

# Chapter 7: Algorithms

In this chapter you will learn:

- ✧ what is meant by computational thinking
- ✧ what is meant by decomposition
- ✧ what is meant by abstraction
- ✧ what is meant by an algorithm
- ✧ how to create an algorithm using a flow chart
- ✧ how to create an algorithm using pseudocode
- ✧ what is meant by a searching algorithm
- ✧ what is meant by a sorting algorithm

## What is computational thinking?

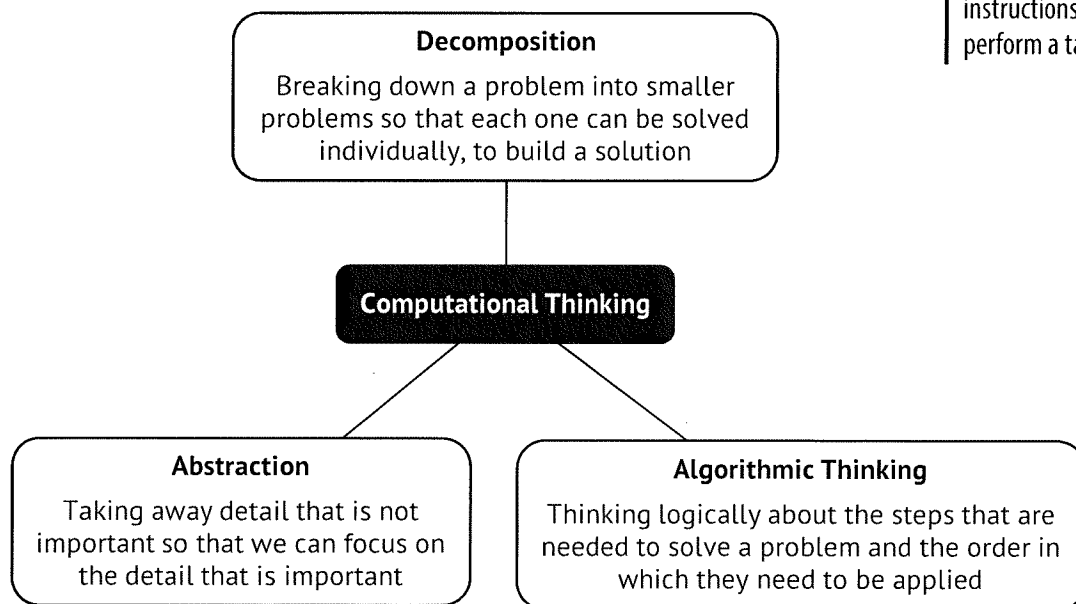
**OCR specification reference:**

- ☑ computational thinking:
  - abstraction
  - decomposition
  - algorithmic thinking

**Computational thinking** is not learning to think like a computer, but learning how to look at problems and break them down in a logical way in order to find a solution. To do this we can use abstraction, decomposition and **algorithmic** thinking.

**Computational thinking** – breaking problems down in a logical way in order to find a solution

**Algorithm** – a sequence of instructions that solve a problem or perform a task



When we apply all three of these methods of computational thinking, we are able to effectively take a complex problem and build an efficient solution for it.

Computational thinking is something that we do naturally in many parts of our lives. We naturally go through the steps involved in the many of the routine things we do in our lives, from making our meals to crossing the road. When we need to make a decision about something, such as which television programme to watch, we abstract out the detail, such as how the television works and displays programmes, and focus on what kind of programme we feel like watching and what might be on the television at the moment.

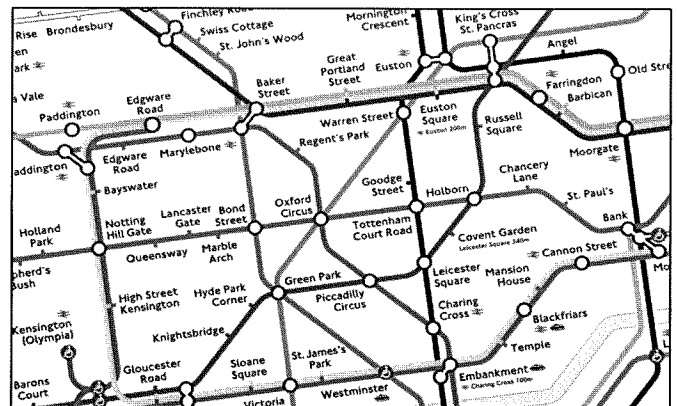
These three methods are skills that are very important for us to develop in order to become effective Computer Scientists. The best way for us to do this is to practise the skills.

