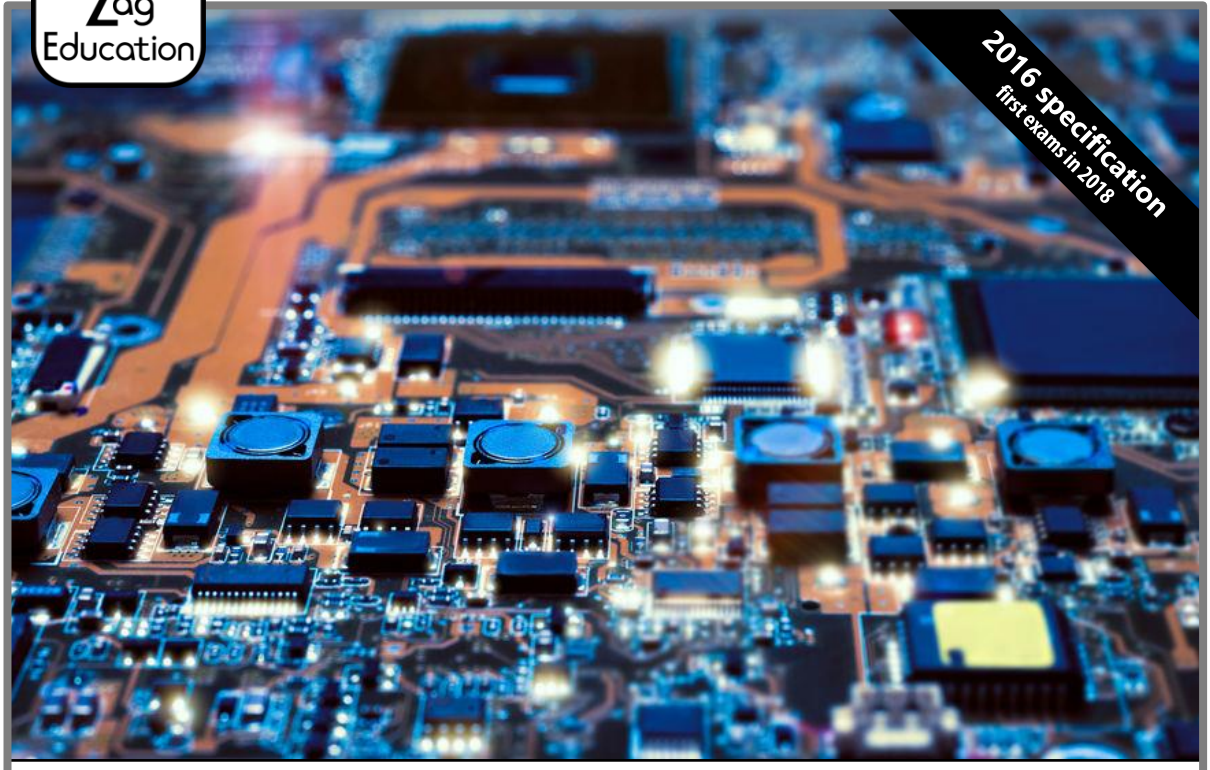
Zig Zag Education

Computer Science GCSE (9–1) | AQA | 8520



Revision Guide

for GCSE AQA (9–1) Computer Science

zigzageducation.co.uk

POD 7108

Contents

Thank you for choosing ZigZag Education!

Talk to Us!

Love it as it is?
Let the author and other teachers know what you think

Got a suggestion?
If your improvement leads to an update we will send you a new copy for free

Found a problem?
We will fix it and send you a free updated copy

We ♥ your feedback!
Let us know what you think using the feedback sheet on the next page.
£10 ZigZag Voucher for detailed & complete reviews!





Web:
zzed.uk/more

Email:
computerscience@zigzageducation.co.uk

Real Person:
0117 950 3199

Fax:
0117 959 1695

Post:
ZigZag Education, Unit 3, Greenway Business Centre, Doncaster Road, Bristol BS10 5PY

Become a Published Author

ZigZag is a large community of over 6000 teachers & educationalists.
Review new titles, publish your own work or write to a brief.

Fancy being involved?

Then register at...

publishmenow.co.uk

The Professional Publishing Community

Alternatively email new resource ideas directly to...
publishmenow@zigzageducation.co.uk

For more resources go to **zzed.uk/more**
where you can preview every page before you buy

Teacher Feedback Opportunity

£10 ZigZag Voucher for detailed & complete reviews! ◆ Use for problems/areas for improvement/positive feedback

Resource ID & Name	7108 Revision Guide for GCSE AQA (9–1) Computer Science

School Name	

Your Name		**Position**	

Overall, what did you think about this resource?.....

.....

.....

.....

I particularly like this resource because… .....

.....

.....

How does it help you or your students? .....

.....

It is better than some other resources because… .....

.....

.....

What might you say to a colleague in a neighbouring school to persuade them to use this resource?.....

.....

.....

.....

How well does it match your specification (and which specification is this)?.....

.....

Other comments, suggestions for improvements, errors found (please give page numbers) etc. .....

.....

.....

Resources I would like published:	

Resources I might write*, or have written,* for consideration for publication:	

Fax to:	Email to:	Submit online:	Post to:
0117 959 1695	**feedback@ zigzageducation.co.uk**	**zzed.uk/feedback**	**ZigZag Education, Unit 3, Greenway Business Centre, Doncaster Road, Bristol BS10 5PY**

INTERNAL USE ONLY
Feedback logged: ✓☐
Complete & detailed: Y /N
If detailed, £10 sent: ✓☐

Terms and Conditions of Use

Terms and Conditions

Please note that the **Terms and Conditions** of this resource include point 5.3, which states:

> **"You acknowledge that you rely on your own skill and judgement in determining the suitability of the Goods for any particular purpose."**

"We do not warrant: that any of the Goods are suitable for any particular purpose (e.g. any particular qualification), or the results that may be obtained from the use of any publication, or expected exam grades, or that we are affiliated with any educational institution, or that any publication is authorised by, associated with, sponsored by or endorsed by any educational institution."

Copyright Information

Every effort is made to ensure that the information provided in this publication is accurate and up to date but no legal responsibility is accepted for any errors, omissions or misleading statements. It is ZigZag Education's policy to obtain permission for any copyright material in their publications. The publishers will be glad to make suitable arrangements with any copyright holders whom it has not been possible to contact.

Students and teachers may not use any material or content contained herein and incorporate it into a body of work without referencing/acknowledging the source of the material ("Plagiarism").

Disclaimers

This publication is designed to supplement teaching only. Practice questions may be designed to follow the content of a specification and may also attempt to prepare students for the type of questions they will meet in the examination, but will not attempt to predict future examination questions. ZigZag Education do not make any warranty as to the results that may be obtained from the use of this publication, or as to the accuracy, reliability or content of the publication.

Where the teacher uses any of the material from this resource to support examinations or similar then the teacher must ensure that they are happy with the level of information and support provided pertaining to their personal point of view and to the constraints of the specification and to others involved in the delivery of the course. It is considered essential that the teacher adapt, extend and/or censor any parts of the contained material to suit their needs, the needs of the specification and the needs of the individual or group concerned. As such, the teacher must determine which parts of the material, if any, to provide to the students and which parts to use as background information for themselves. Likewise, the teacher must determine what additional material is required to cover all points on the specification and to cover each specification point to the correct depth.

ZigZag Education is not affiliated with Pearson, Edexcel, OCR, AQA, WJEC, Eduqas, EA, International Baccalaureate Organization or DFE in any way nor is this publication authorised by, associated with, sponsored by or endorsed by these institutions unless explicitly stated on the front cover of this publication.

Teacher's Introduction

This guide has been produced to supplement the teaching of AQA (9–1) Computer Science specification – for first teaching from September 2016 (first exams 2018).

There is a chapter for each of the 7 topics of the specification. It can be used purely as a revision guide or as individual topic summaries, although it is not intended to completely replace taught material.

Each chapter contains, along with the theory material itself, a series of exam-style questions with model answers and commentaries. Throughout the guide, a full range of question types is covered, from single-word answers and definitions to long-answer descriptions and discussions.

