Unit 2 | 2.2.2 | Computational methods

Specification & learning objectives

A Level	Specification point description
2.2.2a	Features that make a problem solvable by computational methods
2.2.2b	Problem recognition
2.2.2c	Problem decomposition
2.2.2d	Use of divide and conquer
2.2.2e	Use of abstraction
2.2.2f	Learners should apply their knowledge of: • backtracking • data mining • heuristics • performance modelling • pipelining • visualisation ...to solve problems

Resources

PG Online textbook page ref: 277-286

Hodder textbook page ref: 21-26

CraignDave videos for SLR 24

Key question: What features make a problem solvable by computational methods?

An algorithmic problem with a finite set of inputs will always be solvable. This does not necessarily mean computable, but a solution that can be mapped from a set of inputs 1, 2… *n* which can be output to yes/no can be solved by algorithm. The simplest method of understanding this is to map the inputs via a table to the appropriate answer.

A problem that has an infinite set of valid inputs causes more problems, as some will be solvable whereas others may not.

One classification of algorithmic problems can be determined by its time complexity. Any problem that can be solved with *an* polynomial time complexity or less, i.e. $O(n^a)$ or less, is known as *tractable*.

Key question: What features make a problem solvable by computational methods?

Computable and non-computable problems

Correct solutions can always be found for a solvable problem using an algorithm. Just because they are solvable, however, does not mean that they are computable in a reasonable amount of time, but they will be solvable in less than infinite time. Unsolvable problems are those that cannot be solved by an algorithm that will produce the right answer all the time, or problems that might take an infinite amount of time to solve.

Key question: What is divide and conquer?

This is a programing paradigm or common algorithmic technique in which a problem is broken down recursively until it is solved using a simple technique (conquer).

Key question: What is backtracking?

Backtracking is a method of finding solutions by trying a method and then going back if it fails to try another method. This sounds complicated but is a good recursive technique to finding a solution to a problem.

Key question: What is data mining and how can it be used to discover new trends?

Data mining is using pattern recognition or summarising of data within large data sets to find patterns or trends within the data. This has two useful features:

Spotting erroneous data

For example, a data-mining algorithm could search through a set of customers in an orders table to check on addresses in a particular area. A misspelt town name would be spotted as not part of the normal lists. If in the data there were five common towns in the data set but one unique value found it would almost certainly be an error or worth investigation.

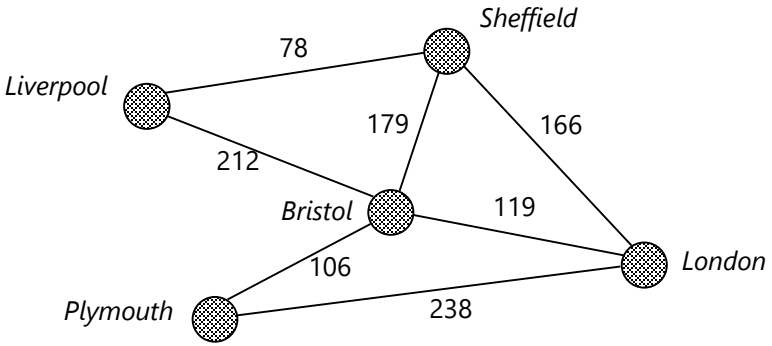
Spotting trends in data

This is especially useful in the retail industry as spotting a trend can significantly improve sales and profitability. For example, by data-mining credit card sales and days it is possible to identify which customers are coming into the store on particular days and what was being bought. It may be found that most people buy milk when then enter a supermarket, so placing it towards the back of the store means that a customer has to walk by other products to get to the milk and is more likely to buy other products.

Key question: What are heuristics?

In computer science there can be a compromise between the best solution and a quick solution. Often speed is an important factor so a 'good enough' solution or method instead of finding the perfect solution is acceptable.

An example of this is the travelling salesman problem (to the right) in which the shortest path is a problem that increases with complexity; however, using the nearest path heuristic method produces a solution which is acceptable.



This method is often utilised within satellite navigation systems in which a reasonably fast route is acceptable for quick calculations.

Key question: What is performance modelling?

This method utilises simulation to predict the performance of a method used. Simulation models are solutions that, instead of trying to solve a problem, reflect what happens in a situation. The advantage of performance modelling is that it allows a person to try 'what if'-type questions and use tools to optimise the best solution.

Key question: What is pipelining in the context of programming?
(Not CPU pipelining)

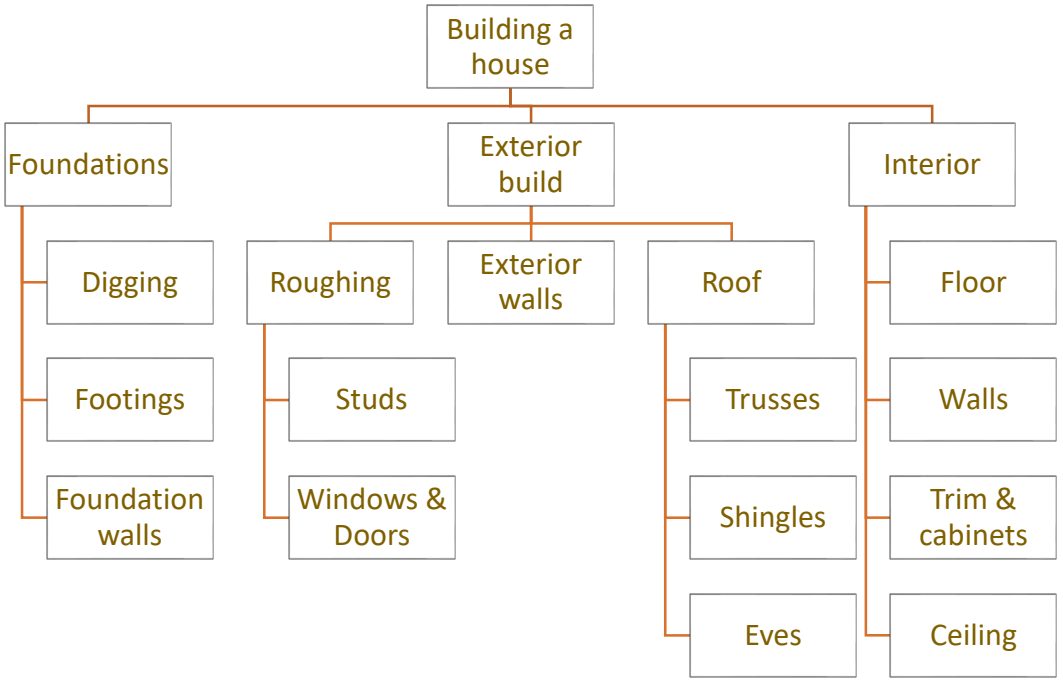
Just as the technique of *pipelining* is used in increasing the speed of a processor *(see section 1.1.1)*, pipelining in development is where the output of a stage is fed as the input into the next stage. This method of thinking is using the production-line-type thinking and is a good method for programming parallel processes.

Key question: How can visualisation be used to help solve a problem?

Visualisation is the method of finding a solution to a problem by using visual or diagrammatic methods. For example, a tree structure is not actually stored as a tree within the computer but is best solved by drawing the tree structure to solve a problem. Developers often use diagrams to help organise data structures within a computer.

Typical exam questions

Fig.1 illustrates the steps to build a house.



1. In the context of Fig.1, explain the concepts of problem decomposition and divide and conquer. **[4]**

Typical exam questions

2. Explain why the architect of the house would use performance modelling before building starts. **[2]**

3. In the context of building a house, explain the concept of 'levels of abstraction', and the advantage this gives the housing developer. **[3]**

Unit 2 | 2.2.2 | Computational methods

Target: Overall grade:

Minimum expectations & learning outcomes

☐	Terms 225-233 from your A Level Key Terminology should be included and formatted.
☐	You must demonstrate a clear understanding of the various computational methods listed in the specification table. This could be done as a series of diagrams, flow charts, annotated code snippets or a combination of all these presentation techniques. There are many examples of these methods covered in other SLRs you may wish to use as examples. E.g. performance modelling of scheduling algorithms. Abstraction and problem decomposition have already been covered in SLR18 & 20.
☐	Answer the exam questions.

Feedback

Breadth	Depth	Presentation	Understanding
☐ All	☐ Analysed	☐ Excellent	☐ Excellent
☐ Most	☐ Explained	☐ Good	☐ Good
☐ Some	☐ Described	☐ Fair	☐ Fair
☐ Few	☐ Identified	☐ Poor	☐ Poor

Comment & action required

Unit 2 | 2.2.2 | Computational methods

Reflection & Revision checklist

Confidence	Clarification
☹ 😐 ☺	Candidates need to be able to determine if a problem can be solved using computational methods, such as decomposition, abstraction, calculations, storage of data.
☹ 😐 ☺	Candidates need to be able to recognise a problem from a description of a scenario, decompose the problem and use abstraction to design a solution.
☹ 😐 ☺	Candidates need to 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.
☹ 😐 ☺	Candidates need to understand 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 understand 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.
☹ 😐 ☺	Candidates need to understand 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.
☹ 😐 ☺	Candidates need to understand the principles, and purpose of performance modelling, and how it is used in the production of software.
☹ 😐 ☺	Candidates need to understand the principle of pipelining and how it is used within programming (for example the result from a process feeds into the next process).
☹ 😐 ☺	Candidates need to understand 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 ahead what is going to happen or what they will need to do.