Let's take a simple problem of entering exam marks for a class, storing them in a file, then outputting the average mark for the exam. We can plan an algorithm to solve this problem by using a **flow chart**. If we decompose our problem, we can see what the components parts would need to be:

**Flow chart** – a diagrammatic way to represent an algorithm

- An input to enter the name of each student
- A data store to store the student's name
- An input to enter the exam mark for each student
- A data store to store the exam mark
- A process to store the student name and exam mark in a file
- A process to add each mark to a running total in order to calculate the average mark
- A data store to store the running total
- A process to count the number of marks input
- A data store to store the number of marks input
- A process to divide the running total by the class size to get the average mark
- An output to show the average mark for the exam

We have now decomposed our problem into a series of small problems that we can use to build our algorithm. We have abstracted out any detail that is not important to our problem, such as what the pass mark was for the exam, or what exam marks a student has achieved previously. That level of detail is not important for the problem we are looking to solve.



Metro maps are a great example of abstraction: we only want to know which station is connected to which; we don't care about their actual relative positions or the exact curvature of the rail lines.

## How can we create an algorithm?


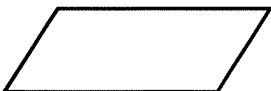

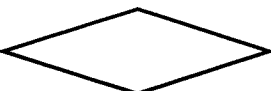


### OCR specification reference:

- ☒ how to produce algorithms using:
  - pseudocode
  - flow diagrams
- ☒ interpret, correct or complete algorithms

An algorithm is a sequence of steps or instructions that are carried out to solve a problem or perform a task. We can create algorithms using programming languages, but it is better if we plan our algorithm first. We can use two methods to plan an algorithm: flow charts and **pseudocode**.

**Pseudocode** – a way of designing a program in programming-type statements that are not specific to any programming language

A flow chart is a diagram that we can create to plan and demonstrate the flow of data in a solution. When creating flow charts, we use a series of symbols to represent the different elements of the program; these are:

Symbol	Explanation
	<i>Start/stop</i> – used to show where the beginning and end of our flow chart is.
	<i>Input/output</i> – used to demonstrate where our flow chart will take in an input, or provide us with an output.
	<i>Process</i> – used to show any processes or calculations that are happening in our flow chart.
	<i>Decision</i> – used to demonstrate a decision or choice that needs to be taken in our flow chart. It will have a 'yes' and a 'no' path from the decision symbol.
	<i>Sub-process</i> – used to show where a sub-process will occur in our algorithm.
	<i>Arrow</i> – used to show the flow of data through our flow chart.

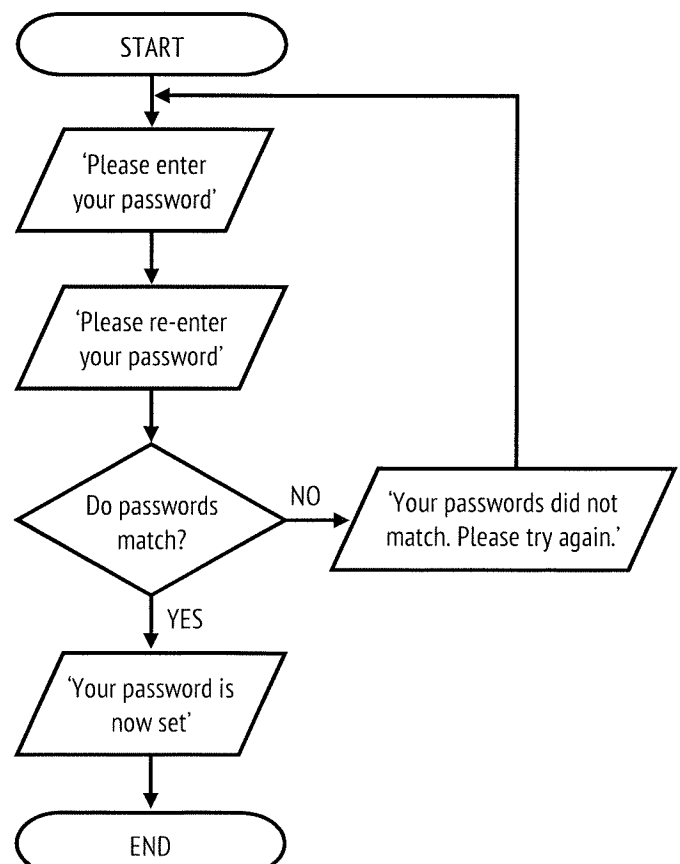
When we create a flow chart, we combine these symbols together in the order in which we want instructions to occur. We then join the symbols together with an arrow to show the direction of the flow.

