

# **Unit 8**

# Algorithms

Algorithm noun

1. a list of rules to follow in order to solve a problem.

#### NAME \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

#### CLASS \_\_\_\_\_\_\_\_\_\_\_\_\_

##### COMPUTATIONAL THINKING

Computers can be used to help us solve problems. However, before a problem can be tackled, the problem itself and the ways in which it could be solved need to be understood. Computational thinking allows us to take a complex problem, understand what the problem is and develop possible solutions. We can then present these solutions in a way that a computer, a human, or both, can understand.

There are **four** key techniques (cornerstones) to **Computational Thinking**. Complete the tasks found on [http://games.thinkingmyself.com](http://games.thinkingmyself.com/) and give definitions of the key techniques below.

|  |
| --- |
| **Decomposition** |
|  |
|  |
|  |
|  |





|  |
| --- |
| **Pattern Recognition** |
|  |
|  |
|  |
|  |

|  |
| --- |
| **Abstraction** |
|  |
|  |
|  |
|  |



|  |
| --- |
| **Algorithmic Thinking** |
|  |
|  |
|  |
|  |



Finally, these simple steps or rules are used to program a computer to help solve the complex problem in the best way. Thinking computationally is not programming. It is not even thinking like a computer, as computers do not, and cannot, think. Simply put, programming tells a computer what to do and how to do it. Computational thinking enables you to work out exactly what to tell the computer to do.

##### TASK 1

You are meeting up with a group of friends this afternoon. All of you like different things, but you have been put in charge of finding something to do that will keep most of your friends happy. Think of some questions could you consider to help you come up with a plan?

|  |
| --- |
| Possible questions to consider |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |

##### TASK 2

Look at the videogame screen to the right. What do you need to consider in order to complete the level as efficiently as possible?

|  |
| --- |
|  |
|  |
|  |
|  |
|  |
|  |
|  |

From these details, you can work out a strategy for completing the level in the most efficient way

|  |
| --- |
|  |
|  |
|  |
|  |
|  |
|  |

##### DECOMPOSITION TASK 1

How would you decompose the problem of making a sandwich? Think of questions you need to know the answers to in order to make a sandwich for someone else.

|  |
| --- |
|  |
|  |
|  |
|  |
|  |
|  |
|  |

When we **decompose** a complex problem we often find **patterns** among the smaller problems we create. The patterns are similarities or characteristics that some of the problems share.

##### PATTERN RECOGNITION TASK 1

Draw a dog in the space below

|  |
| --- |
|  |

|  |
| --- |
| What common characteristics do dogs have that help us recognise if it is a dog? |
|  |
|  |
|  |
|  |
|  |
|  |

**PATTERN RECOGNITION TASK 2**

The classic Sudoku game involves a grid of 81 squares. The grid is divided into nine blocks, each containing nine squares. The rules of the game are simple: each of the nine blocks has to contain all the numbers 1-9 within its squares. Each number can only appear once in a row, column or box.

Use your pattern recognition skills to solve the puzzle!

Easy



Tricky

**ABSTRACTION TASK 1**

In order to be able to draw a dog we need to put its characteristics into **two** groups:

* General details – common to all dogs
* Specific details – specific to certain individual dogs

Complete the table below, using abstraction, to filter out general details from specific details. Once we have don’t this we can build a basic idea of what a dog basically looks like. Once we know what a dog basically looks like we can **describe** how to draw a basic dog.

|  |  |
| --- | --- |
| **General Characteristics** | **Specific Characteristics** |
|  |  |

**ABSTRACTION TASK 2**

In order to be able to calculate the cost of a car journey, we need to be able to identify the factors involved. Some factors we will know before the car journey and some we won’t. Because we are trying to find an approximate cost, we are going to have to do some abstraction.

