

Copyright

© 2016 PG Online Limited

The contents of this unit are protected by copyright.

This unit and all the worksheets, PowerPoint presentations, teaching guides and other associated files distributed with it are supplied to you by PG Online Limited under licence and may be used and copied by you only in accordance with the terms of the licence. Except as expressly permitted by the licence, no part of the materials distributed with this unit may be used, reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic or otherwise, without the prior written permission of PG Online Limited.

Licence agreement

This is a legal agreement between you, the end user, and PG Online Limited. This unit and all the worksheets, PowerPoint presentations, teaching guides and other associated files distributed with it is licensed, not sold, to you by PG Online Limited for use under the terms of the licence.

The materials distributed with this unit may be freely copied and used by members of a single institution on a single site only. You are not permitted to share in any way any of the materials or part of the materials with any third party, including users on another site or individuals who are members of a separate institution. You acknowledge that the materials must remain with you, the licencing institution, and no part of the materials may be transferred to another institution. You also agree not to procure, authorise, encourage, facilitate or enable any third party to reproduce these materials in whole or in part without the prior permission of PG Online Limited.

Algorithms

Teacher’s Guide

Introduction

This teacher’s guide contains a detailed lesson plan to accompany the set of PowerPoint slides and worksheets for each lesson.

The lessons are designed to form a basis for ideas for the teacher and should be adapted to suit the teaching style and preferences of the individual teacher, and the resources and nature of the individual school or Computing / ICT department.

The material supplied for this unit includes:

* 6 PowerPoint presentations, each designed to cover one lesson
* 6 worksheets
* 5 homework sheets
* An end-of-unit test for assessment purposes
* Python programs implementing pseudocode

Summary

This is a theoretical unit covering all of Section 4.3 Fundamentals of algorithms (except Section 4.3.3 Reverse Polish which is covered in Unit 9). Searching and sorting algorithms are covered in an interactive and practical way, with reference to Big-O notation in terms of time and space complexity. It also covers Section 4.1.1.15 on the role of stack frames in subroutine calls, and Section 4.1.1.16 on recursive techniques, putting these into practice with tree traversals and a depth-first graph traversal. Optimisation algorithms, such as Dijkstra’s shortest path algorithm are covered along with a complete topic on the limits of computation, including intractable algorithms and the Halting problem.

Learning Outcomes for the unit

**At the end of this Unit all students should be able to:**

* state the essential characteristics of a recursive algorithm
* insert items into a binary search tree
* state the order in which nodes are visited in pre-order, in-order and post-order tree traversals
* give examples of linear, polynomial, exponential and logarithmic functions
* compare two algorithms in terms of efficiency
* explain the principles of a linear and binary search
* state a possible order in which nodes are visited in depth first and breadth first graph traversals
* state applications of each graph traversal
* state the purpose and applications of Dijkstra’s shortest path algorithm
* Describe the Travelling Salesman problem
* Explain what is meant by a tractable or intractable problem

**Most students will be able to:**

* trace through recursive algorithms including pre-order, in-order and post-order tree traversals
* state the time complexity of an algorithm
* write an algorithm for a linear search
* trace through a bubble sort algorithm
* explain how the merge sort works and analyse its time complexity
* describe applications of each graph traversal
* be able to trace Dijkstra’s shortest path algorithm
* Give examples of intractable problems
* Explain what is meant by a Heuristic method and why it might be used
* Describe the Halting problem and its significance for computation

**Some students will be able to:**

* write a recursive algorithm to solve a simple problem
* show the changing contents of a call stack as a recursive routine is executed
* derive the time complexity of an algorithm
* write an algorithm for a binary search
* write a merge algorithm
* trace depth-first and breadth-first graph traversal algorithms

Previous Learning

Students would benefit from having studied relevant material from the new KS3 National Curriculum and more specifically a Computer Science related GCSE. However, the material presented in this unit will not assume that students have studied these topics prior to this course.

Suggested Resources

The textbook *AQA A Level Computer Science (Year 2)* or *AQA A Level Computer Science* (AS and A Level Year 2 in one volume).



A complete course text that provides a comprehensive understanding of each topic in both years of the new AQA A Level Computer Science specification. It is presented in an accessible and interesting way, with many in-text questions to test students’ understanding of the material and ability to apply it.