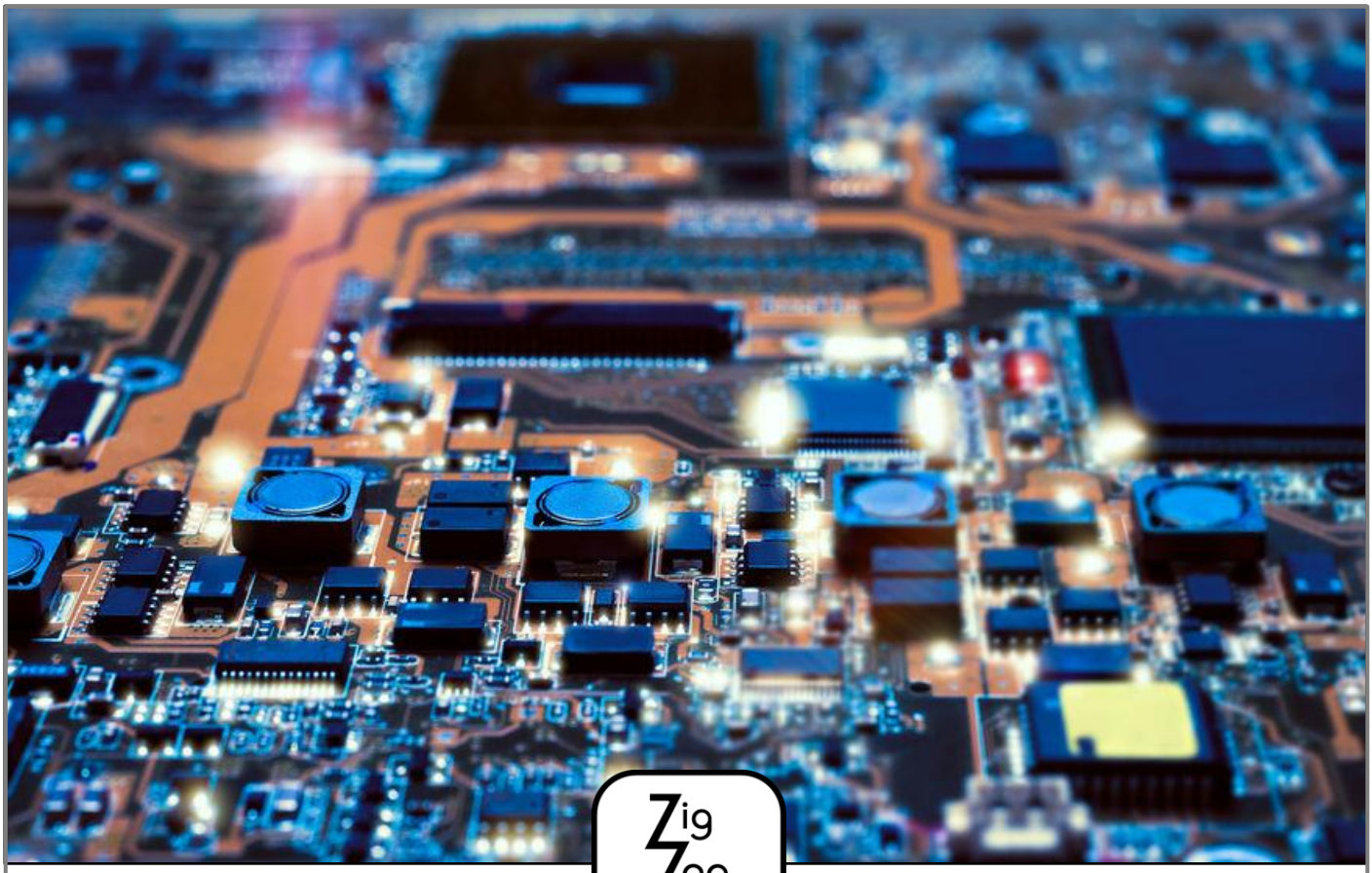
One chapter can be distributed to students each week and can be used to supplement taught material by aiding such homework/classwork tasks as providing written summaries of a chapter and/or completing the end-of-chapter questions.

More imaginative supplementary tasks that can use this guide as a starting point include the following (you may want to build some or all of these into a weekly routine, each week focusing on a different chapter):

- Providing students with lines from the appropriate section of the specification and asking them to treat each line as if it were a question. The guide can aid them in locating the answer.
- Asking students to produce five multiple-choice questions based on each chapter. Each question they produce needs to contain a correct answer, three realistic wrong answers and an indication of which answer they believe is correct. The better sets of questions can be archived to produce a half-term multiple-choice quiz, generated by students.
- Asking students to produce a mind map of each chapter as a means of aiding revision.
- Dividing students into groups to deliver presentations on different areas of a chapter. If the group is fairly mature, these presentations can be peer-assessed.

The revision notes are provided in A5 booklet form as well. Please note that this booklet does not contain the mind maps or answers.

R Lee, September 2017



Zig Zag Education

GCSE AQA (9–1) Computer Science

Revision Guide

Revision Checklist

1. Fundamentals of Algorithms	☐ Define the term 'algorithm' ☐ Explain what is meant by 'decomposition' ☐ Explain what is meant by 'abstraction' ☐ Interpret pseudocode ☐ Write pseudocode ☐ Identify inputs, processes and outputs within a given algorithm ☐ Describe the purpose of a given algorithm ☐ Define the term 'efficiency' regarding algorithms ☐ Compare different algorithms' efficiencies ☐ Describe the linear search ☐ Describe the binary search ☐ Compare linear and binary search algorithms ☐ Describe the bubble sort ☐ Describe the merge sort ☐ Compare bubble and merge sorting algorithms
2. Programming	☐ Describe what is meant by 'data type' ☐ Define 'integer', 'real', 'Boolean', 'character' and 'string' data types ☐ Declare and assign constants and variables ☐ Describe 'iteration' ☐ Describe 'selection' ☐ Define the term 'subroutine' ☐ Use and explain definite (count-controlled) iteration ☐ Use and explain indefinite (condition-controlled) iteration ☐ Use and explain nested structures ☐ Explain the need for meaningful identifiers ☐ Write code that accepts keyboard input ☐ Write code to display output on a screen ☐ Write code to read from and write to a text file ☐ Write code to perform string operations 'length', 'position', 'substring' and 'concatenation' ☐ Write code to cast one data type as another ☐ Describe the advantages of using subroutines ☐ Describe the purpose of parameters ☐ Write code that uses subroutines ☐ Write code that performs arithmetic operations (addition, subtraction, division, multiplication) ☐ Write code that performs relational operations (combinations of less than, equal to, not equal to and greater than) ☐ Write code that performs logical operations (AND, OR, NOT) ☐ Define the term 'data structure' ☐ Explain and use one-dimensional and two-dimensional arrays ☐ Explain and use records ☐ Distinguish between global and local variables ☐ Describe and write validation routines ☐ Describe and write authentication routines ☐ Describe and select appropriate normal, boundary and erroneous test data ☐ Distinguish between low-level and high-level languages ☐ Distinguish between interpreters, compilers and assemblers for program translation

3. Fundamentals of Data Representation	☐ Define the term ‘number base’ ☐ Describe decimal, binary and hexadecimal number bases ☐ Explain the purpose of hexadecimal ☐ Use binary to represent whole numbers between 0 and 255 ☐ Use hexadecimal to represent whole numbers ☐ Perform conversions between binary and decimal ☐ Perform conversions between binary and hexadecimal ☐ Perform conversions between decimal and hexadecimal ☐ Define the terms ‘bit’, ‘byte’, ‘kilobyte’, ‘megabyte’, ‘gigabyte’ and ‘terabyte’ ☐ Perform binary addition ☐ Explain and apply a binary shift to a number ☐ Define the term ‘character set’ ☐ Describe the ASCII and Unicode character sets ☐ Define the term ‘pixel’ ☐ Describe how size and colour depth affects image file size ☐ Calculate bitmap image file sizes given the number of pixels and the colour depth ☐ Convert between binary data and monochrome images ☐ Explain what is meant by the term ‘analogue’ ☐ Describe sound sampling ☐ Describe sampling rate and sample resolution ☐ Calculate sound file size given the sampling rate and sample resolution ☐ Define the term ‘data compression’ ☐ Explain the need for data compression ☐ Create and use Huffman trees ☐ Calculate the number of bits required to store data encrypted with Huffman encoding ☐ Explain the operation of run length encoding (RLE) ☐ Manually apply run length encoding to a given piece of data
4. Computer Systems	☐ Define the terms ‘hardware’ and ‘software’ ☐ Construct truth tables for AND, OR, NOT logic gates ☐ Create truth tables for logic circuits with up to three inputs ☐ Create, modify and interpret logic circuits ☐ Distinguish between system software and application software ☐ Describe the roles of the operating system ☐ Explain the Von Neumann architecture ☐ Describe the roles of the ALU, the control unit, the clock and the bus within a CPU ☐ Describe the factors that affect CPU performance ☐ Describe the fetch-execute cycle ☐ Distinguish between main memory and secondary storage ☐ Explain the need for secondary storage ☐ Describe the operation of optical, magnetic and solid state secondary storage ☐ Define the term ‘cloud storage’ ☐ Describe the advantages and disadvantages of using cloud storage ☐ Explain what is meant by the term ‘embedded system’