Complete the table below to show factors that will and won’t affect the way we calculate the cost of the journey.

|  |  |
| --- | --- |
| **Relevant factors** | **Irrelevant Factors**  |
|  |  |

**ALGORITHMIC THINKING – PSEUDOCODE TASK 1**

In an algorithm, each instruction is identified and the order in which they should be carried out is planned. Algorithms are often used as a starting point for creating a computer program, and they are sometimes written as a **flow diagram** or in **pseudocode**.

**Pseudocode** is not a programming language, it is a simple way of describing a set of instructions that does not have to use specific syntax. Writing in pseudocode is like writing in a programming language. Each step of the algorithm is written on a line of its own in sequence. Usually, instructions are written in uppercase, variables in lowercase and messages in sentence case.

Planning a program that asks people what the best subject they take is, would look like this in pseudocode. Explain what each step of the code does below

|  |  |
| --- | --- |
| REPEAT |  |
|  OUTPUT 'What is the best subject?' |  |
|  INPUT user inputs best subject |  |
|  STORE the input in the answer variable |  |
|  IF answer = 'Computer Science' THEN |  |
|  OUTPUT 'Of course it is!' |  |
|  ELSE |  |
|  OUTPUT 'Try again!' |  |
| UNTIL answer = 'Computer Science' |  |

**ALGORITHMIC THINKING PSEUDOCODE TASK 2**

Using the rules set in the last example, write some pseudocode so that the computer asks a true or false question. If the user answers true then a congratulations message is shown, but if the answer is false output a message to say they were wrong.

|  |
| --- |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |

**ALGORITHMIC THINKING – FLOW DIAGRAM TASK 1**

A **flow diagram** is a diagram that represents a set of instructions. Flow diagrams normally use standard symbols to represent the different instructions. There are **few** **rules** about the level of detail needed in a flow diagram. Sometimes flow diagrams are broken down into many steps to provide a lot of detail about exactly what is happening. Sometimes they are simplified so that a number of steps occur in just one step.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Name** | **Start / End** | **Process** | **Decision** | **Input / Output** | **Connector** |
| **Symbol** |  |  |  |  |  |
| **Usage** | The beginning and end points in the sequence | An instruction of command | A decision, either yes or no | An input is data received by the computer. Output is data sent from a computer | A jump from one point to another |
| *Arrows are used to connect the symbols. The direction of the arrow shows the flow of the instructions* |

Using the correct symbols above, draw a flow chart to show the algorithm from PSEUDOCODE TASK 1

|  |
| --- |
|  |

**ALGORITHMIC THINKING – FLOW DIAGRAM TASK 2**

Draw a flow diagram for checking a password is more than 7 characters long and different from a username. If either is incorrect it will ask the user to input the username and password again.

|  |
| --- |
|  |

##### SEARCHING ALGORITHMS

We often need to find **one** item of **data** amongst thousands, millions or more. For example, you might need to find someone’s phone number on your phone, or a name on Facebook. Therefore **searching algorithms** are important. Without them you would have to look at each item of data – each phone number or business address – individually, to see whether it is what you are looking for. In a large set of data, it will take a long time to do this. Instead, a searching algorithm can be used to help find the item of data you are looking for. There are many different types of searching algorithms. The two you need to remember are: **Linear** **search** and **Binary** **search.**

**LINEAR SEARCH**

A linear search will begin at the first item in a list and will search through each item in turn until it reaches the correct item. It will then output the item. If it reaches the end of the list before finding the correct item, a message will be output to say that the item is not found.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 34 | 6 | 23 | 8 | 56 | 45 | 67 | 29 | 54 | 12 |

Use a linear search to find the number 56 from the list above. How many items would the algorithm check before it found the number?

|  |
| --- |
|  |

Draw a flow diagram to show how linear searches work below

|  |
| --- |
|  |

**BINARY SEARCH**

Binary search is a faster method for searching for an item that is in an **ordered list**. An ordered list is one where the sequence of items in the list is important. An ordered list does not necessarily contain a sequence of numbers (eg 1, 2, 3, 4) or characters (eg A, B, C, D). It might also contain, eg a list of names in alphabetical order, a list of ﬁles from smallest to largest or a list of records from earliest to most recent.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Albert | Ben | Edward | Freda | Gemma | Lisa | Peter | Rebecca | Simon |

A binary search is carried out to find the name Freda. Which names will be examined?

