# **KS4 – Algorithms**

## Unit introduction

The main focus of this unit is on searching and sorting algorithms, though other topics are covered, such as computational thinking, flow charts, and tracing algorithms. Learners will have opportunities to analyse, interpret, modify, and implement a range of algorithms.

Each lesson that introduces a new searching or sorting algorithm uses numbers hidden under cups to help learners visualise how the algorithm works step by step. This is so that learners can better understand how a computer executes these algorithms, as a computer cannot ‘see’ all of the items in a list at once as a human can, but instead is only able to compare two values at a time. There are also written instructions alongside each of these steps that together represent the algorithm in structured English; coded solutions are provided in a subsequent lesson.

There are many crossovers with the KS4 Programming unit so the order of which unit is taught first, or possibly alongside, depends on the teacher. For instance, the lessons that involve tracing code (Lesson 3), coding linear search and binary search (Lesson 6) and coding bubble sort and insertion sort (Lesson 9) require learners to be familiar with programming concepts such as selection, iteration, and lists. However, the lessons on computational thinking, representing algorithms, and introducing the searching and sorting algorithms can be taught without any prior programming experience. These non-programming dependent lessons could be taught at the beginning of KS4 and then the code-based lessons in this unit could be revisited once learners are better prepared.

## Overview of lessons

| **Lesson** | **Brief overview** | **Learning objectives** |
| --- | --- | --- |
| 1 Computational thinking | In this lesson, learners are introduced to three computational thinking techniques: decomposition, abstraction, and algorithmic thinking. Learners will explore how these skills can be applied when solving a wide range of problems, both computer-based and in their everyday lives. They will be using these techniques throughout this unit when analysing and solving problems, especially around searching and sorting data.  Learners will be provided with a new problem that they have to help solve by applying decomposition, abstraction, and algorithmic thinking. There will be an opportunity for learners to peer- or self-assess their work and participate in discussions around computational thinking. | * Define the terms decomposition, abstraction, and algorithmic thinking * Recognise scenarios where each of these computational thinking techniques is applied * Apply decomposition, abstraction, and algorithmic thinking to help solve a problem |
| 2 Representing algorithms | The main focus of this lesson is on developing flow charts. This lesson assumes that learners have already covered the flow chart lesson in the KS4 Programming unit, although this lesson can also be used to introduce the flow chart symbols if required. A step-by-step worked example of a flow chart for a coin toss game is used to cover each of the flow chart symbols apart from subroutines, though you can ask learners to develop the dice roll simulation in the worksheet as a subroutine.  Learners are also introduced to using structured English to specify the steps of an algorithm in detail, though this is referred to as ‘written descriptions’ throughout the unit so as to not introduce more terminology than is necessary. Structured English is used to describe each of the searching and sorting algorithms later in the unit, so that learners can focus on the logic of the algorithm before being shown programmed examples of each algorithm. The only algorithm that does not have a programmed version is merge sort since this uses recursion and is beyond the GCSE specification. | * Describe the difference between algorithms and computer programs * Identify algorithms that are defined as written descriptions, flow charts, and code * Analyse and create flow charts using the flow chart symbols |
| 3 Tracing algorithms | For this lesson, learners will be shown examples of tracing a Python program and a flow chart. Trace tables are great for walking through an algorithm and are often used to locate logic errors. However, the focus of this lesson is mainly on using a trace table to understand how the algorithm works, as this is what learners will use trace tables for in the coding, searching, and sorting algorithms lessons, which are lessons 6 and 9 of this unit respectively. That being said, there is a logic error in the second task of the worksheet for finding the lowest number in a list, which learners will use a trace table to detect. The working code for all the algorithms is linked below.  It is assumed that learners have already covered the lessons on selection, while loops, for loops, nested loops, and lists in the KS4 Programming unit. These concepts are essential to analysing and interpreting the code for the searching and sorting algorithms later on in this unit. Learners should also be familiar with calculating integer division and modulo in Python, which is covered in Lesson 7 of the Programming unit. Integer division is key to the binary search program in this unit. | * Use a trace table to walk through code that contains a while loop, a for loop, and a list of items * Use a trace table to detect and correct errors in a program |