The complete book is divided into 12 sections, each containing roughly six chapters. Each chapter covers material that can comfortably be taught in one or two lessons. It will also be a useful reference and revision guide for students throughout the AS and A Level courses.

Two short appendices contain A Level content that could be taught in the first year of the course as an extension to related AS topics.

Each chapter contains exercises, some new and some from past examination papers, which can be set as homework. Answers to all these are available to teachers only in a Teachers Supplement which can be ordered from our website www.pgonline.co.uk.

Vocabulary

Vocabulary associated with this Unit, such as:

recursion, recursive subroutine, call stack, tree traversal, pre-order, in-order, post-order traversal,

Big-O notation, linear, polynomial, exponential, logarithmic functions, permutation, time complexity,

linear search, binary search, binary tree search, bubble sort, merge sort,

depth-first traversal, breadth-first traversal,

optimisation problem, Dijkstra’s shortest path algorithm,

limits of computation, travelling salesman problem (TSP), computational problem, tractable and intractable problems, heuristic solution, computable and non-computable problems, Halting problem

Assessment

Assessment will be by means of regular homework and a test with examination style questions.

Topic plans

|  |  |  |
| --- | --- | --- |
| Topic 1 | Recursive algorithms |  |
| Learning Objectives:   * Be familiar with the use of recursive techniques in programming languages * Be able to solve simple problems using recursion * Be able to trace recursive tree-traversal algorithms: pre-order, post-order, in-order | | |
| Content | | Resources |
| **Starter**  The factorial function is a good way to introduce recursion, in which a subroutine calls itself. Students will be familiar with the factorial notation, in which  5! = 5 x 4 x 3 x 2 x 1  Ask them to calculate this, and then 6! in terms of 5! (120 x 6 = 720)  Note that 0! = 1 by definition.  Get them to calculate 7! (720 x 7 = 5040) and note in passing how quickly the numbers increase. How can n! be defined in terms of itself? (See next slide)  **Main**  Move on to recursion and look at the pseudocode on the slide headed **A recursive routine**.  Before the subroutine reaches the last statement, it calls itself with the parameter n-1.  So factorial = 4 x factorial (3)  = 4 x 3 x factorial(2)  = 4 x 3 x 2 x factorial(1)  = 4 x 3 x 2 x 1 x factorial(0)  When the subroutine is called with the parameter 0, factorial is put equal to 1 control passes to the OUTPUT statement.  **1** is output.  The end of the subroutine is reached, so control passes to the statement after the CALL statement, statement 6.  This is the OUTPUT statement so **1** is output.  The subroutine ends again, with parameter n = 1  n! = 1 so 1 is again output.  This is repeated as the subroutine continues to unwind and the numbers  1, 1, 2, 6, 24 are output.  The Python program factorial.py demonstrates this. If the students do not use Python, they can write the program in a language with which they are familiar.  The next slide shows how the call stack is used. This is a bit of revision of this topic which students met in Unit 7.  **Essential characteristics of a recursive subroutine**  Go over the three essential characteristics. These need to be understood and remembered.  **Stack overflow**  If recursion carries on too long, the program will crash as memory space for the stack will run out. The example on Stack overflow crashes because there is no base case and recursion continues for ever. See if the students can spot this.  There are programs called **recursiveSum (Python and VB versions)** in the respective program folders.  There are space limits on an iterative routine if for example appends to a list indefinitely. Also it may loop for ever if an error has been made.  Many problems can be solved using either an iterative routine or a recursive routine.  Students could program these routines if time and equipment is available. A Python version of both subroutines is in **addNums.py**  Give out **Worksheet 1** and ask students to complete Task 1.  Go over the answers. Tracing through a recursive routine is tricky!  **Tree traversal algorithms**  Tree traversals are a good example of recursion, as a tree is essentially a recursive data structure. A tree consists of subtrees which consist of subtrees…  As a brief revision, ask students to write down the order of nodes visited for each traversal in the tree shown.  Pre-order: 24 15 9 17 38 29 58  In-order: 9 15 17 24 29 38 54  Post-order: 9 17 15 29 58 38 24  If they have any trouble, explain the process and give them another tree to practise on.  What is special about an in-order traversal? (In a binary search tree, the nodes are visited in ascending order).  Ask them to create a binary search tree from an unordered list of numbers. This was covered in Unit 7.  **Algorithms for tree traversal**  In the algorithms, -1 is used as a “dummy” pointer to indicate that this node is a leaf (i.e. has no subtrees). Task 2 in the worksheet reminds students of this.  Go over the answers when students have completed the questions.  **Plenary**  Recall the three essential properties of a recursive routine.  Give out **Homework 1**. | | PowerPoint Guide: Algorithms T1 Recursive algorithms  factorial.py  stackOverflow.py  recursiveSum.py  recursiveSum.vb  addNums.py  Algorithms Worksheet 1 Recursive algorithms  Algorithms Worksheet 1 Answers  recursiveSumEvenNumbers.py  recursiveSumEvenNumbers.vb  sumList (recursive).py  sumList (recursive).vb  Algorithms Homework 1 Recursive algorithms  Algorithms Homework 1 Answers |

