Definitions and Concepts for AQA Computer Science A-level

Topic 4: Theory of Computation

4.1 Abstraction and Automation

4.1.1 Problem Solving

Logical Reasoning: The use of a set of facts (axioms) to draw conclusions and determine whether new information is true or false.

4.1.2 Following and Writing Algorithms

Algorithms: A sequence of steps that can be followed to complete a task and that always terminates.✢

Correctness: An algorithm is said to be correct when it is consistent with the specification and produces the expected output for any given input.

Efficiency: A property of an algorithm that is related to the amount of resources (memory space and time in particular) that an algorithm uses in its execution.

Hand-tracing: The process of looking at a program's entire code or code extract and running through the instructions as though you are the computer.

Pseudo-code: A human-readable method of writing the steps of an algorithm without any particular programming language.

4.1.3 Abstraction

Abstraction by Generalisation or Categorisation: Simplifying a problem by grouping together common characteristics of a problem to arrive at a hierarchical relationship.

Representational Abstraction: Simplifying a problem by only taking into consideration the necessary details required to obtain a solution, leaving a representation without any unnecessary details.

4.1.4 Information Hiding

Information Hiding: The process of hiding all details of an object that do not contribute to its essential characteristics.✢

4.1.5 Procedural Abstraction

Procedural Abstraction: Simplifying a problem by breaking it down into a series of procedures or subroutines that are generalised with variable parameters. Knowledge of the implementation of each procedure is irrelevant.

Procedure: The result of abstracting away the actual values used in any particular computation is a computational pattern or computational method.✢

4.1.6 Functional Abstraction

Functional Abstraction: Simplifying a problem by breaking it down into a series of reusable functions which disregard the particular computational method.

4.1.7 Data Abstraction

Data Abstraction: The storage and representation of data in a computer system along with its logical description and interaction with operators. This allows the construction of new compound data objects from existing ones.

Data Objects: Data abstractions that hide details of how data are actually represented from the user.

4.1.8 Problem Abstraction/Reduction

Problem Abstraction/Reduction: The repeated removal of unnecessary details from a problem until an underlying problem representation is reached which is identical to a previously solved problem.

4.1.9 Decomposition

Procedural Decomposition: The process of breaking down a problem into a number of sub-problems, so that each sub-problem accomplishes an identifiable task, which might itself be further subdivided.✢

4.1.10 Composition

Composition Abstraction: The reverse process of decomposition where a complex system of compound procedures is built from its smaller, simpler procedures.

Data Abstraction Composition: The process of combining data objects to form compound data.✢

4.1.10 Automation

Automation: The process of creating algorithms and implementing them as data structures and models of real-life situations that run without a significant need for human intervention.

4.2 Regular Expressions

4.2.1 Finite State Machines (FSMs) With and Without Output

Accepting States: An optional state of a FSM that indicates whether or not an input has been accepted by the FSM.

Finite State Machines: A model of computation for a machine that is always in one of a fixed number of states.◊

Mealy Machines: A finite state machine that determines its outputs from the present state and from the inputs.◊

State Transition Diagrams: A visual representation of a FSM that uses circles to represent states and arrows to indicate transitions between states.

State Transition Tables: A tabular representation of a FSM that contains the current state, inputs and their consequent successor state.

4.2.2 Maths for Regular Expressions

Cardinality of a Finite Set: The number of elements in the set.✢

Cartesian Product of Sets: The set of all ordered pairs (a, b) where a is a member of the first set, A, and b is a member of the second set, B.✢

Countable Sets: A set with the same cardinality as some subset of natural numbers.✢

Countably Infinite Sets: A set that can be counted off by the natural numbers.✢

Difference: An operator that produces a set containing all the elements present in either of the initial operand sets, but not both.

Empty Sets: A set with no elements. Represented by {} or Ø.

Finite Sets: A set whose elements can be counted off by natural numbers up to a particular number.✢

Infinite Sets: A set that is not finite.

Intersection: An operator that produces a set containing only the elements present in both initial operand sets.

Membership: The property of an element being within a set.

Proper Subsets: A set that is fully contained in another set, and the other set contains elements that are not present in the proper subset.

Set Comprehension: The creation of a set by mathematically defining the elements that qualify to be in the set, rather than listing out all its elements.

Sets: An unordered collection of values in which each value occurs at most once.✢

Subsets: A set such that all its elements are present in the other set being considered.

Union: An operator that produces a set containing all the elements from both initial operand sets.

4.2.3 Regular Expressions

Regular Expressions: A way of describing sets and the elements present within it using strings of characters.

4.2.4 Regular Language

Regular Languages: A language that can be represented by a regular expression.✢

4.3 Context-Free Languages

4.3.1 Backus-Naur Form (BNF)/Syntax Diagrams

Backus-Naur Form (BNF): A notation technique to express syntax of languages in computing.

4.4 Classification of Algorithms

4.4.1 Comparing Algorithms

Space Complexity: A measure of the amount of memory space needed by an algorithm to solve a particular problem of a given input size.

Time Complexity: A measure of the amount of time needed by an algorithm to solve a particular problem of a given input size.

4.4.2 Maths for Understanding Big-O Notation

Functions: A mapping from one set of values, the domain, to another set of values, drawn from the co-domain.✢

Permutations: An arrangement of objects such that their ordering matters.

4.4.3 Order of Complexity

O(1) (Constant time/space): An algorithm that always takes a constant amount of time or memory space to execute, regardless of the input size.

O(2^n) (Exponential time/space): An algorithm whose execution time or required memory space grows exponentially with input size (higher complexity than polynomial).

O(log(n)) (Logarithmic time/space): An algorithm whose execution time or required memory space grows logarithmically with input size (lower complexity than polynomial).

O(n) (Linear time/space): An algorithm whose execution time or required memory space is directly proportional to the size of its input.

O(n^k) (Polynomial time/space): An algorithm whose execution time or required memory space is proportional to the input size raised to the power of a constant (k). Eg. O(n^2), O(n^3) etc.

4.4.5 Classification of Algorithmic Problems

Intractable Problems: Problems that have no polynomial (or less) time solution. ✢

Tractable Problems: Problems that have a polynomial (or less) time solution. ✢

4.4.6 Computable and Non-Computable Problems

Computable Problems: Problems that can be solved algorithmically. ✢

Non-Computable Problems: Problems that cannot be solved algorithmically. ✢

4.4.7 Halting Problem

The Halting Problem: The unsolvable problem of determining whether any program will eventually stop if given particular input. ✢

4.5 A Model of Computation

4.5.1 Turing Machine

Halting State: A state with no outgoing transitions that halts a Turing Machine.

Start State: A Turing Machine's initial starting state.

Transition Function: A function that determines how a Turing Machine moves from one state to the other. It is equivalent to a state transition diagram (see 4.2.1).

Turing Machine: A formal model of computation that consists of a finite state machine that controls one or more tapes, where at least one tape is of unbounded length (ie infinitely long).◊

Universal Turing Machine: An interpreter that can simulate any Turing machine by reading the description of any arbitrary Turing Machine and faithfully executing operations on data precisely as the machine would.◊

Definitions with a '✢' taken from [AQA AS and A-level Computer Science specification version 1.5](#)
Definitions with a '◊' taken from [AQA AS and A-level Computer Science subject specific vocabulary](#) (last accessed 24th April 2021)