

# Revision checklist

The tables below can be used as a revision checklist: **It doesn’t contain all the detailed knowledge you need to know, just an overview.** For more detail see the syllabus and talk to your teacher.

[A Level Further Mathematics A – H245 specification.](https://www.ocr.org.uk/Images/308752-specification-accredited-a-level-gce-further-mathematics-a-h245.pdf)

The table headings are explained below:

| **OCR Reference.** | **Content Description**  (unshaded content is AS content) | **R** | **A** | **G** | **Notes** |
| --- | --- | --- | --- | --- | --- |
| Each item of content has a unique specification reference code.   * **Mathematics A – H240**  1. Mathematics A: Pure 2. Mathematics A: Statistics 3. Mathematics A: Mechanics  * **Further Mathematics A – H245**  1. Further Mathematics A: Pure Core 2. Further Mathematics A: Statistics Option 3. Further Mathematics A: Mechanics Option 4. Further Mathematics A: Discrete Option 5. Further Mathematics A: Additional Pure | | You can use the tick boxes to show when you have revised an item and how confident you feel about it.  R = **RED** means you are really unsure and lack confidence; you might want to focus your revision here and possibly talk to your teacher for help  A = **AMBER** means you are reasonably confident but need some extra practice  G = **GREEN** means you are very confident.  As your revision progresses, you can concentrate on the **RED** and **AMBER** items in order to turn them into **GREEN** items.  You might find it helpful to highlight each topic in red, orange or green to help you prioritise. | | | You can use the notes column to:   * add more information about the details for each point * add formulae or notes * include a reference to a useful resource. * Highlight areas of difficulty or things that you need to talk to your teacher about or look up in a textbook. |

You must be able to use all the formulae and identities given for the Pure Core mandatory strand of A Level Further Mathematics, without those formulae and identities being provided, either in these forms or in equivalent forms. Those formulae and identities may only be provided where they are the starting point for a proof or as a result to be proved.

Learners will be given a Formulae Booklet in each assessment which has both the A Level Mathematics and the A Level Further Mathematics formulae (the version used for AS has only the AS Maths and Further Maths formulae).

**Discrete Mathematics option formulae**

**Inclusion-exclusion principle**For sets *A*, *B* and *C*:



**The hierarchy of orders**

**Sorting algorithms**  
Bubble sort:

Start at the left hand end of the list unless specified otherwise.

Compare the first and second values and swap if necessary. Then compare the (new) second value with the third value and swap if necessary. Continue in this way until all values have been considered.

Fix the last value then repeat with the reduced list until either there is a pass in which no swaps occur or the list is reduced to length 1, then stop.

Shuttle sort:

Start at the left hand end of the list unless specified otherwise.

Compare the second value with the first and swap if necessary, this completes the first pass. Next compare the third value with the second and swap if necessary, if a swap happened shuttle back to compare the (new) second with the first as in the first pass, this completes the second pass.

Next compare the fourth value with the third and swap if necessary, if a swap happened shuttle back to compare the (new) third value with the second as in the second pass (so if a swap happens shuttle back again). Continue in this way for  passes, where *n* is the length of the list.

Quick sort:

The first value in any sublist will be the pivot, unless specified otherwise.

Working from left to right, write down each value that is smaller than the pivot, then the pivot, then work along the list and write down each value that is not smaller than the pivot. This produces two sublists (one of which may be empty) with the pivot between them and completes the pass.

Next apply this procedure to each of the sublists from the previous pass, unless they consist of a single entry, to produce further sublists. Continue in this way until no sublist has more than one entry.

**Network algorithms**  
Dijkstra’s algorithm

START with a graph *G*. At each vertex draw a box, the lower area for temporary labels, the upper left hand area for the order of becoming permanent and the upper right hand area for the permanent label.

STEP 1 Make the given start vertex permanent by giving it permanent label 0 and order label 1.