| 4 Linear search | In this lesson, learners are introduced to one of the two searching algorithms they need to know about: linear search. They will go over the steps of carrying out a linear search, and perform a linear search in real life and with a sample of data.  They will look at how searching is a common activity for both computers and humans, and discuss how the instructions can differ when the items are unordered or ordered. At this point linear search will be introduced as the only reasonable way to search through an unordered list of items. The learners will then be shown a demonstration on the slides using a number hidden under a row of cups.  Learners will follow the instructions to perform a linear search in small groups with a set of ten cards. After this, they will be introduced to the best- and worst-case scenarios of the performance of an algorithm before carrying out a linear search on a small sample of data. | * Identify why computers often need to search data * Describe how linear search is used for finding the position of an item in a list of items * Perform a linear search to find the position of an item in a list |
| 5 Binary search | In this lesson, learners are introduced to binary search, the second and final searching algorithm they need to know about. They will go over the steps of carrying out a binary search and perform a binary search with playing cards and with a sample of data.  Learners will be made aware that a binary search is only possible if data is ordered, otherwise a linear search must be performed or the data must be sorted. This is a great opportunity to acknowledge one of the reasons why sorting algorithms are useful before the learners are introduced to them in future lessons.  Learners should also be able to identify why binary search is generally a more efficient algorithm than linear search when dealing with ordered data because of its ‘divide and conquer’ approach. This should be made apparent to learners when going over the cup demonstration on the slides and when carrying out a binary search of their own with cards and a data sample.  One of the challenges learners can often be faced with is knowing what item to check when there is an even number of items. To make things clear without the need for a mathematical formula, the slides state that the middle-left item should be the next midpoint. In the next lesson, learners will be presented with the Python code for a binary search which uses the expression midpoint DIV 2 to clarify this middle-left choice. | * Describe how binary search is used for finding the position of an item in a list of items * Perform a binary search to find the position of an item in a list * Identify scenarios when a binary search can and cannot be carried out |
| 6 Comparing searching algorithms | In this lesson, learners will compare the features of linear search and binary search and the suitability of each algorithm in different contexts. They will also interpret the code of both algorithms in Python, as well as analysing the efficiency of two implementations of the linear search algorithm. Demo versions of linear search and binary search that output the steps of each comparison are linked in the ‘You may also need’ section, where you will also find the commented code used on the slides and worksheets.  Learners will start by analysing the factors that can affect the performance of linear and binary search: whether the data is ordered, the number of comparisons, and the simplicity of the algorithm. Next, they will perform a linear and binary search on a sample of data so that they can visually compare the number of comparisons made when using each algorithm.  The slides for the final two activities demonstrate and explain a step-by-step Python implementation for each searching algorithm. The slides for both the linear search and binary search in Python build up from the inside out, focusing first on how one comparison is made before adding the functionality to repeat this process.    Each of the Python activities culminates with a worksheet that gets learners to further explore and understand the demoed code with questions and tracing the algorithm with a given set of data. Furthermore, a more efficient version of linear search is presented in a worksheet so that learners can compare the efficiency of these two implementations. | * Compare the features of linear and binary search and decide which is most suitable in a given context * Interpret the code for linear search and binary search * Trace code for both searching algorithms with input data |
| 7 Bubble sort | This lesson introduces learners to the first sorting algorithm in this unit: bubble sort. They will discuss why and where sorting is used in real life, become familiar with performing a bubble sort on a set of data, and investigate the efficiency of bubble sort. The sample data used in these worksheets and most of the other sorting algorithm worksheets is words instead of numbers, as learners can often make mistakes when comparing whether one word is higher or lower than another word. This will allow them plenty of practice in case they have to do this in an exam. Some learners can find writing the alphabet out helpful.  The essence of sorting data is to make searching easier, and this idea should frame the narrative of all the sorting lessons. The learners will start by performing a single pass first, so that they become used to the crux of the algorithm before doing multiple passes. By the end of the lesson they should be able to execute a full bubble sort on a sample of data. They should also be aware that this algorithm is too slow for real-world applications. It is, however, a good algorithm for introducing sorting algorithms and should be seen as a stepping stone to other sorting algorithms that perform better on large sets of data.  Learners will also investigate a few ways to improve the efficiency of the algorithm, such as stopping if no swaps are made on a single pass and reducing the number of comparisons by one after each pass. This can result in the bubble sort algorithm being a viable option in some cases, for instance small sets of data that are nearly sorted. | * Identify why computers often need to sort data * Traverse a list of items, swapping the items that are out of order * Perform a bubble sort to order a list containing sample data |