5. Fundamentals of Computer Networks	☐ Define the term 'computer network' ☐ Describe the pros and cons of using a network ☐ Distinguish between PANs, LANs and WANs ☐ Distinguish between wired and wireless networks ☐ Explain the operation of 'star' and 'bus' network topologies ☐ Define the term 'network protocol' ☐ Describe the following protocols: Ethernet, Wi-Fi, TCP, UDP, IP, HTTP, HTTPS, FTP, SMTP, IMAP ☐ Explain why network security is necessary ☐ Explain the role of each of the following methods of network security: authentication, encryption, firewall, MAC address filtering ☐ Describe the four-layer TCP/IP model ☐ Identify which protocols operate at which level of the TCP/IP model
6. Fundamentals of Cybersecurity	☐ Define the term 'cybersecurity' ☐ Describe social engineering techniques, including phishing, pharming, shouldering and pretexting ☐ Describe malware, including viruses, Trojans, spyware and adware ☐ Describe the threat of weak and default passwords ☐ Describe the threat of misconfigured access rights ☐ Describe the threat of removable media ☐ Describe the threat of unpatched software ☐ Explain the role of penetration testing ☐ Explain the role of biometric measures ☐ Explain the role of password systems ☐ Explain the role of CAPTCHA ☐ Explain the role of email confirmation ☐ Explain the role of automatic software updates
7. Ethical, Legal and Environmental Impacts	☐ Define the terms 'ethical', 'legal' and 'environmental' ☐ Explain the impact of technology on privacy ☐ Describe the conflict over privacy that occurs between government and the population ☐ Identify ethical, legal and environmental implications in a range of situations

1. Fundamentals of Algorithms

i **Algorithm** – a series of instructions that describes how to solve a specific problem or perform a specific task.

Many problems that we face in our day to day lives can be solved by following a series of steps. Something as simple as getting into a car can be broken down into the following:

- pressing 'unlock' on the key
- pulling the handle
- climbing in
- sitting on the chair
- closing the door by pulling the inside handle

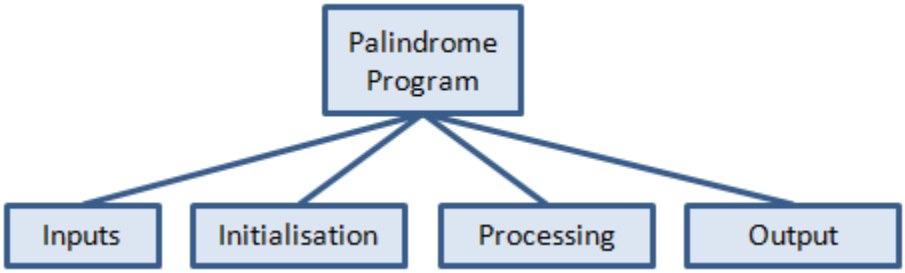
Each stage cannot be broken down further, and they tell us exactly how to get into the car. This is the idea of an algorithm - as series of explicit instructions that describe exactly what to do.

Representing Algorithms

i **Decomposition** – breaking a problem down into smaller 'sub-problems'. This has a number of advantages:

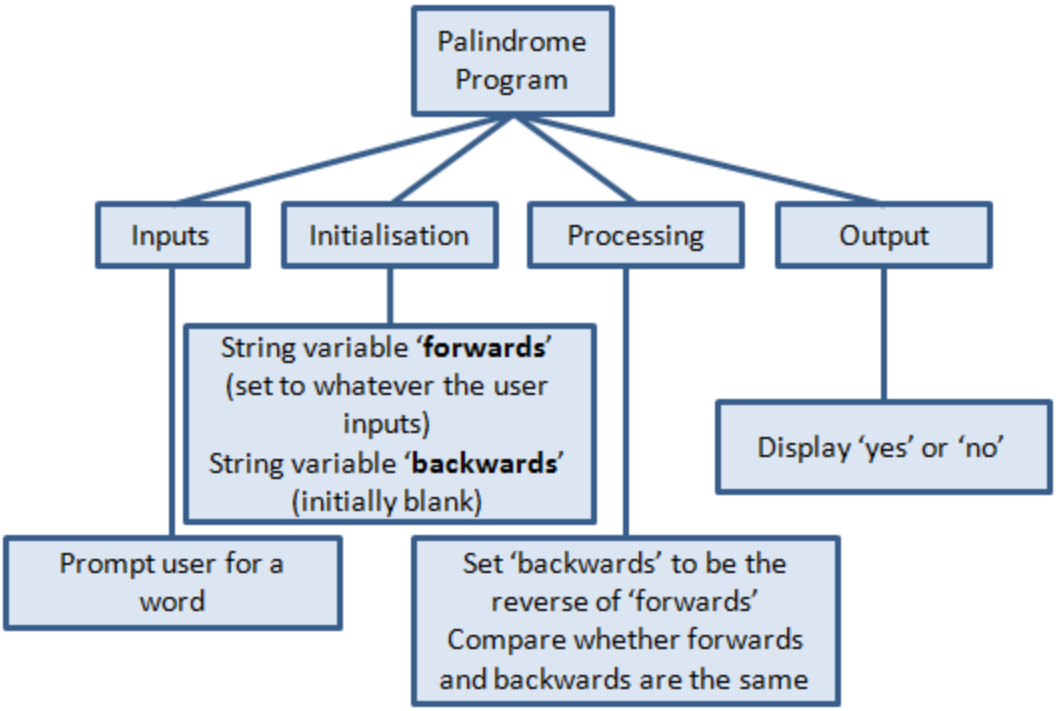
- Smaller problems are easier to solve than larger problems
- Each 'sub-problem' can be developed separately, making planning and working to a timescale easier
- 'Sub-problems' are easier to distribute among a team than one large problem.

Suppose you want to develop a program that identifies whether or not a word is a palindrome (the same spelled forwards as backwards). You might decompose this problem as follows:



Inputs	What data enters the system, and how?
Initialisation	What variables are needed and what are their initial values?
Processing	What processes (calculations, sorts, etc.) are carried out on the inputs?
Outputs	What data leaves the system and in what format?

You could now examine the **requirements** of each component. What exactly is needed at each stage of the program?



At this point, each component can be designed, possibly using a flow chart or pseudocode.

Sometimes, multiple levels of decomposition are required. Looking at the 'processing' stage in the above diagram, it might be that this particular sub-problem could be broken into smaller sub-problems.

i **Abstraction** – hiding the layers of complexity within a system, in order to focus on one specific layer of complexity.

You might use the 'min()' function in Python, which identifies the lowest value in a series of numbers, but you probably have no idea exactly *how* this function operates. This is not a problem – you do not need to know how it works, only what it does. The complexity of the min() function has been hidden from you.

Commonly used methods of defining algorithms include **pseudocode** and **flow charts**.

The exam might contain questions on any combination of these methods. One way to be ready for this is to practice converting between them. Try turning a flow chart into pseudocode, or vice-versa.

Pseudocode

i **Pseudocode** – a cross between English and a generic-looking programming language. While pseudocode would not compile, a competent programmer could convert it to code that *would* compile.

A pseudocode algorithm for selling tickets might be as follows. Larger purchases are charged at a lower rate per ticket:

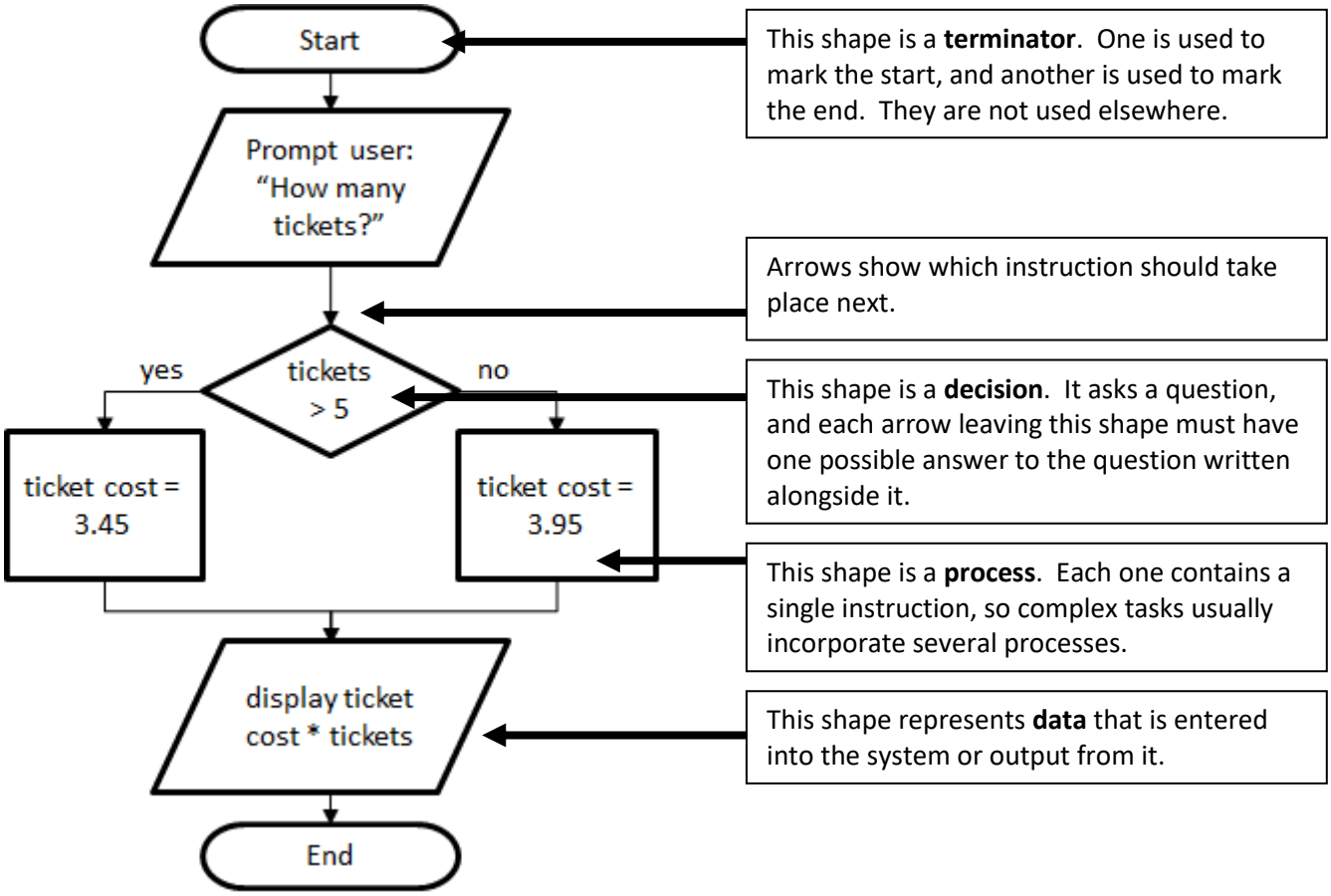
```
Prompt user: "How many tickets"
If tickets > 5
    Display: tickets * 3.45
Else
    Display: tickets * 3.95
```

Don't worry about writing pseudocode that matches a mark scheme specifically. As long as the meaning of your pseudocode is clear, and you use keywords such as IF correctly, you will be given credit for a correct answer.

Flow charts

Flow chart – a means of defining an algorithm using shapes and arrows. Within each shape, an action is taking place or a decision is being made. The direction of the arrows indicates which action or decision comes next.

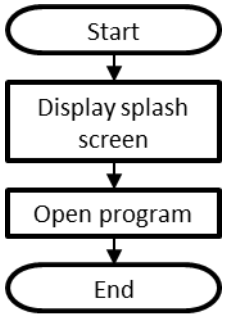
A flow chart does the same job as pseudocode in defining an algorithm, but it is more understandable to someone who is not a programmer. The flow chart segment below defines the same part of a program as the pseudocode above, but it looks very different:



Control Structures

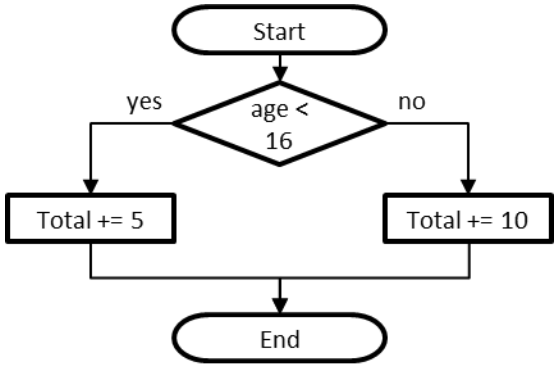
Sequence is when instructions in an algorithm each need to be executed only once, and in the order in which they are written

```
Display splash screen
Open program
```



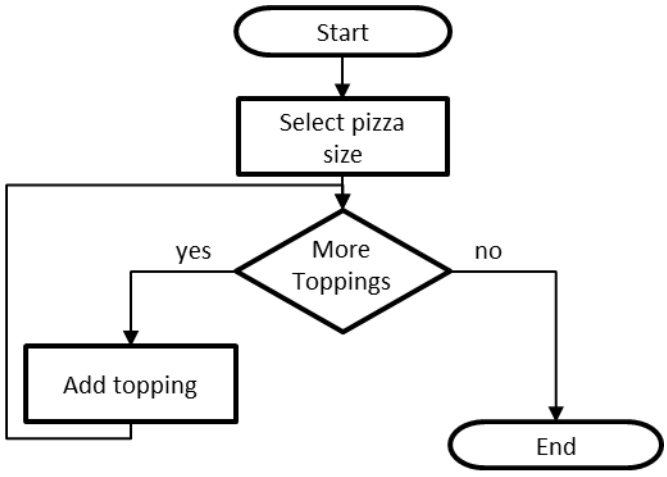
Selection is when one sequence of instructions or another, but not both, is executed. The sequence that is executed will depend on what happened in earlier instructions. This example either adds £10 or £5, depending on whether an adult or a child is paying to get into an attraction.

```
IF age < 16
      Add 5 to total
ELSE
      Add 10 to total
END IF
```



Iteration takes place when one or more instructions is executed more than once. Iteration is also known as repetition. Iteration is taking place when program code (or pseudocode) contains the words LOOP, FOR, WHILE or REPEAT. This program snippet might be for an online system for ordering pizza:

```
Select pizza type
WHILE more toppings required
      Select pizza topping
REPEAT
```



Interpreting Algorithms

You may be provided with an algorithm and asked what it does or what the output would be, so you need to be able to read and understand algorithms.

```
1     int number = 3
2     int result = 1
3     while number > 1
4           result = result * number
5           number = number – 1
6     end while
7     output result
```

Interpreting an algorithm works best when you do so one line at a time, as a computer would. A **trace table** can and should be used to keep track of variables that change throughout the algorithm. The variables in this program are called '`number`' and '`result`'.

number	result	Commentary
3	1	In lines '1' and '2', the variables are given these values.
3	1	In line 3, which is the start of a loop, we are told that the loop will run as long as 'number' is greater than '1'. Since 'number' is '3', we can continue.
3	3	In line 4, 'result' is set to itself multiplied by 'number'. '1' times '3' is '3'.
2	3	Line 5 says that 'number' should have one subtracted from it.
2	3	Line 6 marks the end of the loop, so we go back to the start.
2	3	'number' is still greater than '1', so the loop runs again.
2	6	In line 4, 'result' is set to itself multiplied by 'number'. '2' times '3' is '6'.
1	6	number is reduced by '1' again on line '5'.
1	6	Line 6 marks the end of the loop, so we go back to the start.
1	6	The loop will not run a third time, because 'number' is not greater than '1' any more ('1' is not greater than '1'), so we jump to the first line after the end of the loop.
1	6	'result' is displayed, which is currently '6'.

Looking at an algorithm as a whole can be daunting, but following it one line at a time makes it a good deal simpler; no individual line is particularly complicated, and errors can be easier to identify.

Efficiency of Algorithms

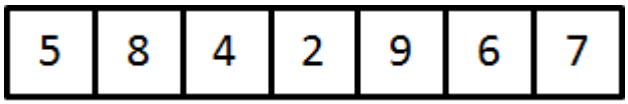
i **Efficiency** – a measure to compare two different algorithms that solve the same problem. A more efficient algorithm is a better choice. Efficiency can be measured in a number of different ways, but at GCSE level, you need only worry about **time** efficiency. An algorithm that can be executed in 20 instructions is more efficient than one that takes 30 instructions.

Searching Algorithms

i **Searching** – determining whether a specific piece of data exists within a data structure. If it does exist, a search algorithm will reveal its location.

Linear search

i **Linear search** – a search algorithm that begins at one end of a data structure, checking each data item in turn until the required item is found, or the end of the structure is reached.



If a linear search were being used to find the number 7, the numbers 5, 8, 4, 2, 9, 6 and 7 would be examined in that order. If the number 1 were being sought, it would not be found, but each element in the data structure would be examined to verify this.

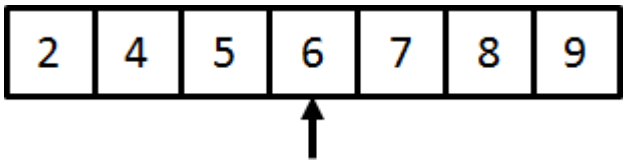
```
FOR A = 0 TO DATA.length
      IF DATA(A) = (item being searched for)
            return A
      END IF
END FOR
RETURN (not found)
```

If the item being searched for is found, the code returns the location within the data collection. The first location is numbered '0', the second '1', the third '2' and so on.

Binary Search

Binary search – a search algorithm that begins in the middle of a data structure, eliminating half of the remaining data with each pass. Binary searches are only appropriate when performed upon a sorted data structure.

A binary search for the number '7':



The value in the middle is 6; the value we are searching for is larger than that and, in a sorted array, would be found somewhere to the right of 6. Consequently, the 6, and everything to its left, is disregarded:

There are now three elements to be searched through. Again, the binary search begins in the middle, and again, elements can be disregarded because they can only contain numbers larger than seven:

The number we were searching for has been found.

Binary searches are quicker because, at each stage, half of the remaining data is disregarded. If an array of one million elements were to be searched using a binary search, it would take no more than 20 iterations to find a particular piece of data or discover that the data is not contained within the array. An array of one billion items would only require 30 iterations.

A binary search is more **efficient** than a linear search since it will, on average, find a search term more quickly than a linear search. However, binary searches do not work on unsorted data, so efficiency is not the only consideration.

Linear Search	Binary Search
+ Functions on unsorted data	+ Far more time efficient
- More time-consuming than a binary search	- Will not work on unsorted data

Sorting Algorithms

Sorting

i **Sorting** – putting data into order, whether that be numerical order, alphabetical order or chronological order, ascending (A-Z) or descending (Z-A).

Not sorted:

4	2	7	5	3

Sorted:

2	3	4	5	7

Bubble Sort

(4 2 7 5 3) ← Initial data (unsorted)

First Pass

(2 4 7 5 3) ← The 2 and the 4 have been switched, as 4 is greater than 2
(2 4 7 5 3) ← The 4 and the 7 are not switched, as they are already in the correct order
(2 4 5 7 3) ← The 5 and the 7 are switched, as 7 is greater than 5
(2 4 5 3 7) ← The 3 and the 7 are switched, as 7 is greater than 3

Second Pass

(2 4 5 3 7) ← The 2 and the 4 are not switched, as they are already in the correct order
(2 4 5 3 7) ← The 4 and the 5 are not switched, as they are already in the correct order
(2 4 3 5 7) ← The 3 and the 5 are switched, as 5 is greater than 3
(2 4 3 5 7) ← The 5 and the 7 are not switched, as they are already in the correct order

Third Pass

(2 4 3 5 7) ← The 2 and the 4 are not switched, as they are already in the correct order
(2 3 4 5 7) ← The 4 and the 3 are switched, as 4 is greater than 3
(2 3 4 5 7) ← The 4 and the 5 are not switched, as they are already in the correct order
(2 3 4 5 7) ← The 5 and the 7 are not switched, as they are already in the correct order

At this stage, the list is sorted, but one more pass would be performed, because a bubble sort only ends when a complete pass has yielded no changes or when n-1 (n being the number of elements to be sorted) passes have taken place.

Merge Sort

(4 9 5 1 3 2 7 8)	←	Unsorted data set
(4) (9) (5) (1) (3) (2) (7) (8)	←	Data is split into individual units
(4 9) (5) (1) (3) (2) (7) (8)	←	The first pairing, 4 and 9 are brought together and are already in order
(4 9) (1 5) (3) (2) (7) (8)	←	The next pairing is 5 and 1, whose positions are switched, i.e. sorted
(4 9) (1 5) (2 3) (7 8)	←	In this way, all data are merged into sorted pairings

Next, the pairs must be merged into groupings of four. We'll look at the first two pairings:

(4 9) (1 5)

The values '4' and '1' are compared. Since these are the first within their respective pairings, the smallest of the four must be one of these two:

(4 9) (~~1~~ 5) (1)

Next, the first *remaining* value in each pairing is compared with the other. Values '4' and '5' are compared here:

(~~4~~ 9) (~~1~~ 5) (1 4)

At any given point, two numbers are examined, with the next in order being added to a sorted copy of the data set. The copy is always in order:

(~~4 9~~) (~~1 5~~) (1 4 5 9)

The same principle is applied to the other two pairings to leave two sorted clusters of four data items:

(1 4 5 9) (2 3 7 8)		← Two sorted clusters of four data items each
(~~1~~ 4 5 9) (2 3 7 8)	(1)	← '1' and '2' were compared
(~~1~~ 4 5 9) (~~2~~ 3 7 8)	(1 2)	← '4' and '2' were compared
(~~1~~ 4 5 9) (~~2 3~~ 7 8)	(1 2 3)	← '4' and '3' were compared
(~~1 4~~ 5 9) (~~2 3~~ 7 8)	(1 2 3 4)	← '4' and '7' were compared

Eventually, this would result in a single, sorted data set, comprising 8 data items. This process could be repeated to sort a data set containing any number of data items.

Bubble Sort	Merge Sort
+ Straightforward to program + Does not use much extra memory space while the sort is in progress	+ More efficient in terms of time
- Can take a very long time with a large set of data to sort	- More complex to program - Requires lots of memory space as it runs

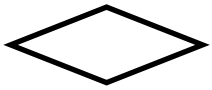
Sample Examination Questions

13. What is meant by the term **algorithm**? [1]

14. State the name of each of the following flowchart shapes. [3]

A

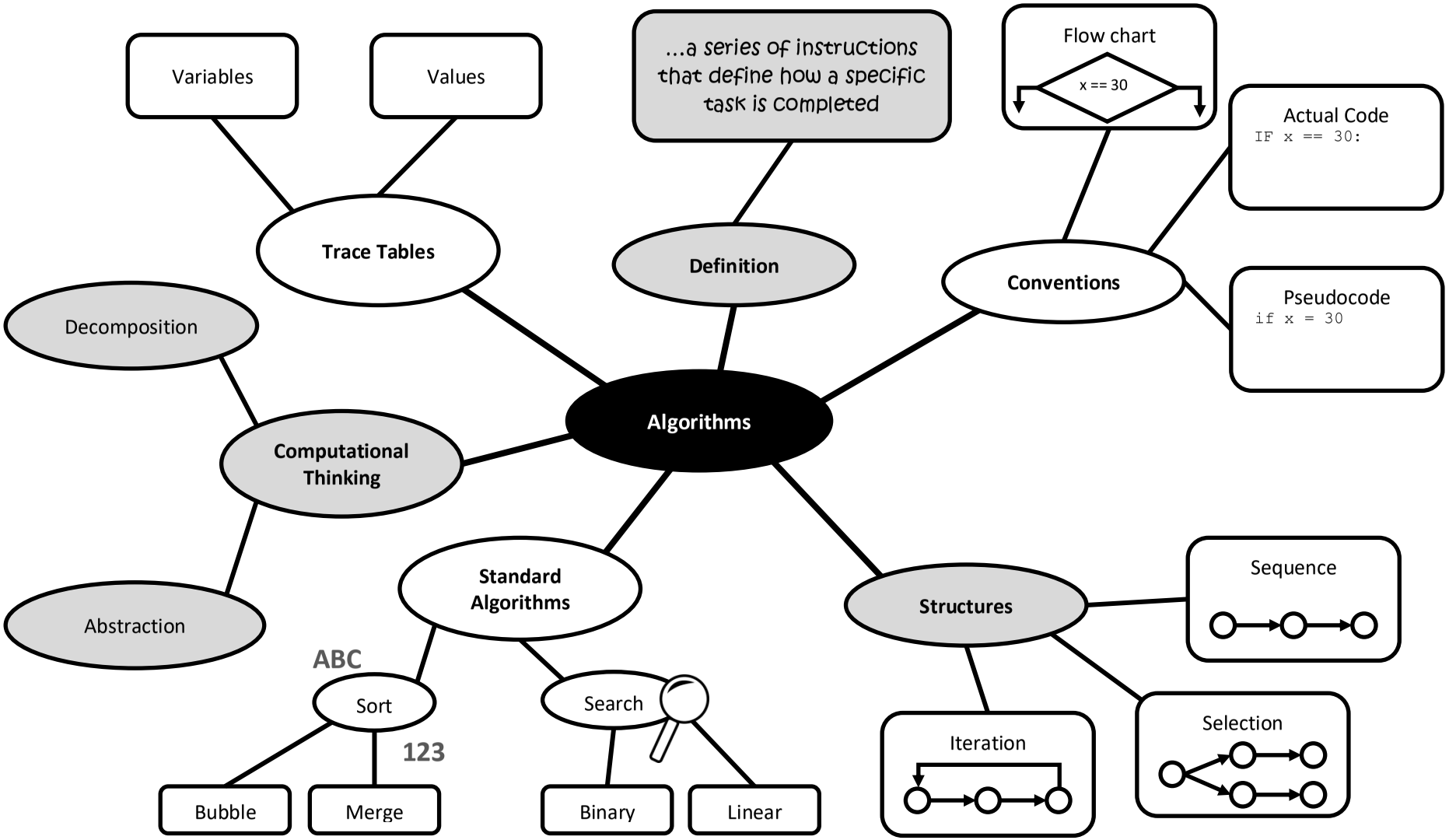
B



C

15. Describe the operation of a **bubble sort** algorithm. [3

'Fundamentals of Algorithms' Mind Map



2. Programming

Data Types

i **Variable** – a named space in memory, large enough to store a single piece of data. Each variable has a data type, and commonly-used data types are listed below.

i **Constant** – a named space in memory with a value that can never change while the program is running. Useful for pi (which will never change), or VAT (which seldom changes). The Python language does not allow for constants, but some other languages do.

Data Type	Description	Use in Python	Use in VB.Net
Boolean	Can be either true or false	`b = True`	`Dim bn as Boolean` `bn = true`
Character	A single letter, number, punctuation mark, space, etc.	Python does not have a 'character' data type, but most other languages do.	`Dim cr as Char` `cr = 'a'`
Integer	Whole numbers – positive, negative or zero	`i = 5`	`Dim i as Integer` `i = 5`

Real	Decimal numbers – positive or negative (zero can also be stored in a 'real' variable.	`pi = 3.142`	`Const pi as Double`
			`pi = 3.142`
			(constant declaration)

String	A collection of characters, used to store names, addresses, phone numbers, etc.	`name = "Bob"`	`Dim name as String` `name = "Bob"`

i **Declaration** – code that causes a variable to exist. Once a variable has been declared, it can be used. Trying to use a variable before it has been declared will cause an error.

i **Assignment** – the process of putting a value into a variable. Most languages use a single equals '=' to do this. In the instruction `i = 5`, the value '5' has been assigned to the variable 'i'.

Programming Concepts

Sequence

Sequence means that instructions will always execute in the order in which they were written. Every line will be executed once and only once:

```
hourlyRate = float(input("Hourly rate: "))
hours = int(input("Hours worked: "))
print(hourlyRate * hours)
```

This program asks for the hourly rate and the number of hours worked before displaying the total pay (those figures multiplied). At this point, the program ends.

Selection

Selection is when one path through the code is chosen where multiple possible paths exist. The presence of the word 'if' is the most common indicator of selection, although there are others.

```
hours = int(input("Number of hours: "))
rate = float(input("Rate per hour: "))
if hours > 40:
    print("Normal Earnings:     " + str(40 * rate))
    print("Overtime Earnings:   " + str((hours - 40) * rate * 1.5))
else:
    print("Normal Earnings:     " + str(hours * rate))
    print("No Overtime")
```

The first two lines will always run; the program asks the user for the number of hours and the rate per hour, storing each value in a separate variable. Then a decision is made. If 'hours' is greater than 40, the first two indented lines will run. Otherwise, the last two indented lines will run. There would never be a circumstance when all four would be executed.

Indefinite Iteration (pre-check)

Iteration means 'looping'. Code that is iterative might be executed multiple times, even though it is only written once.

```
looping = True
while looping == True:
    hourlyRate = float(input("Hourly rate: "))
    hours = int(input("Hours worked: "))
    print(hourlyRate * hours)
    userInput = input("Another? (y/n)")
    if userInput == "n":
        looping = False
```

The second line specifies the condition that will make the rest of the code loop. The code will keep repeating until the 'looping' variable is set to false. If 'looping' is set to false to begin with, the code will not even run once.

i **Nesting** – this involves placing one programming structure inside another one. The code above contains an IF structure (on the penultimate line) that is **nested** within the WHILE structure. There is no limit to the number of levels of nesting; that IF structure could have contained another WHILE structure, which itself contained another WHILE structure…

Indefinite Iteration (post-check)

Another approach is to check whether to run the loop *after* the rest of the code. This would perform exactly the same task as the previous block of code. The only difference is, with the 'while' keyword at the end instead of at the start, the code will always execute at least once. Many programming languages support post-check iteration, but Python is not one of them.

Definite Iteration

The above code loops until the user enters the letter 'n'. It is impossible to know in advance how many times the loop will run. Sometimes, code is required that runs a predetermined number of times:

```
for x in range(1, 13):
    print(x * 2)
```

This code will run twelve times, with the variable 'x' taking the values 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11 and 12 in order. The purpose of this code is to display the two times table.

Operations

i **Operation** – in the context of computer science, an operation is an action that is performed on one or more pieces of data in order to produce additional data. There are **arithmetic tions**, **relational operations** and **Boolean operations**.

i **Arithmetic operation** – a process performed on one or more numbers. Examples of arithmetic operators are: + - * /

Python Code	Explanation
`total = 5 + 10`	Addition **(15)**
`result = 10 - 5`	Subtraction **(5)**
`product = 5 * 10`	Multiplication **(50)**
`answer = 10 / 5`	Division **(2)**
`outcome = 13 DIV 5`	Integer division **(2)** – the remainder is ignored
`solution = 13 MOD 5`	Modulus **(3)** – divides but uses only the remainder

i **Relational operation** – a comparison between two values to check, for example, whether they are equal, or whether one is less than or greater than the other. Relational operations are found in IF statements and as part of loops.

Symbol Used in Exam	Python Code	Explanation
>	`if (x > y):`	If 'x' is greater than 'y'…
<	`while (a < b):`	While 'a' is less than 'b'…
≥	`if (q >= r):`	If 'q' is either greater than, or equal to, 'r'…
≤	`while (j <= k):`	While 'j' is either less than, or equal to, 'k'…
=	`if (e == f):`	If 'e' and 'f' are equal…
≠	`while (m != n):`	While 'm' and 'n' are *not* equal…

i **Logic operation** – A logic operation can have one of only two outcomes – true or false. A logic **operator** connects together logic **expressions** to produce more complex logic expression. AND, NOT and OR are the most commonly used logic operators.

Python Code	Explanation
`if (age > 15 and age < 65):`	Age has to be both above 15 *and* below 65, otherwise the contents of this IF structure would not run.
`if (age < 16 or age > 64):`	The contents of this IF structure would run if age was either below 16 *or* above 64.
`if (not(age < 16)):`	Translates to 'if age is *not* less than 16'. The 'not' operator inverts the outcome of a logic expression, changing it either from 'true' to 'false' or from 'false' to 'true'.

Here is a piece of Python code that combines all three types of operator. It calculates the cost of a group of adults and children entering a zoo. How much would it cost for two adults and one child?

```
adults = int(input("How many adults: "))
children = int(input("How many children: "))

if ((adults + children >= 5) or (adults >= 2)):
    cost = (adults * 9) + (children * 4.5)
else:
    cost = (adults * 10) + (children * 5)
print(cost)
```

Data Structures

i **Data structure** – a structured (organised) means of storing related data. While a variable can only store a single piece of data, a data structure can contain many.

Many data structures exist, and it's even possible to invent your own. In GCSE Computer Science, you need to be familiar with **one-dimensional arrays**, **two-dimensional arrays** and **records**.

i **One-dimensional array** – a data structure for storing multiple data items *of the same data type*. Think of a one-dimensional array as a row of variables. Instead of each one having a name, the whole array has a single name, while each **element** in the array has an **index number**, indicated below. A one-dimensional array might store a pupil's most recent spelling test scores.

0	1	2	3	4	5	6	7	8	9	10

The first element is always numbered '0'.

Python Code	Explanation
`myArray = [4,6,1,2,9,0]`	Creates an array called 'myArray' and populates it with six integers.
`myArray[0] = 5`	Places the number '5' into the first element (index '0') of the array.
`print(myArray[2])`	Displays the third element of the array – in this case, '1'.

i **Two-dimensional array** – a data structure for storing multiple data items *of the same data type*. Very much like a one-dimensional array, but can be considered as a grid rather than a row:

0, 0	0, 1	0, 2	0, 3
1, 0	1, 1	1, 2	1, 3
2, 0	2, 1	2, 2	2, 3

Elements are referred to, as seen here, with two numbers, much like 'x' and 'y' coordinates. If this grid were larger, it could be used as a grid for a computerised game of battleships or noughts and crosses.

Python Code	Explanation
`myArray2 = [["a","b"],["c","d"]]`	Create a two-dimensional array that contains four letters: a | b c | d It might help to visualise it in this way, although the computer does not store array contents in a table.
`myArray2[1][0] = "z"`	This would replace 'c' with 'z'.
`print(myArray2[0][1])`	Displays the requested letter, 'b' in this case.

i **Record** – a data structure that can accept multiple data items that do not need to be of the same data type. As far as Python is concerned, there is no difference between array, above, and records, so they are managed in the same way. One record might store a student's name, year group and average test score.

Python Code	Explanation
`student1 = ["Bob", 8, 89.2]`	Creates a record called 'student1'; note the different data types.
`student1[1] = 9`	'Bob' has been moved from year 8 to year 9.
`print(student1[2])`	Displays 89.2 – Bob's attendance.

If you are asked to name a suitable data structure for a specific situation, it can only be a record or an array (at least this is true at GCSE level). If the data structure is going to store multiple *different* data types, a record is always the right choice.