|  |  |  |
| --- | --- | --- |
| Topic 2 | Big-O notation |  |
| Learning Objectives:   * Be familiar with the concept of a function as a mapping from one set of values to another * Be familiar with the concept of constant, linear, polynomial, exponential and logarithmic functions * Be familiar with the notion of permutation of a set of objects or values * Be familiar with the Big-O notation to express time complexity * Be able to derive the time complexity of an algorithm | | |
| Content | | Resources |
| **Starter**  This topic is all about the efficiency of algorithms and how that can be measured.  Show the first slide and ask which is more efficient. The formula in the second algorithm sums an arithmetic progression.  However large n is, the second algorithm will only use 2 statements. The first algorithm has n + 1 statements.  **Main**  Before moving on to Big-O notation, students need to be reminded of different types of function, and the shapes of the graphs that represent them.  **Mathematics revision**  This is just a gentle lead into the mathematics. Remember, that not all Computing students will necessarily be taking A-Level mathematics. It will be necessary to explain some concepts from the basics.  **Linear functions**  The slide shows a linear function in algebraic and graphical form.  **Quadratic functions**  Next, discuss the quadratic function, and the fact that the term in n2 will increase very much faster than the other two terms as n becomes very large.  **Logarithmic functions**  Spend a few minutes going over logarithms. Be consistent and insist on the subscript with the log term. Just assure students that we will only deal with base 2 in computing.  **Big-O notation**  At this point you can introduce Big-O notation.  **Analysing an algorithm**  In order to measure how efficient an algorithm is, you need to calculate the number of statements it executes for a problem of size n. From this, the Big-O notation for the algorithm can be derived.  The given algorithm has 1 + 3n + n = 1 + 4n statements. This is a linear function. The coefficient of n and the constant term are ignored in Big-O notation because they have very little effect on the execution time of the algorithm when n is large.  So the time complexity is O(n).  **Logarithmic functions**  “Divide and conquer” algorithms are very efficient, with execution time increasing very slowly as the size of the problem doubles. A binary search is a good example. It is O(log2 n)  Give out **Worksheet 2** and ask students to complete **Task 1**.  Go over the answers when the task is complete.  **Permutations**  Some examples of permutations are given – you can probably think of others.  **With repetition**  From a pot of 10 numbers (0 – 9):  Possible choices for 1 digit = 101 = 10  Possible choices for 2 digits = 101 + 101 = 102 = 100  Possible choices for 3 digits = 101 + 101 + 101 = 103 = 1000  Possible choices for 4 digits = 101+101+101+101=104=10,000  Ask if students can identify the Big-O for this problem, just by looking at the pattern.  **Without repetition**  The example with 3 colours is small enough that students can check their logic by drawing out all the different permutations. Once they’re convinced that 3 x 2 x 1 is correct, have them generalise the logic by using 5 and 7 differently coloured balls. They may need reminding that this formula is a factorial.  **Comparison of time complexities**  The graph shows that execution time for algorithms of time complexity O(n!) or O(2n) increases very rapidly. These algorithms are generally impractical for any problem where n is other than a very small number.  Ask students to complete the questions in **Task 2** of the worksheet.  Question 3 shows that a recursive algorithm is not necessarily the best choice. The problem with this particular recursive algorithm is that the same values gets calculated over and over again, each time the routine is called.  It is instructive to run the Python program, which times each routine for different values of n. Don’t try it with n>35!  (You will get different answers from the ones given.)  **At the end of this worksheet there is a multiple choice quiz on Big-O notation.** You could give this to the students now, or keep it for a revision lesson.  **Plenary**  Make sure students understand how to calculate Big-O, and that only the dominant term is significant. Coefficients are ignored.  Give out **Homework 2**. | | PowerPoint Guide:  Algorithms T2 Big-O notation.pptx  <http://logbase2.blogspot.co.uk/2008/08/log-calculator.html>  Algorithms Worksheet 2 Big-O notation  Algorithms Worksheet 2 Answers  Algorithms Homework 2 Big-O notation  Algorithms Homework 2 Answers |