A binary search is made to find the name Jo. Which names will be examined?

Will it take more than 3 tries to find any name? If so, which name(s)? How many names have to be examined?

Draw a flow diagram to show how linear searches work below

|  |
| --- |
|  |

**COMPARISON OF SEARCHES**

Different algorithms might be best used in different situations. For example, sometimes an algorithm won’t work with a particular set of data, and in some instances one algorithm will be much quicker or more efficient than another.

|  |  |  |
| --- | --- | --- |
|  | Advantages | Disadvantages |
| Linear Search |  |  |
| Binary Search |  |  |

In a list of 100 items, how many items have to be examined to establish that an item is NOT in the list. Show your working out below

|  |
| --- |
|  |

##### SORTING ALGORITHMS

A sorting algorithm will put items in a list into an order, such as alphabetical or numerical order. For example, a list of customer names could be sorted into alphabetical order by surname, or a list of people could be put into numerical order by age. Sorting a list of items can take a long time, especially if it is a large list. A computer program can be created to do this, making sorting a list of data much easier.

There are many types of sorting algorithms. The three you need to understand are **bubble** **sort**, **merge** **sort** and **insertion** **sort**.

**BUBBLE SORT**

A bubble sort algorithm goes through a list of data a number of times, comparing two items that are side by side to see which is out of order. It will keep going through the list of data until all the data is sorted into order. Each time the algorithm goes through the list it is called a ‘pass’.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Unordered List** | **3** | **7** | **1** | **9** | **2** |
| 1st Pass |  |  |  |  |  |
| 2nd Pass |  |  |  |  |  |
| 3rd Pass |  |  |  |  |  |

Using your own list of numbers, Use the bubble sort

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Unordered List** | **23** | **5** | **16** | **76** | **54** | **31** | **40** | **13** | **61** | **48** |
| 1st Pass |  |  |  |  |  |  |  |  |  |  |
| 2nd Pass |  |  |  |  |  |  |  |  |  |  |
| 3rd Pass |  |  |  |  |  |  |  |  |  |  |
| 4th Pass |  |  |  |  |  |  |  |  |  |  |
| 5th Pass |  |  |  |  |  |  |  |  |  |  |
| 6th Pass |  |  |  |  |  |  |  |  |  |  |

The bubble sort is considered to be one of the simplest sorting algorithms as it only ever focuses on two items rather than the whole list of items.

|  |  |
| --- | --- |
| Advantages | Disadvantages |
|  |  |

**INSERTION SORT**

The insertion sort algorithm is the simplest algorithm to understand. It takes each item in turn and puts it in the right place using the first item in the list as a starting point.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Ball** | **Tackle** | **Post** | **Head** | **Shoot** | **Goal** |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

|  |  |
| --- | --- |
| Advantages | Disadvantages |
|  |  |

**MERGE SORT**

The merge sort algorithm is an example of a “divide and conquer” algorithm and takes advantage of two facts.

* Small lists are easier to sort than large lists
* It’s easier to merge two ordered lists than two unordered lists

Use the merge sort list to write these letters in alphabetical order

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **F** | **P** | **A** | **L** | **T** | **D** | **K** | **H** |

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |

|  |  |
| --- | --- |
|  |  |

|  |  |
| --- | --- |
|  |  |

|  |  |
| --- | --- |
|  |  |

|  |  |
| --- | --- |
|  |  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |
| --- |
|  |

|  |  |
| --- | --- |
|  |  |

|  |  |
| --- | --- |
|  |  |

|  |  |
| --- | --- |
|  |  |

|  |  |
| --- | --- |
|  |  |

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |

|  |  |
| --- | --- |
| Advantages | Disadvantages |
|  |  |