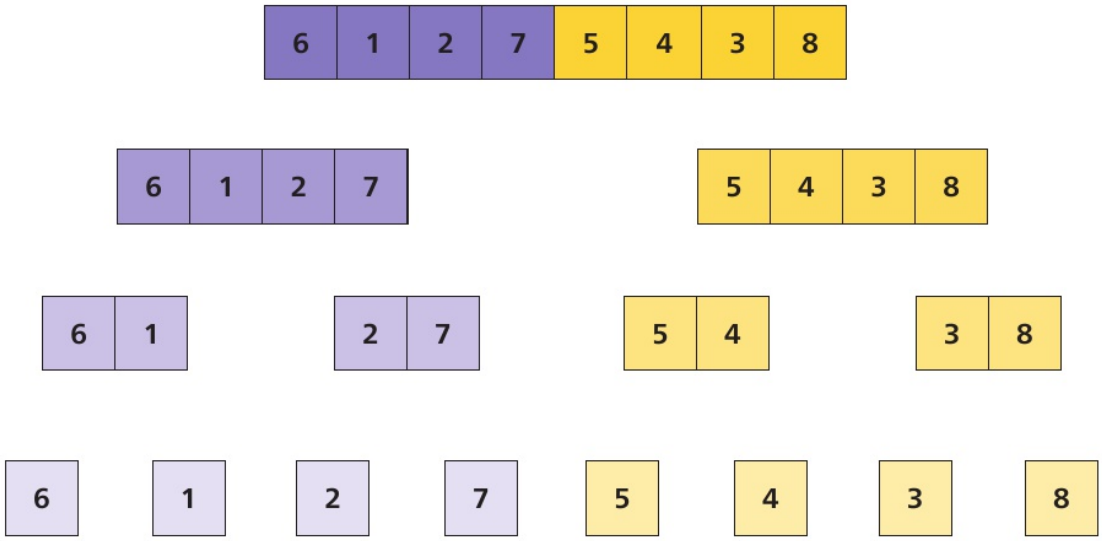
algorithm
merge sort
program
pseudocode
subprogram

Learning objectives

1. Understand what a merge sort is and how it works.
2. Know how to create a merge sort using pseudocode.
3. Experiment with a merge sort using Python.

What is a merge sort?

A **merge sort** divides a jumbled sequence in half again and again until only single values remain. These values are then compared against each other and merged back together in the correct order. Look at this example:



The sequence has been broken down into single values. Next, the **algorithm** compares pairs of values, just like in a bubble sort. The first two values are 6 and 1. 6 is greater than 1, so they swap. This happens to each pair until they are sorted like this: