Curriculum Map: Computing, Year 13

(All topics of year 12 are also tested at end of Year 13)

	Autumn	Spring and Summer
Content Declarative knowledge 'I Know'	**NEA main body of work 1.1.1 Structure and function of the processor 1.1.2 Types of processors 1.2.4 Types of Programming Language 2.2.1 Programming techniques 1.3.2 Databases 1.4.1 Data Types 2.3.1 Algorithms	**NEA- finish 1.3.1 Compression, Encryption and Hashing 2.3.1 Algorithms 1.4.2 Data Structures 1.3.4 Web Technologies 1.2.2 Applications Generation 1.4.3 Boolean Algebra 2.1.5 Thinking concurrently 2.2.2 Computational methods Exam preparation
Skills Procedural Knowledge 'I know how to'	**** NEA main body of work** • Design phase • Development & Testing phase – 3 iterations **1.1.1 Structure and function of the processor (Year 13 topics)** d) The use of pipelining in a processor to improve efficiency. **1.1.2 Types of processor (Year 13 topics)** b) GPUs and their uses (including those not related to graphics). **1.3.3 Networks (Year 13 topics)** c) Network security and threats, use of firewalls, proxies and encryption. d) Network hardware. **1.2.4 Types of Programming Language** a) Need for and characteristics of a variety of programming paradigms. b) Procedural languages. *c) Assembly language (including following and writing simple programs with the Little Man Computer instruction set). (covered in year 12)* d) Modes of addressing memory (immediate, direct, indirect and indexed). *e) Object-oriented languages with an understanding of classes, objects, methods, attributes, inheritance, encapsulation and polymorphism (consolidation, as covered in year 12)* **2.2.1 Programming techniques (Year 13 topics)** *b) Recursion, how it can be used and compares to an iterative approach. (consolidation, as (covered in year 12)* f) Use of object-oriented techniques (*consolidation, as (covered in year 12)*	****NEA- finish** • Beta testing • Evaluation **2.3.1 Algorithms (last topics)** f) Standard algorithms (Merge sort, Quick sort, **Dijkstra's shortest path algorithm, A* algorithm**). **1.3.1 Compression, Encryption and Hashing** (a) Lossy vs Lossless compression. (b) Run length encoding and dictionary coding for lossless compression. (c) Symmetric and asymmetric encryption. (d) Different uses of hashing. **1.4.2 Data Structures (Year 13 topics)** (b) The following structures to store data: linked-list, graph (directed and undirected), stack, queue, **tree, binary search tree, hash table**. (c) How to create, traverse, add data to and remove data from the data structures mentioned above. **1.3.4 Web Technologies (Year 13 topics)** (b) Search engine indexing. (c) PageRank algorithm. (d) Server and client-side processing. **1.2.2 Applications Generation (Year 13 topics)** (d) Translators: Interpreters, compilers and assemblers. (e) Stages of compilation (lexical analysis, syntax analysis, code generation and optimisation). (f) Linkers and loaders and use of libraries.

	1.3.2 Databases (Year 13 topics) (c) Normalisation to 3NF. (d) SQL – Interpret and modify. (e) Referential integrity. (f) Transaction processing, ACID (Atomicity, Consistency, Isolation, Durability), record locking and redundancy. **1.4.1 Data Types (Year 13 topics)** h) Floating point arithmetic, positive and negative numbers, addition and subtraction. i) Bitwise manipulation and masks: shifts, combining with AND, OR, and XOR. **2.3.1 Algorithms (Year 13 topics)** (a) Analysis and design of algorithms for a given situation. (b) The suitability of different algorithms for a given task and data set, in terms of execution time and space. (c) Measures and methods to determine the efficiency of different algorithms, Big O notation (constant, linear, polynomial, exponential and logarithmic complexity). (d) Comparison of the complexity of algorithms. (e) Algorithms for the main data structures, (stacks, queues, trees, linked lists, depth-first (post-order) and breadth-first traversal of trees). f) Standard algorithms (**Merge sort, Quick sort**, Dijkstra's shortest path algorithm, A* algorithm).	**1.4.3 Boolean Algebra (Year 13 topics)** (c) Use the following rules to derive or simplify statements in Boolean algebra: De Morgan's Laws, distribution, association, commutation, double negation. (e) The logic associated with D type flip flops, half and full adders. **2.1.5 Thinking concurrently** (a) Determine the parts of a problem that can be tackled at the same time. (b) Outline the benefits and trade-offs that might result from concurrent processing in a particular situation. **2.2.2 Computational methods** (a) Features that make a problem solvable by computational methods. (b) Problem recognition. (c) Problem decomposition. (d) Use of divide and conquer. (e) Use of abstraction. (f) Learners should apply their knowledge of: •• backtracking •• data mining •• heuristics •• performance modelling •• pipelining •• visualisation to solve problems.
Strategies Conditional Knowledge 'I know when to'	Why and which tasks are GPUS suited for? What kind of architectures support parallel processing? How to minimise, or preventing threats to system security / cybercrime? How to use indexing to optimise the searching for data? How problems that arise from transaction processing can be reduced using ACID rules and record locking. What kind of programming problems lend themselves to the object-oriented paradigm and which one lend themselves to procedural? How the best case, average case and worst-case complexities of algorithms can be used to decide which ones to se for given scenarios and data structures.	Justify the best data structures to use for given problems based on their behaviour and traversal mechanisms. Whether to use lossy or lossless compression for given media type (e.g. photos, videos, audio etc.) Whether run length or dictionary encoding is best for a given scenario. Whether hashing or encryption is the best protection technique for a given detail that needs to be stored. Be able to recommend the benefits and trade-offs that are brought from concurrent processing, when applied to different scenarios
Key Questions	Understand the purpose of GPUs and what applications they are used for (candidates need to understand how GPUs are used to aid graphics, but also other applications for example their use in modelling, data mining, etc.). Candidates should understand the benefits and using GPUs and why they are suited to certain tasks (specialist instructions, multiple cores and SIMD processing).	Understand the behaviour of linked-lists, graphs, stacks, queues, trees, binary search trees and hash tables. Be aware of how the aforementioned data structures can be implemented. We would recommend a general understanding of these principles that can be applied to a given scenario rather than trying to memorise code patterns. Candidates should have experience of implementing these structures in a variety of contexts, for example through a procedural program, through a different data structure and

	Understand that parallel processing can be achieved through different (i.e. multiple processors in the same computer or distributed across multiple cores in a CPU or GPU). What are the security issues and threats involved with networked computers? Candidates need to be aware of threats such as hackers, viruses, unauthorised access, denial of service, spyware, SQL injection, phishing and pharming. Candidates need to know about ways of minimising, or preventing these threats for example firewalls, secure passwords, anti-virus, anti-spyware etc. What are the different hardware required to connect to and/or build a network (e.g. modem, router, cable, NIC, Wireless Access Points, hub, switch etc)? Candidates need to understand the purpose of the hardware. How to use Normalisation to produce a suitable database schema and explain the benefits of doing so. Be able to normalise up to 3NF. Candidates should know and be able to apply the criteria for 1NF, 2NF and 3NF. Discuss a range of methods for capturing data (such as forms, OCR, OMR and sensors) selecting data (such as Query By Example and SQL), managing data (such as changing data by manipulating it – e.g. arithmetic functions, adding, editing, deleting the data) and exchanging data (with commons formats such as CSV and JSON). Be able to discuss/justify suitable methods as part of a more open question. Candidates need to understand the need to interrogate data within a database. Candidates should understand the purpose of indexing in a database and the benefits of using indexing to optimise the searching for data. Candidates need to have experience of a range of methods for capturing data (such as forms – what do they collect, what do they look like - data mining, where does the data come from, how is it collected and analysed), selecting data (such as how to produce QBEs – adding fields, tables, criteria, sorting - selecting through Boolean expressions – AND, OR, NOT), managing data (such as changing data by manipulating it – e.g. arithmetic functions - , adding, editing, deleting the data) and exchanging data (such as methods of transferring data – electronic i.e. memory stick, e-mail, and non-electronic e.g. paper based - appropriate formats for the transfer of data and communication mediums to transfer data – such as the structure, is it in a table or a list).(such as forms, data mining), selecting data (such as producing QBEs, selecting through Boolean expressions), managing data (such as manipulation, adding, editing, deleting) and exchanging data (such as methods of transferring data, appropriate formats for the transfer of data and communication mediums to transfer data). Candidates should understand what is meant by transaction processing, and scenarios where transaction processing takes place. Candidates should understand the problems that arise from transaction processing, and how these can be overcome. Candidates should understand the ACID rules for transaction processing, and why databases should be built to these standards. Candidates should	through an object-oriented approach. Candidates need to be able to read, trace and write code to implement features of these data structures. Understand the need for compression, especially when transferring data via the Internet. Candidates need to understand the difference between lossy and lossless compression, and the benefits and drawbacks of each type. Candidates need to be able to recommend a type of compression for a given scenario. Candidates need to understand how run-length encoding can reduce the size of a file for example with a text file or image. Candidates should understand how dictionary coding works by substituting entries with a unique code. Candidates should have practical experience of using these algorithms with small example files. Candidates should understand the need for encryption. Candidates should understand how symmetric and asymmetric encryption work to encrypt and decrypt data. Candidates should understand the need for and purpose of using hashing algorithms to store data. Candidates should be aware of different uses for hashing, such as the storing of passwords. Understand the purpose of HTML, CSS and JavaScript. Candidates should have experience of writing webpages using HTML, CSS and JavaScript. Be able to read, write, amend and interpret code using HTML, CSS and JavaScript. Candidates should understand how and why search engine results are indexed. They should understand how PageRank ranks these results. Candidates should understand how page rank works at a high level. Candidates need to understand the difference between server and client-side processing, and should be aware of examples (for example JavaScript code vs PHP code) of processing on both sides. Candidates should be aware of the benefits and drawbacks of both types of processing. What are the De Morgan's law, and be able to apply these to a Boolean statement. Candidates should have experience of manipulating and simplifying Boolean statements using these rules of distribution, commutation, association and double negation. Candidates need to understand the purpose and principles of D type flip flops and how and where they are used in a computer. They should be able to recognise how they can be triggered by a clock pulse (see practice paper 2 for an example). Candidates are not expected to memorise the logic gates that make up a D-type flip flop. Candidates need to understand the purpose and function of an adder circuit, and the difference between a half and full adder. They should be able to recognise and draw the logic gates and truth tables for full and half adders. Understand that there are a range of possible solutions to a task, and that these algorithms may be different in respect to their execution time and the amount of memory they make use of. Candidates need to be able to compare different algorithms for a given data set and demonstrate an understanding of which is more efficient in terms of speed and/or memory. Candidates need to be able to compare the use of one or more algorithms against several

	understand how record locking prevents the overriding of data, and understand how record locking takes place. Understand the need for normalised floating-point numbers. Candidates should be able to normalise a floating-point number. Be able to perform addition and subtraction floating point arithmetic including addition and subtraction of both positive and negative numbers. Be able to perform right and left logical shifts. Candidates should understand the effect of right and left shifts on binary numbers. Understand the purpose of using masks with bitwise operators, and should have experience of applying masks using AND, OR and XOR. Understand that there are a variety of types of programming paradigms such as procedural, OOP, low-level, and that each has its strengths and weaknesses in specific scenarios, topics or areas. Candidates need to have knowledge and experience of using a procedural programming language for example Python, VB.NET etc. Candidates need to be experienced in using procedural programming features such as (but not limited to) variables, constants, selection, iteration, sequence, subroutines, string handling, file handling, Boolean and arithmetic operators. Candidates need to be able to read, trace, amend and write procedural program code. What is the purpose and need for assembly language. They should be able to read, write, trace and amend programs written in the Little Man Computer language. Candidates need an understanding of addressing, which should be integrated with assembly language. Candidates should have experience of using immediate, direct, indirect and indexed addressing in the writing, reading and tracing of programs written in assembly language. Candidates need to understand object-oriented code (as specified in the pseudocode guide). They need to understand classes, objects, attributes and methods. They need to understand the difference between private and public attributes and methods. Candidates need to understand encapsulation and the use of get and set methods to access private attributes. Candidates need to understand the purpose and principles of inheritance. Candidates need to understand polymorphism and how it can be used within a program. Candidates need to be able to read, trace, amend and write code that makes use of these object-oriented techniques. Understand how the efficiency of an algorithm is measured using Big O notation. Understand the meaning of constant, linear, polynomial, exponential and logarithmic complexity. They need to be able to recognise and draw each of these complexities of using a graph and be able to read and write the notation.	different data sets, to determine how they will differ in their use of memory and speed of execution. How Dijkstra's shortest path algorithm works. Candidates need to be able to calculate the shortest path in a graph or tree using Dijkstra's shortest path algorithm. Candidates need to be able to read and trace code that performs Dijkstra's shortest path algorithm. How the A* algorithm works. Candidates need to be able to calculate the shortest path in a graph or tree using the A* algorithm. Candidates need to be able to read and trace code that performs the A* algorithm. Understand what is meant by thinking concurrently. They need to be able to work out which parts of a program can be developed to take place (be processed) at the same time, and which parts are dependent on other parts. Candidates need to understand the benefits and trade-offs that are brought from concurrent processing, and be able to apply these to a given scenario. How concurrent processing could be applied to a specific program, why it would be applied to that program, and what problems might arise from using it. Understand that there are a number of stages involved in compilation. How lexical analysis works and how the code is converted into tokens with the removal of unnecessary elements (e.g. comments and whitespace). How syntax errors are identified and reported at the end of the syntax analysis. Candidates need to understand how the abstract syntax tree will be fed into the next stage of code generation, and that the object code is then created. Candidates need to understand why optimisation is important and how the results of lexical analysis feeds into syntax analysis, and how the tokens are checked to ensure they meet during (and after) code generation. What code libraries are, how they are used and the benefits and drawbacks from using libraries. Candidates should have experience of using libraries to write programs. Candidates should understand how libraries are used during compilation, and how linkers and loaders are used to combine the code and library code into the final executable file. Be able to determine if a problem can be solved using computational methods, such as decomposition, abstraction, calculations, storage of data. Be able to recognise a problem from a description of a scenario, decompose the problem and use abstraction to design a solution. Understand how divide and conquer can be used within a task to split the task down into smaller tasks that are then tackled. Candidates also need to identify how tasks can be carried out simultaneously to produce a solution. What is the purpose of backtracking within an algorithm, for example when traversing a tree? Candidates need to be able to read, trace and write code that makes use of backtracking for a given scenario.

	Candidates need to know the best- and worst-case complexities for the searching and sorting methods. Candidates need to understand the difference between best case, average case and worst-case complexities and how and why these can differ for an algorithm. Understand a tree structure, both binary and multi-branch trees. Candidates need to be able to add and remove data to/from a tree. Candidates need to be able to read, trace and write code to implement a tree structure (including adding and removing items). Candidates need to understand how a tree can be implemented using a different data structure, such as a linked list. Why and how trees are traversed. Candidates need to understand how a depth-first (post-order) traversal works, and be able to perform the traversal on a tree. Candidates need to be able to read, trace and write code for a post-order traversal. Candidates need to understand how a breadth-first traversal works, and be able to perform the search on a tree. Candidates need to be able to read, trace and write code for a breadth-first traversal on a tree. Why is a linked list a dynamic data structure? Candidates need to be able to add, remove and search for data to/from/in a linked list. Candidates need to be able to read, trace and write code to implement a linked list (including adding, removing and search for items). Understand the need for searching and sorting algorithms. Candidates need to understand pre-conditions required to perform a specific algorithm. How a merge sort works and be able to perform a merge sort on a set of data. Candidates need to be able to read, trace and write code to perform a merge sort. How a quick sort works and be able to perform a quick sort on a set of data. Candidates need to be able to read, trace and write code to perform a quick sort.	What is meant by data mining, and how data mining is used in a situation. Candidates need to understand the complexities within data mining and how a program will search for and interrogate the data. What is meant by heuristics, and how they can be used within a program (for example the A* algorithm). Candidates should have some experience of programming a simple heuristic and be able to apply their knowledge to a given scenario to explain the purpose and benefits of using heuristics in a solution. What are the principles, and purpose of performance modelling, and how it is used in the production of software? What is pipelining and how it is used within programming (for example the result from a process feeds into the next process). How visualisation can be used to create a mental model of what a program will do or work, and that from this they can plan what is going to happen or what they will need to do.
Assessment topics	September: Year 13 baseline test / Y12 PPE re-sit (as appropriate) Progress test(s) at various points	Progress test(s) at various points Pre-PPE tests Y13 PPEs (Feb) Progress tests
Cross curricular links/Character Education	Maths, Electronics, English language Problem solving, Resilience Mirroring the practices in the real world of software development	