Let's take a simple password verification program. This is a program that will ask a user to set a password and re-enter it to verify they originally entered the password they were meant to. We want our solution to allow our user to do the following:

- Input their choice of password
- Re-enter their password
- Make sure the first password entered is the same as the second password entered
- Output a message to say whether the password has been successfully set or not
- Keep asking the user to set their password until they are successful

We have decomposed our problem.

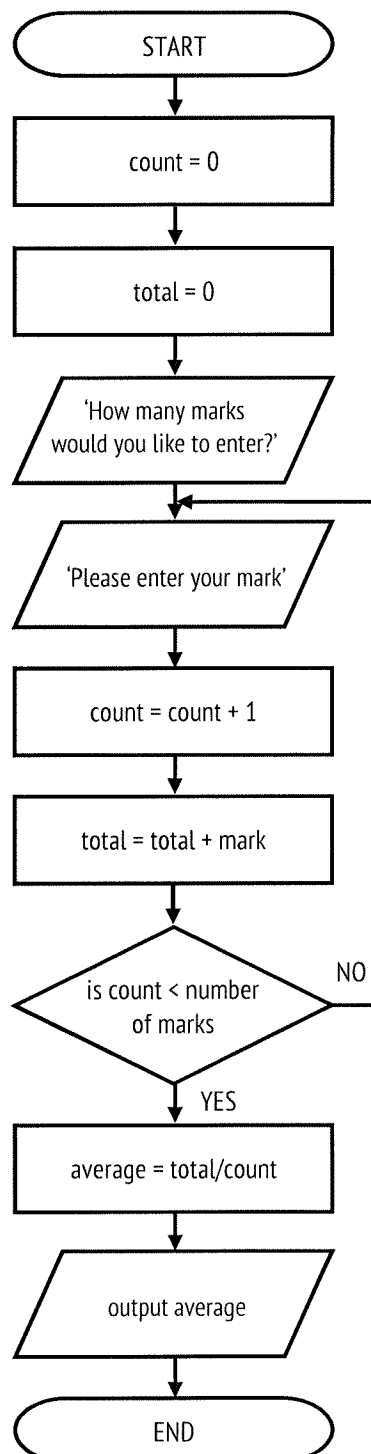
We can now create a flow chart to represent a design for our solution:



Let's create another flow chart for part of our earlier problem. We want to create a program that will calculate the average mark for an exam. We want our solution to do the following:

- Allow the user to input the number of marks they want to enter
- Keep a count of the marks entered
- Keep a total of the marks entered
- Allow the correct number of marks to be entered
- Calculate the average mark
- Output the average mark

We have decomposed our problem. We can now create a flow chart to represent a design for our solution:



We can also represent a design for an algorithm using pseudocode. Pseudocode is a way of writing a program in programming-type statements that are not specific to any programming language. There is no set standard for pseudocode, but OCR have provided a guide to their standard for pseudocode on their website, so this is the standard we will use.