| 8 Insertion sort | In this lesson, learners will explore another sorting algorithm: insertion sort. Some exam boards do not require learners to know insertion sort, so do check the specification first. Learners will start by discussing how they would sort objects in real life, which may lead them to describe something akin to an insertion sort. The link between sorting real life objects and insertion sort should be particularly evident when asking learners how they would add a new item to a group of sorted objects.  To begin with, the slides take learners through a broken-down version of one pass of an insertion sort that reflects the coded solution introduced in the next lesson. Then the slides will take learners through multiple passes of an insertion sort, which is how they will apply the algorithm when executing it on samples of data. Being able to recognise and highlight the sublists at different stages of an insertion sort is essential for showing the individual steps of an insertion sort and will help with understanding the coded implementation next lesson.  Learners will get to practise performing an insertion sort on samples of data of their own, which will include a partially worked example to support them with carrying out each pass of the algorithm. Again, words are used instead of numbers in the data samples, as this is often what learners struggle more with and therefore need more practice. | * Insert an item into an ordered list of items * Describe how insertion sort is used for ordering a list of items * Perform an insertion sort to order a list containing sample data |
| 9 Coding sorting algorithms | In this lesson, learners will analyse and evaluate code for bubble sort and insertion sort in Python, as well as comparing different implementations of the bubble sort algorithm. Demo versions of bubble sort and insertion sort that output the steps of each pass are linked in the ‘You may also need’ section, where you will also find the commented code used on the slides and worksheets and the three versions of bubble sort.  Learners will first be presented with different statements referring to bubble sort and insertion sort and need to work out whether they are true or false. The remaining slides demonstrate and explain a step-by-step Python implementation for each sorting algorithm, as well as two improvements to the bubble sort code. The slides for both the bubble sort and insertion sort in Python build up from the inside out, focusing first on how one pass is performed before adding the functionality to repeat this process.    Each of the Python activities culminates with a worksheet that gets learners to further explore and understand the demoed code with questions and tracing the algorithm with a given set of data. Furthermore, a more efficient version of bubble sort is presented in a worksheet so learners can compare the efficiency of these two implementations. | * Interpret the code for bubble sort and insertion sort * Trace code for both sorting algorithms with input data * Identify factors that could influence the efficiency of a bubble sort implementation |
| 10 Merge sort | In this lesson, learners will explore the final sorting algorithm in this unit: merge sort. They will start by considering how they might go about combining two groups of sorted items into one sorted group before being taken through the steps of one merge of a merge sort. This leads on to an activity that focuses solely on merging a pair of lists together so that they can be comfortable with this process before moving onto the full algorithm.  The slides will provide a further step-by-step visualisation of an entire merge sort, first splitting the lists until each item is in a list of its own and then merging pairs of lists in order. It is essential for learners to identify which pairs of lists should be merged together and which items in a pair of lists need to be compared at each stage in the algorithm.  Learners will get to practise executing a full merge sort on the samples of data provided. Both worksheets contain a partially worked example to support them with merging lists together. They will also be given an overview of how well merge sort performs compared to bubble sort. | * Merge two ordered lists of items into a new ordered list * Describe how merge sort is used for ordering a list of items * Perform a merge sort to order a list containing sample data |
| 11 Algorithms review | In this lesson, learners will have time to practise and cement their knowledge of some of the things they have learnt in the algorithms unit. The worksheets contain a range of questions on flow charts, searching algorithms, and sorting algorithms that will help prepare them for the summative assessment for the unit.  The slides contain solutions to a select few tasks from the worksheets that will provide you with an opportunity to discuss these questions and check learners’ understanding. It will also be useful to get learners to peer-assess the worksheets as much as possible so that they can gain an insight into how other learners have answered the questions, especially the ‘describe’ and ‘explain’ questions. | * Interpret algorithms and suggest improvements * Analyse and fix errors in a flow chart * Perform searching and sorting algorithms on samples of data |
| 12 Summative assessment | In the final lesson of the algorithms unit, learners will complete a summative assessment that will test their understanding of searching and sorting algorithms, flow charts, tracing, and computational thinking terms. The type of questions mainly consist of multiple-choice and short-answer questions to make marking easier. Learners will start the lesson by exploring a piece of code in Python before developing the function for linear search in the program provided, which contains subgoal labels written as comments in the code. | * Develop a linear search function in Python |