|  |  |  |
| --- | --- | --- |
| Topic 3 | Searching and sorting |  |
| Preparation:   * The Word documents **Girls names (10)** and **Girls names (20)** contain two sets of girls’ names. You need one set for each student, preferably laminated and cut up in advance. These will be used in kinaesthetic exercises to manually search and sort a list.   Learning Objectives:   * Know and be able to trace and analyse the time complexity of the linear search and binary search algorithms * Be able to trace and analyse the time complexity of the binary tree search algorithm * Know and be able to explain and trace and analyse the time complexity of the bubble sort algorithm * Be able to trace and analyse the time complexity of the merge sort algorithm | | |
| Content | | Resources |
| **Starter**  Hand out the name cards and show the first two slides. Students can perform a manual search – hopefully a systematic linear search rather than picking random cards… this can be a topic for discussion.  On average, students should have had to turn over 5 cards. Ask students how many each turned up and see if this is true. Worst case scenario is that Lily is the 10th card to be turned over.  **Main**  **Algorithm for linear search**  Give students some time to study the algorithm, write it down and fill in the missing lines. Answer on next slide, “Analysing the algorithm”.  Ask them to calculate the number of steps for a list of size n (worst case scenario) and hence, the Big-O time complexity of this algorithm.  **Bubble sort**  Students should be familiar with the Bubble sort. If necessary, demonstrate it manually and then let them practise it using the name cards. At the end of the first pass, the last name in the sequence should be correctly placed at the end. The second pass looks only at (n-1) items, the third pass, (n-2) items etc.  Ask them to calculate the Big-O time complexity of the algorithm. It is O(n2). The clue is the two nested loops.  One way to calculate it is: Outside x (inside x 3 times) = n x (n-1) x 3 = 3n(n-1) = 3n2 – 3n = O(n2)  The slide with the answers calculates it in a slightly different way.  **Binary search**  Go over the algorithm for a binary search.  Give out **Worksheet 3** and ask students to complete Task 1. They will use the cards to do a manual binary search.  Go over the answers.  **Binary tree search**  In a binary search tree, the order in which the girls’ names are entered into the tree will determine how many items have to be examined before the right one is found. But as for the binary search, 4 is the maximum.  The order of complexity for a binary search of a balanced tree is O(log2 n). This is because half the choices are discarded on each comparison.  If the items are already sorted before being put into the binary search tree, the tree will be unbalanced. A search for the targeted name degrades to a linear search. Therefore, the time complexity will be O(n).  **Merge sort**  This is explained in some detail. Once again, students can use the cards to carry it out manually once the 2-step process has been explained.  The merge algorithm is shown on a separate slide. Students can practise this with the exercise in **Task 2** of **Worksheet 3**.  The Big-O time complexity is O(n log n)  **Plenary**  Go over the main points of the lesson, and give out **Homework 3.** | | PowerPoint Guide:  Algorithms T3 Searching and sorting.pptx  Girls names (10)  Algorithms Worksheet 3 Searching and sorting  Algorithms Worksheet 3 Answers  Merge sort.py  Merge sort.vb  Algorithms Homework 3 Searching and sorting  Algorithms Homework 3 Answers |

|  |  |  |
| --- | --- | --- |
| Topic 4 | Graph traversal algorithms |  |
| Learning Objectives:   * Be able to trace depth-first and breadth-first algorithms * Describe typical applications of each | | |
| Content | | Resources |
| **Starter**  As a revision exercise, ask students to draw the graph represented by the adjacency list.  **Main**  Revise the terminology of a graph – edges, nodes or vertices, neighbours or adjacent nodes, directed and undirected graphs.  **Traversing a graph**  Describe the two ways of traversing a graph – depth-first and breadth-first.  **Depth first traversal**  A graph can be represented in a programming language using a data structure. The slide shows how the graph shown would be represented in Python.  This algorithm uses a **stack** to keep account of which nodes have been visited, and a list of visited nodes. A stack is used to keep track of the previous node visited so that we can backtrack when the end of a path is reached by popping the item off the top of the stack, searching for another path.  The algorithm starts by adding the start vertex A to the list of visited nodes. When going through the algorithm graphically, each node is coloured dark blue once it has been added to the visited list.  Subsequent slides show the progress of the algorithm. The algorithm ends when all nodes have been visited and the stack is empty.  **Algorithm for depth-first traversal**  The algorithm is recursive, and a system call stack is maintained automatically so does not need to be created manually.  Python and VB programs **dfs.py/vb** implementing the algorithm are included in the relevant program folders.  Students will not be expected to reproduce the algorithm but should understand and be able to trace it through. Practice at tracing through a depth first algorithm is given both in the worksheet and in the homework.  **Applications of depth-first search**  Navigating a maze is a classic example of using a depth-first search.  Note: If you’re looking for any solution to the maze, then you can stop the algorithm when the exit node is first encountered. However, if you’re looking for all possible solutions, then you have to keep going until all the nodes have been visited.  Give out **Worksheet 4** and ask students to complete Task 1. The algorithm given is slightly different because it does not produce a list of visited nodes, it simply finds a path from the start node to every other node using a depth-first traversal.  Tracing through a recursive algorithm is daunting at first. The best advice for students is that they should understand how the depth-first traversal works and keep looking at a graphical image of the graph they are traversing.  Some guidance is given both in the Worksheet answers and in the Homework answers – you can decide how much help they need to complete the homework, and whether to give guidance in advance or let them try without guidance.  **Breadth-first traversal**  A queue is used instead of a stack as the supporting data structure. It is necessary to distinguish between nodes which have been **visited** (coloured pale blue) and added to the queue, and nodes which have been **fully explored** (dequeued and coloured dark blue).  Once again a graphical representation of the steps in the algorithm is given.  Ask students to do **Task 2** on the worksheet.  There are Python and VB programs **bfspy/vb** in the program folders.  The breadth-first traversal can be turned into a depth-first by simply changing the queue into a stack. This is demonstrated in the programs **dfs or bfs iterative version.py/vb**. Simply change the statement  currentNode = supporting\_adt.pop(0)  to currentNode = supporting\_adt.pop  in a depth-first traversal to switch between breadth-first and depth-first.  Students may like to try running the program.  **Applications of breadth-first search**  Some applications are given – in the next lesson students will learn about a well-known algorithm for finding the shortest path between two nodes.  **Time complexity of each search**  This depends on the number of edges. In a sparse graph, it is O(n). In a graph with the maximum number of edges, it is O(n2).  **Plenary**  Go over the main points and give out **Homework 4**. | | PowerPoint Guide:  Algorithms T4 Graph traversal algorithms.pptx  dfs.py  dfs.vb  Algorithms Worksheet 4 Graph traversal algorithms  Algorithms Worksheet 4 Answers  bfs.py  bfs.vb  dfs or bfs iterative version.py  dfs or bfs iterative version.vb  Algorithms Homework 4 Graph traversal algorithms  Algorithms Homework 4 Answers |