When writing programs in pseudocode we can use the following statements:

Statement	Description
<code>variable = "data"</code>	This allows us to declare a variable and assign data to it.
<code>variable = input("user prompt")</code>	This allows the user to input data and assign it to a variable.
<code>print(variable)</code>	This allows us to provide a user with an output that gets printed to the screen.
<code>for i = 0 to 3   print(variable) next i</code>	This allows us to create a counting loop so that we can perform a set of instructions a set number of times.
<code>while variable == false   variable = input("user prompt") endwhile</code>	This allows us to create a condition loop where the condition is checked at the start of the loop.
<code>do   variable = input("user prompt") until variable == true</code>	This allows us to create a condition loop where the condition is checked at the end of the loop.
<code>if variable == 1 then   print(1) elseif variable == then   print(2) else   print(0) endif</code>	This allows us to create selection in our program.  We can add multiple selection statements through the use of elseif.
<code>switch variable:   case 1:     print(1)   case 2:     print(2)   default:     print(0) endswitch</code>	This allows us to create selection in our programming using a set number of options.  We can add a default option to account for any inputs that do not match an option.
<code>function double(parameter)   return parameter *2 endfunction</code>  calling:  <code>variable = double(argument)</code>	This allows us to store a set of instructions inside a function.  We can then call the function and it will return a value.
<code>procedure name(parameter)   instruction 1   instruction 2 endprocedure</code>  calling:  <code>name(argument)</code>	This allows us to store a set of instructions inside a procedure.  We can then call the procedure any time we want to carry out the set of instructions.  These differ from functions, as functions return a value.

Statement	Description
<pre> array name[3]  array name[3,5]  name[0] = "entry1" name[1] = "entry2" name[0, 0] = "entry1" name[0, 1] = "entry2"  print(name[1]) print(name[0, 1]) </pre>	<p>This allows us to create arrays. The first is a one-dimensional array, the second is a two-dimensional array.</p> <p>We can then assign, amend and extract values from each element in the array.</p>
<pre> file = openRead("text.txt") f = myfile.readline() file.close()  file = openWrite("text.txt") f = myfile.writeline() file.close() </pre>	<p>These are the statements that are needed for file handling. They allow us to open the file in read mode, read from the file, and then close it.</p> <p>Or if you want to write to a file you can open the file in write mode, write to the file, and then close the file afterwards.</p>
<pre> variable.length  variable.substring(start, noOfCharacters) </pre>	<p>These are string manipulation statements. They allow us to find out the length of a string.</p> <p>They also allow us to extract sections of characters out of a string.</p>

We can use this pseudocode standard to write a program to represent our first flow chart, the password verification program:

```

match = false
while match == false
    password1 = input("Please enter your password")
    password2 = input("Please re-enter your password")
    if password1 == password2
        print("Your password has been set")
        match = true
    else
        print("Your passwords do not match")
        match = false
    endif
endwhile

```

**Discussion point:** How would you write the pseudocode for the second flow chart we created?

There are two particular types of algorithm that you need to know: these are searching algorithms and sorting algorithms. **Searching** and **sorting algorithms** are used on data sets to make them simpler to understand and easier to access.

## What is a searching algorithm?

### OCR specification reference:

- ☑ standard searching algorithms:
  - binary search
  - linear search

A searching algorithm is one that is designed to look through a data set and find a particular item of data. When a data set is very large, running into hundreds and thousands of entries, manually trying to find a particular item of data can be time-consuming. A searching algorithm will perform this function for us. There are two types of searching algorithm that you need to know: these are linear search and binary search.

**Searching algorithm** – the step-by-step procedure used to find an item of data in a data set

### Linear search

A linear search is a simple sequential search of a data set. The algorithm will start at the beginning of the data set and moves through each data item one by one. It will do this until it finds the data item it has been set to look for, or it reaches the end of the data set without finding a matching data item.

If we had a data set of numbers and we wanted to find a particular number in the data set we could use a linear search; for example:

We have the following data set of numbers.

10	25	31	15	85	69	75	21	19	6
----	----	----	----	----	----	----	----	----	---

We want to search the data set to see whether the number 85 appears in it. We start by comparing 85 to the first number in the data set.

Is  $85 = 10$ ?