## Progression

This unit progresses students’ knowledge and understanding of algorithms.

Please see the learning graphs for this unit for more information about progression.

## Curriculum links

[**National curriculum links**](https://www.gov.uk/government/publications/national-curriculum-in-england-computing-programmes-of-study/national-curriculum-in-england-computing-programmes-of-study)

* Develop their capability, creativity, and knowledge in computer science, digital media, and information technology
* Develop and apply their analytic, problem-solving, design, and computational thinking skills

## Assessment

### **Summative assessment**

* Please see the assessment question and answer documents for this unit.

## Subject knowledge

This unit focuses on searching and sorting algorithms; how they can be applied on samples of data and implemented as programs in Python. It also teaches learners the key computational thinking concepts, provides opportunities for developing flow charts, and allows them to trace algorithms that contain conditions and loops.

Enhance your subject knowledge to teach this unit through the following training opportunities:

### **Online training courses**

* [Programming Pedagogy in Secondary Schools: Inspiring Computing Teaching](https://www.futurelearn.com/courses/secondary-programming-pedagogy) by the Raspberry Pi Foundation, hosted by FutureLearn
* [Programming 101: An Introduction to Python for Educators](https://www.futurelearn.com/courses/programming-101) by the Raspberry Pi Foundation, hosted by FutureLearn
* [Programming 102: Think like a Computer Scientist](https://teachcomputing.org/courses/CO208/programming-102-think-like-a-computer-scientist) by the Raspberry Pi Foundation, hosted by FutureLearn

### **Face-to-face and remote courses**

* [Introduction to algorithms, programming, and data in computer science](https://teachcomputing.org/courses/CP228/an-introduction-to-algorithms-programming-and-data-in-gcse-computer-science-face-to-face) (face to face)
* [Introduction to algorithms, programming, and data in computer science (remote)](https://teachcomputing.org/courses/CP438/an-introduction-to-computer-systems-networking-and-security-in-computer-science-remote)
* [Representing algorithms using flowcharts and pseudocode](https://teachcomputing.org/courses/CP220/representing-algorithms-using-flowcharts-and-pseudocode-face-to-face) (face to face)
* [Representing algorithms using flowcharts and pseudocode (remote)](https://teachcomputing.org/courses/CP420/representing-algorithms-using-flowcharts-and-pseudocode-remote)
* [Search and sort algorithms](https://teachcomputing.org/courses/CP230/search-and-sort-algorithms-face-to-face) (face to face)
* [Search and sort algorithms (remote)](https://teachcomputing.org/courses/CP430/search-and-sort-algorithms-remote)

Resources are updated regularly — the latest version is available at: [ncce.io/tcc](http://ncce.io/tcc).

This resource is licensed under the Open Government Licence, version 3. For more information on this licence, see [ncce.io/ogl](http://ncce.io/ogl).