|  |  |  |
| --- | --- | --- |
| Topic 5 | Optimisation algorithms |  |
| Learning Objectives:   * Understand and be able to trace Dijkstra’s shortest path algorithm * Be aware of applications of the shortest path algorithm | | |
| Content | | Resources |
| **Starter**  Discuss some typical shortest path problems.  **Main**  This lesson explains Dijkstra’s shortest path algorithm. Students need to be able to trace through the algorithm and understand how it works, but will not be expected to recall the steps.  After showing Slide 4 entitled “Dijkstra’s shortest path algorithm”, you may find it useful to show a You Tube video showing how a shortest path algorithm works. There is one at <https://www.youtube.com/watch?v=WN3Rb9wVYDY> and there are several others.  Or better, watch the video first and then do the stepping through live on the whiteboard.  The steps as described in this video are:  Assign temporary distances or “costs” of 0 to A (starting vertex) ∞ to all other vertices.  Then visit the vertex with the lowest cost –at the start, this is **a**, the “current vertex”.  Visit neighbours of **a** (**b** and **c** here) and calculate costs by adding edge weights of **ab** and **ac** to cost at current vertex **a**.  If the calculated cost is less than the one at the vertex, replace it.  Move to the vertex with the lowest cost, **c**.  C:\Users\Rob\AppData\Roaming\PixelMetrics\CaptureWiz\Temp\54.png  Cost at **c** + edge weight **bc** = 2 + 1 = 3 so replace cost at **b** with 3. Replace cost at **e** with 2 + 10 = 12.  Repeat the steps for each vertex – the next one to be visited is **b**.  At the end of the algorithm, the distance of the shortest path from the start to each vertex is marked at the vertex.  C:\Users\Rob\AppData\Roaming\PixelMetrics\CaptureWiz\Temp\55.png  When you have finished the explanation, see if students can write down how it works. Make up another graph with different vertices and edges to give them practice at tracing through the algorithm.  **Implementing the algorithm**  The implementation of the algorithm shown on the next few slides is similar to a breadth-first search but uses a priority queue instead of an ordinary FIFO queue.  At each stage, the shortest distance from the start node to each of the neighbours of the current node is calculated. The node with the shortest distance is moved to the front of the priority queue, if it is not already there.  This ensures that the next vertex to be dequeued is the neighbour with the minimum “cost” attached to it.  Give out **Worksheet 5** and ask students to complete the two questions.  These gives practice in following through the algorithm using the priority queue.  **Plenary**  Show the first Plenary slide, ask the questions and make sure that students can briefly describe two or three applications of the shortest path algorithm. Suggested answers are given on the next slide.  Give out **Homework 5**. | | PowerPoint Guide:  Algorithms T5 Optimisation algorithms.pptx  This article is the description of how Dijkstra’s shortest path moved from the shortest distance between any 2 nodes and the shortest distance between all nodes (spanning tree). It has an interesting kinaesthetic activity described at the top.  <http://people.mpi-inf.mpg.de/~mehlhorn/ftp/Toolbox/ShortestPaths.pdf>  Algorithms Worksheet 5 Optimisation algorithms  Algorithms Worksheet 5 Answers  Algorithms Homework 5 Optimisation algorithms  Algorithms Homework 5 Answers |