10	25	31	15	85	69	75	21	19	6
----	----	----	----	----	----	----	----	----	---

85 is not equal to 10, so the algorithm will move on to the next item.

Is  $85 = 25$ ?

10	25	31	15	85	69	75	21	19	6
----	----	----	----	----	----	----	----	----	---

85 is not equal to 25, so the algorithm will move on to the next item.

Is  $85 = 31$ ?

10	25	31	15	85	69	75	21	19	6
----	----	----	----	----	----	----	----	----	---

85 is not equal to 31, so the algorithm will move on to the next item.

Is  $85 = 15$ ?

10	25	31	15	85	69	75	21	19	6
----	----	----	----	----	----	----	----	----	---

85 is not equal to 15, so the algorithm will move on to the next item.

Is  $85 = 85$ ?

10	25	31	15	85	69	75	21	19	6
----	----	----	----	----	----	----	----	----	---

Yes, it is! The algorithm will stop here as the data item 85 is found.



Searching on a computer is very different to how humans search for things.

We can write an algorithm in pseudocode to represent a linear search for the data set:

```
position = 0
list = [10, 25, 31, 15, 85, 69, 75, 21, 19, 6]
length = list.length
number = input("What number would you like to find?")
while position < length AND list[position] != number
    position = position + 1
endwhile
if position >= length then
    print("That number is not in the list")
else
    print("We have found your number!")
endif
```

## Binary search

A binary search is another type of searching algorithm. We mostly use a binary searching algorithm when we have a list of data that is in order. A binary searching algorithm works by dividing a list in half and looking at the item in the middle. If when the list is split there is an even number of items on each side, we look at the first item on the right-hand side.

If we order the list of number we used previously, we can use a binary search to find a particular number.

We had the following set of numbers:

10	25	31	15	85	69	75	21	19	6
----	----	----	----	----	----	----	----	----	---

If we order them, we will have:

6	10	15	19	21	25	31	69	75	85
---	----	----	----	----	----	----	----	----	----

We can search the data again to see whether the number 85 appears in it. We start by dividing the data set in half:

6	10	15	19	21	25	31	69	75	85
---	----	----	----	----	----	----	----	----	----

We have an even number of items, so we look at the data in the first position at the right-hand side. We check to see whether 85 is equal to this number first. 85 is not equal to 25, so we have not found it. We then check whether 85 is greater than 25. It is, so we can discard the list on the left-hand side and only use the one on the right.

25	31	69	75	85
----	----	----	----	----

We then divide this list in half:

25	31	69	75	85
----	----	----	----	----

This time we do have an odd number and therefore have a central number. We check this first to see whether we have found 85. 85 is not equal to 69, so we have not found it. We then check whether 85 is greater than 69. It is, so we can discard the list to the left-hand side and only use the one on the right.

75	85
----	----

We then divide this list in half:

75	85
----	----

We have an even number of items, so we look at the data in the first position at the right-hand side. We check to see whether 85 is equal to this number first. It is! So we have found 85 in the list of data.



In a binary search, the item at the start of the list is called the lower bound, the item at the end of the list is called the upper bound, and the item in the middle of the list is called the midpoint. The upper bound and lower bound will move in the list as we keep dividing it in half.

We can write an algorithm in pseudocode to represent a binary search for the data set:

```
list = [6, 10, 15, 19, 21, 25, 31, 69, 75, 85]
length = list.length
number = input("What number would you like to find?")
lowerBound = 0
upperBound = length - 1
match = false
while match == false AND lowerBound != upperBound
    midPoint = round((lowerBound + upperBound)/2)
    if list[midPoint] == number then
        print("We have found your number!")
        match = true
    elseif list[midPoint] < number then
        lowerBound = midPoint + 1
    else
        upperBound = midPoint - 1
    endif
endwhile
```

For the exam, you need to be able to recognise a searching or sorting algorithm from a set of code. You may be given the algorithm and have to describe what it does.