Strictly speaking, Python doesn't use arrays or records, but **lists** instead. Lists allow combinations of data types, so they behave more like records than arrays. However, when it comes to the exam, you are expected to be able to describe arrays, both one and two-dimensional, as well as records.

Input, Output and File Handling

i **Input** – the process of introducing data into a computer system. In this section, the keyboard will be used as the input device.

i **Output** – the process of communicating data beyond the system, typically to a human user. In this section, the visual display unit (VDU) will be used as the output device.

Python Code	Explanation
`name = input("Name: ")`	Asks the user for their name and stores the input in a variable called 'name'.
`age = int(input("Age: "))`	Asks the user for their age, then stores this as an *integer* in a variable called 'age'.
`height = float(input("Height (m): "))`	Asks the user for their height and stores this as a *real* number in the 'height' variable.
`print("hello")`	Displays the text within the speech marks.
`print(hello)`	Without the speech marks, 'hello' would be the name of a variable. This instruction would display the value stored in that variable.

i **File** – a store of data, used by a program, that continues to exist when the program, and even the computer itself, is no longer running. Many file formats exist, and the practical aspects of this chapter deal with text (.txt) files, typically associated with a program called 'Notepad'.

Python Code	Explanation

`file = open("file.txt", "w")`
`file.write("Hello World! ")`
`file.close()`
	Opens a file to write (w) and adds the text 'Hello World!' Line three not only closes the file but saves it as well. This text would replace anything already in the file.

`file = open("file.txt", "a")`
`file.write("Hello again!")`
`file.close`
	This has the same effect as the above example, but the 'a' means **append**, so the 'hello again' text is written at the end of the file's current content, instead of overwriting it.

`file = open("file.txt", "r")`
`print(file.read())`
`file.close()`
	The 'r' means 'read'. These instructions open the file and display the entire contents.

`file = open("file.txt", "r")`
`print(file.read(4))`
`file.close()`
	This code only displays the first four characters of the text in the file.

String Handling Operations

Python Code	Explanation

`firstName = "Richard"`
`lastName = "Lee"`
`fullName = firstName + " " + lastName`
	The third line uses the '+' operator to **concatenate** strings (join them together), along with a space. The concatenated string is then stored in its own variable, 'fullName'. This could be useful if a person's first and last names are stored in different locations, or entered into different text boxes.

`firstName = "Richard"` `print(len(firstName))`	Displays the **length** (the number of characters) in a string. This would output the number '7'.
`fullName = "Richard Lee"` `print(fullName.index(" "))`	Looks for one string within another. Here, the code will look for the space (" ") in the text 'Richard Lee'. This name is stored as an array of characters, line this: **0** R, **1** i, **2** c, **3** h, **4** a, **5** r, **6** d, **7** (space), **8** L, **9** e, **10** e The space is in **position** '7', so the number '7' would be output by this code.
`fullName = "Richard Lee"` `print(fullName[0:7])`	Displays a **substring** (part of a string), beginning at location '0' and going up to, but not including, the character at position 7. This would output 'Richard'.
`letter = 'a'` `print(ord(letter))`	The purpose of 'ord' is to **convert** the brackets' contents (letter, or 'a') to an ASCII character code. This would display the number 97.
`code = 97` `print(chr(code))`	This performs the **conversion** in reverse. The ASCII code 97 is displayed as its character, 'a'.

i **Casting** – converting a piece of data to a specific data type. For example, a user might enter a string, and the program might convert it to an integer, in order to allow certain calculations.

String → Integer	`a = '123'`
	`b = int(a)`
	The variable 'a' contains a string made up of numeric characters. The second line of code converts this string to the integer one hundred and twenty-three, storing that integer in the variable 'b'.

String → Real	`c = '123.456'`
	`d = float(c)`
	The variable 'c' now contains a string that looks like a decimal number. The second line converts it to a real data type, storing it in 'd'

Integer → String	`e = 789`
	`f = str(e)`
	This time, 'e' contains an integer, which is converted to a string and stored in 'f'.

Real → String	`g = 987.654`
	`h = str(g)`
	The real number 987.654, stored in 'g', is converted here to a string, which is then stored in 'h'.

Random Number Generation

i **Random** – a random number has been selected from a range of numbers if every number in that range had an equal chance of being selected.

Random integers in Python:

```
import random                    ← Imports the 'random' library, allowing random number generation
print(random.randint(0,9))       ← Outputs a random integer between 0 and 9
```

Random numbers have many uses in computing:

- Encrypting data, making it difficult for unauthorised people to understand
- Causing simulations, such as flight simulations, to run differently every time
- Adding variability to computer games; enemy units might have behaviour that varies
- Random sampling of survey participants, i.e. randomly selecting names from a list.

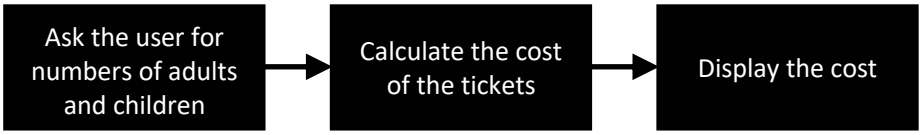
Subroutines

i **Subroutine** – a small subsection of the whole program that performs a specific, well-defined task. If a library program were to be broken into subroutines, one subroutine might issue a book, one might handle returning a book, one might add a new member, etc.

There are benefits to using subroutines to write a program:

- Each subroutine can be given to a different programmer, so working as a team is straightforward
- Subroutines can each be separately tested, without waiting for the whole program to be finished
- Subroutines that are commonly used, such as to sort a data set, can be reused in other programs, saving time and reducing errors

We could break a ticketing program into the following subroutines:



In Python, the subroutines would be declared like this:

```
def promptUser():

def calculateTotal():

def displayTotal(total):
```

The 'def' keyword means we are *defining* a subroutine. The code that belongs to each subroutine would then be placed immediately beneath each definition.

i **Parameter** – a piece of data that is passed into a subroutine in order for that subroutine to do its job. In the above example, 'total' is a parameter that will be passed to the 'displayTotal' subroutine. The other subroutines have no parameters. Multiple parameters can be managed using commas, e.g.
`def displayTotals(total1, total2, total3, etc.)`

The code within the subroutines will only happen if those subroutines are **called**.

i **Calling** – a process where an instruction in one part of the code tells another *named* part of the code to run. If you have a subroutine called 'promptUser', the code within that subroutine will never run without that subroutine being called.

```
promptUser()
total = calculateTotal()
displayTotal(total)
```

These three instructions tell the subroutines to happen in a particular order. Code can now be added to the three subroutines, so that the finished listing looks like the following. Notice how spacing greatly aids readability.

```
1     def promptUser():
2         global adults, children
3         adults = int(input("How many adults: "))
4         children = int(input("How many children: "))
5
6     def calculateTotal():
7         print("adults: " + str(adults))
8         if ((adults + children >= 5) or (adults >= 2)):
9             total = (adults * 9) + (children * 4.5)
10        else:
11            total = (adults * 10) + (children * 5)
12        print(total)
13        return total
14
15    def displayTotal(total):
16        print("Total cost: " + str(total))
17
18    promptUser()
19    total = calculateTotal()
20    displayTotal(total)
```

In Python, you have to call a subroutine after the code for that subroutine has been added. If the code on line 18 appeared before the code on line 1, this program would not run.

The explanation of exactly how this code works is below. Before you rush to read it, try to determine for yourself what each line does, and in what order the instructions would be executed. You'll probably make a few mistakes, but then the right answers will stick with you when you read them. There's no better way to learn about computer science than by making mistakes, as long as you learn from them.

NB the numbered list below tells you the order in which things take place – these numbers do not correlate with the line numbers in the program.

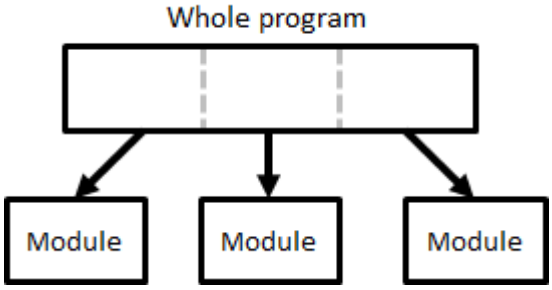
1. The first line to be executed is line 18, where the 'promptUser' subroutine is called. All lines above this belong to the subroutines, which will only run when they are called.
2. Program execution jumps to lines 1–4, where the 'adults' and 'children' variables are set according to user input. These variables are global, meaning they are visible in all other subroutines.
3. Once lines 1–4 have been executed, control jumps back to line 18, which is now complete (the 'promptUser' call is finished), and then on to line 19. The 'calculateTotal' subroutine is called, so control now jumps to lines 6–13.
4. Lines 6–12 calculate the cost of entry based on the number of adults and children.
5. The 'return' statement on line 13 passes the 'total' variable *back* to line 19, which is where the 'calculateTotal' subroutine was called from. Whatever the value of this variable was, it will now be stored in the 'total' variable on line 19.

i **Return** – a command in which a subroutine passes a piece of data *back* to the line of code from which it was called. This is effectively parameter passing, but in reverse. A **function** is a subroutine that returns a value; a **procedure** is a subroutine that does not.