STEP 2 For each vertex that is not permanent and is connected by an arc to the vertex that has just been made permanent (with permanent label ), add the arc weight to P. If this is smaller than the best temporary label at the vertex, write this value as the new best temporary label.

STEP 3 Choose the vertex that is not yet permanent which has the smallest best temporary label. If there is more than one such vertex, choose any one of them. Make this vertex permanent and assign it the next order label.

STEP 4 If every vertex is now permanent, or if the target vertex is permanent, use ‘trace back’ to find the routes or route, then STOP; otherwise return to STEP 2.

Prim’s algorithm (graphical version)

START with an arbitrary vertex of *G*.

STEP 1 Add an edge of minimum weight joining a vertex already included to a vertex not already included.

STEP 2 If a spanning tree is obtained STOP; otherwise return to STEP 1.

Prim’s algorithm (tabular version)

START with a table (or matrix) of weights for a connected weighted graph.

STEP 1 Cross through the entries in an arbitrary row, and mark the corresponding column.

STEP 2 Choose a minimum entry from the uncircled entries in the marked column(s).

STEP 3 If no such entry exists STOP; otherwise go to STEP 4.

STEP 4 Circle the weight *wij* found in STEP 2; mark column *j* ; cross through row *i*.

STEP 5 Return to STEP 2.

Kruskal’s algorithm

START with all the vertices of *G*, but no edges; list the edges in increasing order of weight.

STEP 1 Add an edge of *G* of minimum weight in such a way that no cycles are created.

STEP 2 If a spanning tree is obtained STOP; otherwise return to STEP 1.

Nearest neighbour method

START at a given vertex of *G*.

STEP 1 Find the least weight arc from this vertex to a vertex that has not already been included (or back to the start vertex if every vertex has been included).

STEP 2 If no such arc exists then the method has stalled STOP; otherwise add this arc to the path.

STEP 3 If a cycle has been found STOP; otherwise return to STEP 1.

Lower bound for travelling salesperson problem

START with all vertices and arcs of *G*.

STEP 2 Remove a given vertex and all arcs that are directly connected to that vertex, find a minimum spanning tree for the resulting reduced network.

STEP 3 Add the weight of this minimum connector to the sum of the two least weight arcs that had been deleted. This gives a lower bound.

Route inspection problem

START with a list of the odd degree vertices.

STEP 1 For each pair of odd nodes, find the connecting path of least weight.

STEP 2 Group the odd nodes so that the sum of weights of the connecting paths is minimised.

STEP 3 Add this sum to the total weight of the graph STOP.

**The simplex algorithm**  
 START with a tableau in standard format.

STEP 1 Choose a column with a negative entry in the objective row (or zero in degenerate   
 cases).

STEP 2 The pivot row is the one for which non-negative value of the entry in the final column divided by the positive value of the entry in the pivot column is minimised. The pivot element is the entry of the pivot row in the chosen column.

STEP 3 Divide all entries in the pivot row by the value of the pivot element.

STEP 4 Add to, or subtract from, all other old rows a multiple of the new pivot row, so that the pivot column ends up consisting of zeroes and a single one, and corresponds to the new basic variable.

STEP 5 If the objective row has no negative entries STOP; otherwise return to STEP 1.