## What is a sorting algorithm?

### OCR specification reference:

- ☒ standard sorting algorithms:
  - bubble sort
  - merge sort
  - insertion sort

A sorting algorithm is one that is designed to sort a set of data into order (either increasing or decreasing; we will always use increasing in the examples). There are various ways in which we can sort a data set into order; the three methods we need to know are a bubble sort, a merge sort and an insertion sort.

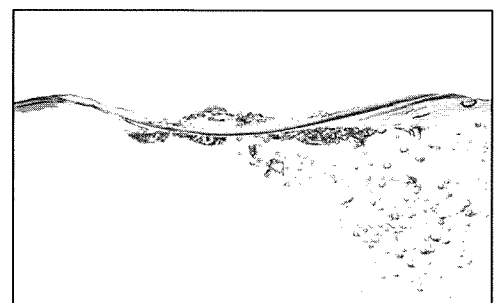
**Sorting algorithm** – the step-by-step procedure used to arrange a data set into an order

### Bubble sort

A bubble sort is the simplest sorting algorithm to understand. A bubble sort starts at the beginning of a list and first of all checks the first item against the second item.

If the first item is greater than the second item, the algorithm knows they are not in order, so it swaps them. If the first item is less than the second item, it just leaves them as they are.

It then moves onto the second item and checks this against the third item in the list. If it is greater than the third item, it swaps them, otherwise it leaves them as they are, and so on. It will go all the way through the data set and then start again at the beginning to do the same thing again. This is called *passing through* the data. It will repeatedly pass through the data until it makes no changes to the data; it then knows they are in order.



This sort is called a bubble sort as the biggest numbers 'bubble' to the end of the array.

We can use part of our original data set:

10	25	31	15	85	69
----	----	----	----	----	----

We first of all look to see whether 10 is greater than 25. It is not, so they stay as they are.

10	25	31	15	85	69
----	----	----	----	----	----

We then look to see whether 25 is greater than 31. It is not, so they stay as they are.

10	25	31	15	85	69
----	----	----	----	----	----

We then look to see whether 31 is greater than 15. It is, so we swap them.

10	25	15	31	85	69
----	----	----	----	----	----

We then look to see whether 31 is greater than 85. It is not, so they stay as they are.

10	25	15	31	85	69
----	----	----	----	----	----

We then look to see whether 85 is greater than 69. It is, so we swap them.

10	25	15	31	69	85
----	----	----	----	----	----

This completes our first pass through the data. We can now begin a second pass.

10	25	15	31	69	85
----	----	----	----	----	----

10	15	25	31	69	85
----	----	----	----	----	----

10	15	25	31	69	85
----	----	----	----	----	----

10	15	25	31	69	85
----	----	----	----	----	----

10	15	25	31	69	85
----	----	----	----	----	----

In that pass the numbers 15 and 25 were swapped. Even though we can see that the list is now in order, the algorithm does not yet know this. It will make one more pass of the data where it will not need to make any changes, so it will then recognise the data is sorted.

We can write an algorithm in pseudocode to represent a bubble sort:

```
list = [10, 25, 15, 31, 69, 85]
length = list.length - 1
swapped = true
while swapped == true
    swapCount = 0
    for n = 0 to length - 2
        if list[n] > list[n+1] then
            swap(list[n], list[n+1])
            swapCount = swapCount + 1
        endif
    next n
    if swapCount == 0
        swapped = false
    endif
endwhile
```

## Merge sort

A merge sort is an example of a Computer Science technique called 'divide and conquer'. It can be a more efficient sorting algorithm as it makes the lists smaller, so they become easier to sort.

A merge sort works by dividing a list in half repeatedly, until it has a set of lists that have one item in them. It then merges together each list until it has ordered the whole list again.

We can use our original list.

10	25	31	15	85	69	75	21	19	6
10	25	31	15	85	69	75	21	19	6
10	25	31	15	85	69	75	21	19	6
10	25	31	15	85	69	75	21	19	6
10	25	15	31	85	69	75	19	21	6
10	25	15	31	85	6	19	21	69	75
6	10	15	19	21	25	31	69	75	85

A merge sort is normally the quickest way of sorting a list. However, it is very difficult to code so you are highly unlikely to ever need to know how to recognise this in an exam. You mainly need to know the process of how a merge sort works.