|  |  |  |
| --- | --- | --- |
| Topic 6 | Limits of computation |  |
| Preparation:   * Put the following link in each student’s area. It is a game which they will find fun and very instructive! It covers a lot of the material taught in the last few lessons.   <http://www.open.edu/openlearn/science-maths-technology/computing-and-ict/computing/the-jewels-heuro>  Learning Objectives:   * Be aware that algorithmic complexity and hardware impose limits on what can be computed * Know that algorithms may be classified as being either tractable or intractable * Be aware that some problems cannot be solved algorithmically * Describe the Halting problem, and understand its significance for computation | | |
| Content | | Resources |
| **Starter**  Discuss what is meant by a computable problem. Could a computer do a 1000-piece jigsaw puzzle?  What is a computational problem? Fifty years ago, people would have said that face recognition was not a computational problem – but it turns out that it is. Even your camera can focus automatically on a smiling face. Think of some problems that are not computational.  Is translating a book a computable problem? Is writing a book a computable problem? Is solving a Sudoku puzzle computable? Is making up a new Sudoku puzzle computable?  A computer can tell you the best route from A to B; can it tell you the best way for you to pass your Computer Science exam?  Can a computer tell the managing director of a company which marketing strategy will be most cost effective – advertising on TV, radio or in the newspapers, special offers, loyalty cards, direct mail etc.?  Are all computational problems actually computable? Not if they take 10 million years to solve. But again, a problem that 50 years ago may have taken years to solve, can now be solved in seconds thanks to superfast computers.  **Main**  Discuss the computational problem of cracking a password. How would students set about this? Are there better ways than the “brute-force” method of trying every option?  The site <https://howsecureismypassword.net/> will tell you how long it will take to crack a particular password.  **Limits of computation**  Discuss the limits of computation. These are algorithmic complexity (think Big-O) and **hardware**.  There is an interesting history of weather forecasting, once considered incomputable, at <http://www-history.mcs.st-andrews.ac.uk/HistTopics/Weather_forecasts.html>  It would be worth students looking at this.  **The travelling salesman problem (TSP)**  This is a very well-known problem, which has applications in many fields from planning the optimum route for the school bus run, to the manufacture of circuit boards.  Students could Google this to read around the subject.  **Tractable problems**  Any problem with Big-O of O(n!) or O (2n) is regarded as **intractable**. Note that intractable is not the same thing as insoluble. The problem may be soluble for a small value of n, or by finding a different algorithm.  Give out **Worksheet 6** and ask students to complete **Task 1**.  Discuss the answers when completed.  **Heuristic methods**  Heuristic methods can be used to find a “good enough” solution to an intractable problem. We all use heuristic methods – such as common sense, an educated guess or gut instinct!  Some problems really are incomputable – see the “four-tile problem”, explained at  <http://www.cs4fn.org/algorithms/tiles.php>  **The Halting problem**  This is the problem of determining for a given input, whether a program will finish running or continue for ever. Alan Turing proved in 1936 that it was impossible to devise a machine or algorithm which will show, given *any* program and its inputs, whether it will halt or continue for ever.  The significance of this is that Turing proved that there are some problems which cannot be solved by computer.  This is an important statement which students would do well to understand and memorise.  Ask students to complete **Task 2** on the worksheet.  **Plenary**  Go over the main points of the lesson. There is no homework this week – students should revise for the test next week.  There is an excellent game called **The Jewels of Heuro** written Dr Michel Wermelinger (Faculty of Mathematics, Computing and Technology) at the Open University which you should encourage all your students to play – it will teach them a lot about algorithms, Dijkstra’s shortest path algorithm, The Travelling Salesman Problem (TSP), brute force methods, and heuristics. And it’s fun!  Try the game by following the link  <http://www.open.edu/openlearn/science-maths-technology/computing-and-ict/computing/the-jewels-heuro>  This is given as the last page of the worksheet but you simply need to put the link in each student’s area rather than printing out the worksheet page. | | PowerPoint Guide:  Algorithms T6 Limits of computation.pptx  Algorithms Worksheet 6 Limits of computation  Algorithms Worksheet 5 Answers  A video that helps make the connection between ‘Turing machines’ and the halting problem.  <https://www.youtube.com/watch?v=macM_MtS_w4>  Another animated one: <https://www.youtube.com/watch?v=92WHN-pAFCs&nohtml5=False> |

|  |  |
| --- | --- |
| Unit assessment | |
| Learning Outcomes:  Students will   * apply their knowledge in answers to a range of questions * be able to highlight areas of strength and any gaps in their understanding of computers | |
| Content | Resources |
| Students should complete the **Assessment Test**. | Algorithms Final Assessment  Algorithms Final Assessment Answers  bubble sort with flag.py |

Acknowledgements

The authors and publisher would like to thank all contributors for their kind permission to reproduce their photographs or images, screenshots of their websites or other copyright material in the PowerPoint Guides.

Diagrams © PG Online Ltd

Images otherwise stated © Shutterstock

Every effort has been made to trace the copyright holders and we apologise in advance for any unintentional omissions. We would be pleased to insert the appropriate acknowledgement in any subsequent edition of this publication.

This material contains links to relevant websites featured in the teacher’s guide. Every effort has been made to ensure that at the time of distribution, the links remain unbroken, the material remains up-to-date and that links are not inadvertently linked to sites that could be considered offensive. PG Online cannot be held responsible for the content of any website mentioned in this material. It is sometimes possible to find relocated sites by typing the original URL into a browser. Any errors should be reported directly to support@pgonline.co.uk and changes will be made in any subsequent editions of the material.

Artwork



From Here the Sea © 2015 Andrew Bird

Acrylic on board, 61x76cm

www.abirdart.co.uk