## Content of Discrete Mathematics (Optional paper Y544)

| **OCR Reference** | **Content Description**  (unshaded content is AS content) | **R** | **A** | **G** | **Notes** |
| --- | --- | --- | --- | --- | --- |
| **7.01a** | a) Understand and be able to use the terms “existence”, “construction”, “enumeration” and “optimisation” in the context of problem solving.  *Includes classifying a given problem into one or more of these categories.* |  |  |  |  |
| **7.01b** | b) Understand and be able to use the basic language and notation of sets.  *Including the term “partition” and counting the number of partitions of a set including with constraints.* |  |  |  |  |
| **7.01c** | c) Be able to use the pigeonhole principle in solving problems. |  |  |  |  |
| **7.01d** | d) Understand and use the multiplicative principle.  Includes knowing that the number of arrangements of *n* distinct objects is . |  |  |  |  |
| **7.01e** | e) Be able to enumerate the number of ways of obtaining an ordered subset (permutation) of *r* elements from a set of *n* distinct elements.  *Includes using the notation* . |  |  |  |  |
| **7.01f** | f) Be able to enumerate the number of ways of obtaining an unordered subset (combination) of *r* elements from a set of *n* distinct elements.  *Includes using the notation .* |  |  |  |  |
| **7.01g** | g) Be able to solve problems about enumerating the number of arrangements of objects in a line, including those involving:    1. repetition, *e.g. how many different eight digit*  *numbers can be made from the digits of*  *12333210?*  2. restriction, *e.g. how many different eight digit*  *numbers can be made from the digits of*  *12333210 if the two 2s cannot be next to each*  *other?* |  |  |  |  |
| **7.01h** | h) Be able to solve problems about enumerating the number of arrangements of only some of a group of objects.  *e.g. how many different four digit numbers can be made from the digits of 12333210?*  *e.g. how many different numbers greater than 20 000 can be made from the digits of 12333210?* |  |  |  |  |
| **7.01i** | i) Be able to solve problems about selections, including with constraints.  *e.g. Find the number of ways in which a team of 3 men and 2 women can be selected from a group of 6 men and 5 women.* |  |  |  |  |
| **7.01j** | j) Be able to solve problems with several constraints.  *e.g. Given a graph showing who dislikes who, find the number of ways of choosing 3 men and 2 women from a group of 4 men and 4 women so that no two people chosen dislike each other.* |  |  |  |  |
| **7.01k** | k) Be able to use the inclusion-exclusion principle for two sets in solving problems.  *e.g.*  *Venn diagrams may be used.*  *e.g. How many integers in are not divisible by 2, 3 or 5?* |  |  |  |  |
| **7.01l** | l) Be able to extend the inclusion-exclusion principle to more than two sets.  *e.g.*  *Venn diagrams may be used.*  *e.g. How many integers in are not divisible by 2, 3 or 5?* |  |  |  |  |
| **7.01m** | m) Be able to find derangements.  *Includes enumeration of the number of derangements of  objects, , by ad hoc methods only.* |  |  |  |  |
| **7.02a** | a) Understand the meaning of the terms vertex (or node) and arc (or edge).  *Includes the concept of the “degree” of a vertex as the number of arcs “incident” to the vertex.*  *Includes the term “adjacent” for pairs of vertices or edges.* |  |  |  |  |
| **7.02b** | b) Understand the meaning of the terms “tree”, “simple”, “connected” and “simply connected” as they refer to graphs.  *Includes understanding and using the restrictions on the vertex degrees implied by these conditions.* |  |  |  |  |
| **7.02c** | c) Understand the meaning of the terms “walk”, “trail”, “path”, “cycle” and “route”.  *A “walk” is a set of arcs where the end vertex of one is the start vertex of the next.*  *A “trail” is a walk in which no arcs are repeated.*  *A “path” is a trail in which no nodes are repeated.*  *A “cycle” is a closed path.*  *A “route” can be a walk, a trail or a path, or may be a closed walk, trail or path.* |  |  |  |  |
| **7.02d** | d) Understand and be able to use the term “complete” and the notation  for a complete graph on *n* vertices.  *Includes knowing that  has*  *arcs.* |  |  |  |  |
| **7.02e** | e) Understand and be able to use bipartite graphs and the notation  for a complete bipartite graph connecting *m* vertices to *n* vertices.  *Includes knowing that  has mn arcs.* |  |  |  |  |
| **7.02f** | f) Use a colouring argument to show that a given graph is, or is not, bipartite. |  |  |  |  |
| **7.02g** | g) Use the degrees of vertices to determine whether a given graph is Eulerian, semi-Eulerian or neither. Understand what these terms mean in terms of traversing the graph. |  |  |  |  |
| **7.02h** | h) Understand and be able to use the definition of a Hamiltonian path, a Hamiltonian cycle and a Hamiltonian graph. |  |  |  |  |
| **7.02i** | i) Know and use Ore’s theorem.  *i.e. For a simple graph  with  vertices, if  for every pair of distinct non-adjacent vertices  and , then  is Hamiltonian.*  *Includes understanding that Ore’s theorem gives a sufficient but not necessary condition for a graph to be Hamiltonian.* |  |  |  |  |
| **7.02j** | j) Understand what it means to say that two graphs are isomorphic. Construct an isomorphism either by a reasoned argument or by explicit labelling of vertices.  *Includes understanding that having the same degree sequence (ordered list of vertex degrees) is necessary but not sufficient to show isomorphism.*  *Includes the term non-isomorphic.* |  |  |  |  |
| **7.02k** | k) Understand and be able to use digraphs.  *Includes the terms “indegree” and “outdegree”.* |  |  |  |  |
| **7.02l** | l) Understand and be able to apply the concepts of planarity, subdivision and contraction.  *i.e. Subdivision is inserting a vertex of degree 2 into an arc. Contraction is contracting two vertices into one so that any arc incident with the original two vertices is incident with the contracted vertex.*  *Includes the notation  for the vertex created by the contraction of the arc AB.*  *Includes drawing planar representations.* |  |  |  |  |
| **7.02m** | m) Know, understand and use Euler’s formula . |  |  |  |  |
| **7.02n** | n) Know and use Kuratowski’s theorem.  *i.e. A finite graph is planar if and only if it does not contain a subgraph that is a subdivision of  or  .* |  |  |  |  |
| **7.02o** | o) Understand and be able to use the concept of thickness.  [*Calculation of thickness  is excluded.*] |  |  |  |  |
| **7.02p** | p) Understand that a network is a weighted graph. Use graphs and networks to model the connections between objects.  *Graphs and networks may be directed or undirected.* |  |  |  |  |
| **7.02q** | q) Use an adjacency matrix representation of a graph and a weighted matrix representation of a network. |  |  |  |  |
| **7.02r** | r) Be able to model problems using graphs or networks, and solve them. |  |  |  |  |
| **7.03a** | a) Understand that an algorithm has an input and an output, is deterministic and finite.  *Includes the use of a counter and the use of a stopping condition in an algorithm.*  *Be familiar with the terms greedy, heuristic and recursive in the context of algorithms.* |  |  |  |  |
| **7.03b** | b) Appreciate why an algorithmic approach to problem solving is generally preferable to ad hoc methods, and understand the limitations of algorithmic methods.  *Includes understanding that algorithmic methods are used by computers for solving large scale problems and that small scale problems are only being used to demonstrate how a given algorithm works.* |  |  |  |  |
| **7.03c** | c) Trace through an algorithm and interpret what the algorithm has achieved. Algorithms may be presented as flow diagrams, listed in words, or written in simple pseudo-code.  *Includes understanding and being able to use the functions  and  Learners may find it useful to have a calculator with these functions, but large numbers of repeated applications will not be required in the assessment.*  *Includes adapting or altering an algorithm to achieve a given purpose, and adjusting a short set of instructions to create an algorithm.*  [*Programming skills will not be required*.] |  |  |  |  |
| **7.03d** | d) Use the order of an algorithm to calculate an approximate run-time for a large problem by scaling up a given run-time.  *Includes understanding that when the “maximum run-time” of an algorithm is represented as a function of the “size” of the problem, the order of the algorithm, for very large sized problems, is given by the dominant term.*  *Learners should know that the sum of the first n positive integers is* .  *Learners should be familiar with the notation  and the concept of dominance in an informal sense only.* |  |  |  |  |
| **7.03e** | e) Compare the efficiency of two algorithms that achieve the same end result by considering a given aspect of the run-time in a specific case.  *e.g. The number of swaps or comparisons to sort a given list*. |  |  |  |  |
| **7.03f** | f) Calculate worst case time complexity, the “maximum run-time” , as a function of the size of a problem by considering the worst case for a specific problem.  *Includes cases of the algorithms for sorting and standard network problems studied in this specification*.  *Includes an informal understanding that, for example*  *is* order  , or equivalently . |  |  |  |  |
| **7.03h** | h) Calculate the run-time as a function of the size of a problem by considering the best case, the worst case or a typical case.  *Includes c*o*nsidering all cases and averaging where appropriate*. |  |  |  |  |
| **7.03g** | g) Be familiar with , where *n* is a measure of the size of the problems and  or . |  |  |  |  |
| **7.03i** | i) Be familiar with:  for ,  for ,  where *n* is a measure of the size of the problem.  *Know the hierarchy of orders and what this means in terms of efficiency. Learners should be aware that:*  , *which is given in the Formulae Booklet.* |  |  |  |  |
| **7.03j** | j) Be able to sort a list using bubble sort and using shuttle sort.  *Bubble sort and shuttle sort will start at the left-hand end of the list, unless specified otherwise in the question.*  *Includes knowing that, in general, sorting algorithms have quadratic order as a function of the length of the list.* |  |  |  |  |
| **7.03k** | k) Be able to sort a list using quick sort.  *Quick sort will pivot on the first value, unless specified otherwise in a question.*  *Includes knowing that quick sort is only*  *in the worst case.*  *Questions may be set that interrogate the application of the method, for example whether the choice of pivot affects the efficiency of quicksort.* |  |  |  |  |
| **7.03l** | l) Be familiar with the next-fit, first-fit, first-fit decreasing and full bin methods for one-dimensional packing problems.  *Includes knowing that these are heuristic algorithms.*  *Includes the terms “online” and “offline”.* |  |  |  |  |
| **7.03m** | m) Extend their knowledge of packing methods.  *e.g. Packing problems in two or three dimensions.*  *e.g. Knapsack problems: given a set of items, each with a mass and a profit, determine which to include so that the total mass does not exceed some given limit and the total profit is as large as possible.* |  |  |  |  |
| **7.04a** | a) Be able to use examples to demonstrate understanding and use of Dijkstra’s algorithm to find the length and route of a least weight (shortest) path.  *Solve problems that require a least weight (shortest) path as part of their solution.*  *Know that Dijkstra’s algorithm has quadratic order (as a function of the number of vertices).* |  |  |  |  |
| **7.04b** | b) Be able to use examples to demonstrate understanding and use of Prim’s algorithm (both in graphical and tabular/matrix form) and Kruskal’s algorithm to find a minimum connector (minimum spanning tree) for a network.  *Solve problems that require a minimum spanning tree as part of their solution.*  *Includes adapting a solution to deal with practical issues.*  *Know that Prim’s algorithm and Kruskal’s algorithm have cubic order (as a function of the number of vertices).* |  |  |  |  |
| **7.04c** | c) Be able to use the nearest neighbour method on a network formed by weighting a complete graph to find an upper bound for the travelling salesperson problem.  *Understand that when the nearest neighbour method is used on a network formed by weighting a graph that is not complete it may stall before reaching every vertex or it may reach every vertex but to close the route may need a path that is not a direct connection from the end back to the start vertex.*  *Includes choosing between two, or more, upper bounds to find the best (least) upper bound.*  *Includes using short-cuts where possible to improve an upper bound.* |  |  |  |  |
| **7.04d** | d) Be able to use a minimum spanning tree on a reduced network to calculate a lower bound for the travelling salesperson problem on a complete graph and understand why this method gives a lower bound.  *Understand that for a graph that is not complete this method can give a value that is not a lower bound.*  *Includes choosing between two or more lower bounds to find the best (greatest) lower bound.* |  |  |  |  |
| **7.04e** | e) Be able to solve the route inspection problem by consideration of all possible pairings of up to six odd nodes.  *If a problem has more than six odd nodes additional restrictions will reduce the number of pairings that need to be considered.*  *Problems may be set that require an understanding of how the number of pairings increases as the number of odd nodes increases. For  odd nodes the number of pairings is* . |  |  |  |  |
| **7.04f** | f) Be able to choose an appropriate algorithm to solve a practical problem.  *Includes adapting an algorithm or a solution to deal with practical issues.* |  |  |  |  |
| **7.05a** | a) Be able to construct and interpret activity networks using activity onarc.  *Appreciate that a path of critical activities (a critical path) is a longest path in a directed network.* |  |  |  |  |
| **7.05b** | b) Be able to carry out a forward pass to determine earliest start times and find the minimum project completion time, and to carry out a backward pass to determine latest finish times and find the critical activities.  *Includes understanding and using the terms burst and merge.* |  |  |  |  |
| **7.05c** | c) Understand, and be able to calculate, (total) float. |  |  |  |  |
| **7.05d** | d) Be able to find latest start times and earliest finish times for activities. Calculate and interpret independent and interfering float. |  |  |  |  |
| **7.05e** | e) Be able to use an activity network to construct a cascade chart and use a cascade chart to determine the effect on the minimum project completion time of a delay to one or more activities, or other scheduling restrictions.  *Cascade charts may be constructed either with one activity on each row or with the critical activities together in one row and a row for each non-critical activity.*  *Float may be shown using dashed lines.*  *Includes constructing a schedule to show how a given number of workers can complete a project subject to given constraints.* |  |  |  |  |
| **7.06a** | a) Be able to set up a linear programming formulation in the form “maximise (or minimise) objective subject to inequality constraints, and trivial constraints of the form variable”.  *Includes:*  *1. identifying relevant variables, including units*  *when appropriate,*  *2. formulating constraints in these variables,*  *including when the information is given in ratio*  *form,*  *3. writing down an objective function and stating*  *whether it is to be maximised or minimised.* |  |  |  |  |
| **7.06b** | b) Be able to use slack variables to convert inequality constraints, each being  a non-negative constant, into equations, together with further trivial constraints of the form variable . |  |  |  |  |
| **7.06c** | c) Be able to investigate constraints and objectives in numerical cases using algebra and ad hoc methods. |  |  |  |  |
| **7.06d** | d) Be able to carry out and interpret a graphical solution for problems where the objective is a function of two variables, including cases where integer solutions are required.  *The region where each inequality is not satisfied will be shaded, leaving the feasible region as the unshaded part of the graph.* |  |  |  |  |
| **7.06e** | e) Be able to discuss the effect of making a change to one or two of the coefficients and how this will change the solution. |  |  |  |  |
| **7.06f** | f) Be able to carry out integer programming, including using the branch-and-bound method. |  |  |  |  |
| **7.07a** | a) Be able to set up an initial simplex tableau in standard format.  *Rows to show objective to be maximised, followed by constraints. Columns to represent objective, variables and then slack variables, with the column representing the right-hand side as the last column*. |  |  |  |  |
| **7.07b** | b) Be able to perform an iteration of a simplex tableau, including the choice of the pivot.  *The column with the most negative value in the objective row should be chosen, unless other specific instruction is given.* |  |  |  |  |
| **7.07c** | c) Be able to interpret the values of the variables, slack variables and objective at any stage and know when the optimum has been achieved.  *Includes discussing the effect of changes to the coefficients.* |  |  |  |  |
| **7.07d** | d) Be able to use the terms “basic feasible solution”, “basic variable” and “non-basic variable” appropriately.  *Basic variables correspond to columns consisting of zeroes together with a single 1*, *the other variables are non-basic.* |  |  |  |  |
| **7.07e** | e) Understand what an iteration of the simplex algorithm achieves, in terms of moving along edges of a multi-dimensional convex polygon, usually in two or three dimensions, and be able to apply this to problems. |  |  |  |  |
| **7.07f** | f) Be able to explain, algebraically, some of the calculations used in the simplex algorithm. |  |  |  |  |
| **7.08a** | a) Understand the idea of a zero-sum game and its representation by means of a pay-off matrix.  *Includes converting a game to a zero-sum form, where appropriate.* |  |  |  |  |
| **7.08b** | b) Be able to reduce a matrix using a dominance argument. |  |  |  |  |
| **7.08c** | c) Be able to identify play-safe strategies and stable solutions and understand what they represent. |  |  |  |  |
| **7.08d** | d) Be able to identify a Nash Equilibrium solution and understand what it represents. |  |  |  |  |
| **7.08e** | e) Be able to determine an optimal mixed strategy for a game with no stable solution by reducing to two variables and using simultaneous equations or a graphical method, where possible.  *Includes knowing that the optimum may occur at an extreme value  or .* |  |  |  |  |
| **7.08f** | f) Be able to determine an optimal mixed strategy for a game with no stable solution by reformulating the problem as a linear programming problem that could be solved using the simplex algorithm. |  |  |  |  |



We’d like to know your view on the resources we produce. Click ‘[Like’](mailto:resources.feedback@ocr.org.uk?subject=I%20liked%20the%20A%20Level%20Further%20Mathematics%20A%20-%20Discrete%20Y544%20Student%20revision%20checklist) or ‘[Dislike’](mailto:resources.feedback@ocr.org.uk?subject=I%20disliked%20the%20A%20Level%20Further%20Mathematics%20A%20-%20Discrete%20Y544%20Student%20revision%20checklist) to send us an auto generated email about this resource. Add comments if you want to. Let us know how we can improve this resource or what else you need. Your email will not be used or shared for any marketing purposes.

Looking for another resource? There is now a quick and easy search [tool to help find free resources](https://www.ocr.org.uk/qualifications/resource-finder/) for your qualification.

OCR is part of Cambridge University Press & Assessment, which is itself a department of the University of Cambridge.

For staff training purposes and as part of our quality assurance programme your call may be recorded or monitored.   
Oxford Cambridge and RSA Examinations is a Company Limited by Guarantee. Registered in England. Registered office   
The Triangle Building, Shaftesbury Road, Cambridge, CB2 8EA. Registered company number 3484466. OCR is an exempt charity.

OCR operates academic and vocational qualifications regulated by Ofqual, Qualifications Wales and CCEA as listed in their qualifications registers including A Levels, GCSEs, Cambridge Technicals and Cambridge Nationals.

OCR provides resources to help you deliver our qualifications. These resources do not represent any particular teaching method we expect you to use. We update our resources regularly and aim to make sure content is accurate but please check the OCR website so that you have the most up to date version. OCR cannot be held responsible for any errors or omissions in these resources.

Though we make every effort to check our resources, there may be contradictions between published support and the specification, so it is important that you always use information in the latest specification. We indicate any specification changes within the document itself, change the version number and provide a summary of the changes. If you do notice a discrepancy between the specification and a resource, please [contact us](mailto:resources.feedback@ocr.org.uk).

© OCR 2021 - You can copy and distribute this resource freely if you keep the OCR logo and this small print intact and you acknowledge OCR as the originator of the resource.

OCR acknowledges the use of the following content: N/A

Whether you already offer OCR qualifications, are new to OCR or are thinking about switching, you can request more information using our [Expression of Interest form](https://www.ocr.org.uk/qualifications/expression-of-interest/).

Please [get in touch](mailto:resources.feedback@ocr.org.uk) if you want to discuss the accessibility of resources we offer to support you in delivering our qualifications.