## Insertion sort

An insertion sort is used to sort a data set into order by looking at each item in turn and placing it in the correct order in the data set. It starts with the second item in a list and looks at whether all the items before it are less than it. If they are, it leaves the item where it is. If they aren't, it places the item in the correct place and moves on to the next item, until it gets to the end of the list.

We can use our original data set:

10	25	31	15	85	69	75	21	19	6
----	----	----	----	----	----	----	----	----	---

25 is greater than anything before it, so it stays where it is.

10	25	31	15	85	69	75	21	19	6
----	----	----	----	----	----	----	----	----	---

31 is greater than anything before it, so it stays where it is.

10	25	31	15	85	69	75	21	19	6
----	----	----	----	----	----	----	----	----	---

15 is not greater than everything before it, so it is moved to the correct place.

10	15	25	31	85	69	75	21	19	6
----	----	----	----	----	----	----	----	----	---

85 is greater than everything before it, so it stays where it is.

10	15	25	31	85	69	75	21	19	6
----	----	----	----	----	----	----	----	----	---

69 is not greater than everything before it, so it is moved to the correct place.

10	15	25	31	69	85	75	21	19	6
----	----	----	----	----	----	----	----	----	---

75 is not greater than everything before it, so it is moved to the correct place.

10	15	25	31	69	75	85	21	19	6
----	----	----	----	----	----	----	----	----	---

21 is not greater than everything before it, so it is moved to the correct place.

10	15	21	25	31	69	75	85	19	6
----	----	----	----	----	----	----	----	----	---

19 is not greater than everything before it, so it is moved to the correct place.

10	15	19	21	25	31	69	75	85	6
----	----	----	----	----	----	----	----	----	---

6 is not greater than everything before it, so it is moved to the correct place.

6	10	15	19	21	25	31	69	75	85
---	----	----	----	----	----	----	----	----	----

Our list is now in order!

An insertion sort is quicker than a bubble sort but slower than a merge sort. It is easier to code than a merge sort, but not as easy to code as a bubble sort. It is very quick to add a new item of data into the correct place in a sorted list using an insertion sort.

For the exam, you need to be able to recognise the logic of a searching or sorting algorithm. You may be given the algorithm and have to describe what it does. You may be given the algorithm and have to show how you are using it to find an item of data. You may be given the algorithm with an error, or incomplete, and you have to correct the algorithm or complete it. Therefore, you should make sure that you are very familiar with the look and logic of all five of these searching and sorting algorithms. They are a key part of Computer Science!



## Chapter Summary

- Computational thinking is learning to break down problems in a logical way in order to build a solution. There are three main methods for computational thinking: abstraction, decomposition and algorithmic thinking.
- An algorithm is a sequence of steps or instructions that are carried out to solve a problem or perform a task. We can use two methods to plan an algorithm: flow charts and pseudocode.
- A searching algorithm is one that is designed to look through a data set and find a particular item of data. There are two types of searching algorithm that you need to know: these are linear search and binary search.
- A linear search is a simple sequential search of a data set. A binary searching algorithm works by repeatedly dividing a list in half until it finds the item of data.
- A sorting algorithm is one that is designed to sort a set of data into order. The three methods we need to know are a bubble sort, a merge sort and an insertion sort.
- A bubble sort starts at the beginning of a list and compares each item to find the greater, and swaps them if this is the case. A merge sort works by dividing a list in half repeatedly, until it has a set of lists that have one item in them. It then merges together each list until it has ordered the whole list again. An insertion sort is used to sort a data set into order by looking at each item in turn and placing it in the correct order in the data set.

## Practice Questions

1. State what is meant by the term 'computational thinking'. [1]
2. Explain what is meant by the term 'abstraction'. [2]
3. Consider the following set of numbers:  
3, 24, 1, 15, 65, 87, 2, 19  
Show that a binary search can be used to find the number 65. [3]
4. Explain two circumstances in which a linear search might be quicker than a binary search. [2]
5. Using an array of numbers, write an algorithm that swaps the first element with the last element. [5]