6. On line 20, the 'displayTotal' method is called, with the contents of 'total' passed to it as a *parameter*. This value (received in brackets on line 15) is then output as part of line 16.

The best way to understand code is to write it. Using this program as a template, add more functionality. Try adding a discount code that takes 10% from the total. You might start by creating a 'promptForDiscountCode' subroutine.

i **Modularisation** – the process of breaking a program into smaller parts called **modules**. A subroutine is a type of module, and the advantages of dividing a program into either subroutines or modules are the same.



i **Scope** – refers to the visibility of a variable, and can either be **local** or **global**.

Python Code	Explanation

`vat = 0.2`
 	
`def sale():`
`  price = 1.99`
`  quantity = 5`
`  total = price * quantity * (1 + vat)`
`  print(total)`
 	
`def refund():`
`  amount = total / (1 + vat)`
	vat is **global** – declared outside of any subroutines, it is visible within *all* of those subroutines. You can see it used in calculations in both the 'sale' and 'refund' subroutines.
	
	The other variables (price, quantity, total, amount) are **local** – declared inside a subroutine, they are only visible from within that subroutine. The last line would cause an error here, as 'total' is local to the 'sale' subroutine, so wouldn't be visible in the 'refund' subroutine.

Local variables, along with parameters and return values, are usually a better choice than global variables. Another programmer, working on another subroutine, could modify global variables and make them behave unpredictably. This is not true of local variables.

When programming, you will find yourself giving names to a lot of aspects of your program, primarily subroutines and variables. When you choose a name, that name needs to describe the purpose of that subroutine or variable. That way, your code would make sense to other developers who may work on it. It would also make sense to you if you revisited it after some time. A variable called 'income' has a clear purpose, whereas a variable called 'var1' does not.

Robust and Secure Programming

Robust	Secure
A robust program continues to function even when confronted with unexpected events, such as a lost network connection or a user inputting data of the wrong data type.	A secure program prevents users from accessing or altering data that they should not access or alter. Usernames and passwords make up one approach for keeping a program secure.

i **Validation** – ensuring that data entered into the computer is reasonable and sensible. For example, checking that a person's date of birth isn't in the future. Validation does *not* ensure that data is correct.

Different types of data can be input, so a range of validation checks exist to suit:

Range check	ensures that data is within a specified range, e.g. ensuring race times for 100m are between 0 and 30 seconds, or ensuring a person's birthday is not after today's date.

```
isValid = False

while isValid == False:
    bedrooms = int(input("How many bedrooms: "))
    if bedrooms >= 1:
        isValid = True
    else:
        print("Please enter a positive number")
```

This code would keep looping until the user has entered a number of bedrooms that is greater than or equal to 1.

Type check	ensures that the correct data type has been entered (e.g. integer or Boolean).
Length check	ensures that a string contains a valid number of characters, such as in a postcode, phone number or National Insurance number.

```
isValid = False

while isValid == False:
    name = input("Name: ")
    if len(name) >= 3:
        isValid = True
    else:
        print("Minimum length: 3 characters")
```

This code asks a user's name repeatedly until they enter a string containing at least three characters. Replacing the values of '3' with '1' would check that the user has entered anything at all.

Lookup check	checks that what the user has entered exists on a list, such as a list of postcodes or days of the week. For shorter lists, the user can select the item from a list themselves.
Presence check	simply checks that the user has entered something. This is essentially a length check that ensures that the length is greater than zero.

i **Authentication** – the software process of ensuring that the person accessing a system is the person who is *supposed* to access that system. The following might be used:

- Usernames and passwords
- Memorable information – prompting for something only the real user should know, such as a favourite place or the name of a first pet
- Checking that the user is using their usual computer, by logging the IP address.

Authentication techniques are used throughout the cybersecurity world.


```
isLoggedIn = False

while isLoggedIn == False:
    name = input("Name: ")
    password = input("Password: ")
    if name == "user" and password == "pass":
        print("Valid login")
        isLoggedIn = True
    else:
        print("Invalid login")
```

This code asks a user's name and password repeatedly until they enter 'name' as their name and 'pass' as their password.

Selecting Test Data

Choice of test data is important. Suppose you have written a program for an estate agent in which they enter the number of bedrooms in a house, which must be an integer between 1 and 15:

Type of Test Data	Description	Examples
Normal (typical)	Data that is valid and that represents how the program would be used.	• 5 • 7 • 2
Boundary (extreme)	Data that is just barely valid, to check that the extreme ranges of normal input work correctly. It's also good practice to test data that is just *outside* the acceptable range.	• 1Classific • 15
Erroneous	Data that should not be accepted by the system. This is included to test whether a programs validation and error messages work correctly.	• -5 • 3.142 • "three"

If you have a piece of code that is supposed to sort eight positive integers into ascending order, the following sets of test data each have a specific purpose:

Test Data	Explanation
1 2 3 4 5 6 7 8	The data is already sorted, so the program should do nothing (which needs to be checked).
8 7 6 5 4 3 2 1	The values are in descending order, which requires everything to be moved; this is as far from being in ascending order as possible.
2 6 4 3 8 7 5	How does the program react to having a number of values other than eight?
-1 2 3 4 5 6 7 8	How does the program react to having a negative integer?
1 2 3 4.2 5 6 7 8	How does the program react to one piece of data being of the wrong data type?
5 4 4 2 9 8 8 5	Does the sort still work correctly if there are duplicate values?

Classification of Programming Languages

	High-level	Low-level
What is it?	High-level languages are more understandable to humans than low-level languages, so high-level languages are far more popular.	More difficult to understand, but can often be executed very quickly by computers.
Examples	• Java • Visual Basic • Python • C#	• Machine code (which consists of '1's and '0') • Assembly code, which is a way of making machine code slightly more readable by using mnemonics. For instance, the command 0110 1010 makes no sense, whereas a command like ADD 34, 32 gives a slightly better clue as to what is going on.
What do you need in order to run code written in these languages?	Either an interpreter or a compiler to enable the code that is typed to be translated into machine code so that the computer can run it.	Machine code does not need to be translated - it is machine ready. Assembly language requires an assembler.
What does code look like?	One line of code might do *several* things, e.g.: A = B + C This instruction finds out the values of B and C, adds them together, then stores the result in B. As one line of code can do several things, one line of a high-level language often translates into several lines of machine code.	One line of code performs a *single* task. In assembly language: LDA B ADD C STA A These instructions would perform the same task as 'A = B + C'. Machine code, to most people, would be a sequence of unintelligible '1's and '0's, but each line of assembly code translates to one line of machine code.
Suitability	• More appropriate if the program is to be used on a variety of different computer builds • Far more people are proficient in high-level than low-level languages, and this may dictate the language type used	• Likely to be used within embedded systems, where specific memory locations and processor functions can be addressed directly • Suited to time-critical applications, where execution must take place as quickly as possible.

Machine code is machine-specific, meaning a machine code program written for one computer will not necessarily work on another. This is because different computers sometimes have different internal layouts and components, which can affect the way that machine code is run.

i

Translator – a program that translates **source code** (which is written by the programmer) into **machine code** (which can be executed by the computer). There are three types of translator:

- **Assemblers** translate assembly language
- **Interpreters** translate then execute high-level code, one line at a time
- **Compilers** translate an entire high-level program before executing it.

Computers can only execute machine code. If a program is written using any other language, it must be translated to machine code before it can be executed.

For low-level translation, an assembler is the right tool for the job. For high-level translation, interpreters and compilers each have advantages and disadvantages.

Interpreter	Compiler
+ A program that contains errors can still be run, up to where the error exists + Debugging is easier, as the problematic line can be more easily pinpointed.	+ A compiled object code file will run more quickly than re-interpreting a source code file + It is more difficult for unauthorised people to modify a compiled object code file.
- Every time you run the program, it needs to be interpreted, which is time-consuming - It is easier for unauthorised people to access your source code.	- More memory is needed for the compilation process than for interpretation - The entire program needs to be error free in order for it to compile.

Sample Examination Questions

4. This question is based on the following program:

```
int total = 0                {keeps a running total}
int count = 0                {logs how many are entered}
float mean = 0               {to store the mean}
int input = <user input>
while input > -1
      total = total + input
      count = count + 1
      input = <user input>
end while
mean = total / count
output mean
```

   a) Give an example of each of the following from the code: [5]
      i. Variable
      ii. Comment
      iii. Arithmetic operator
      iv. Relational operator
      v. Iteration keyword

   b) When the program is running, what would the user need to do in order to signal that they have entered all of their numbers? [1]

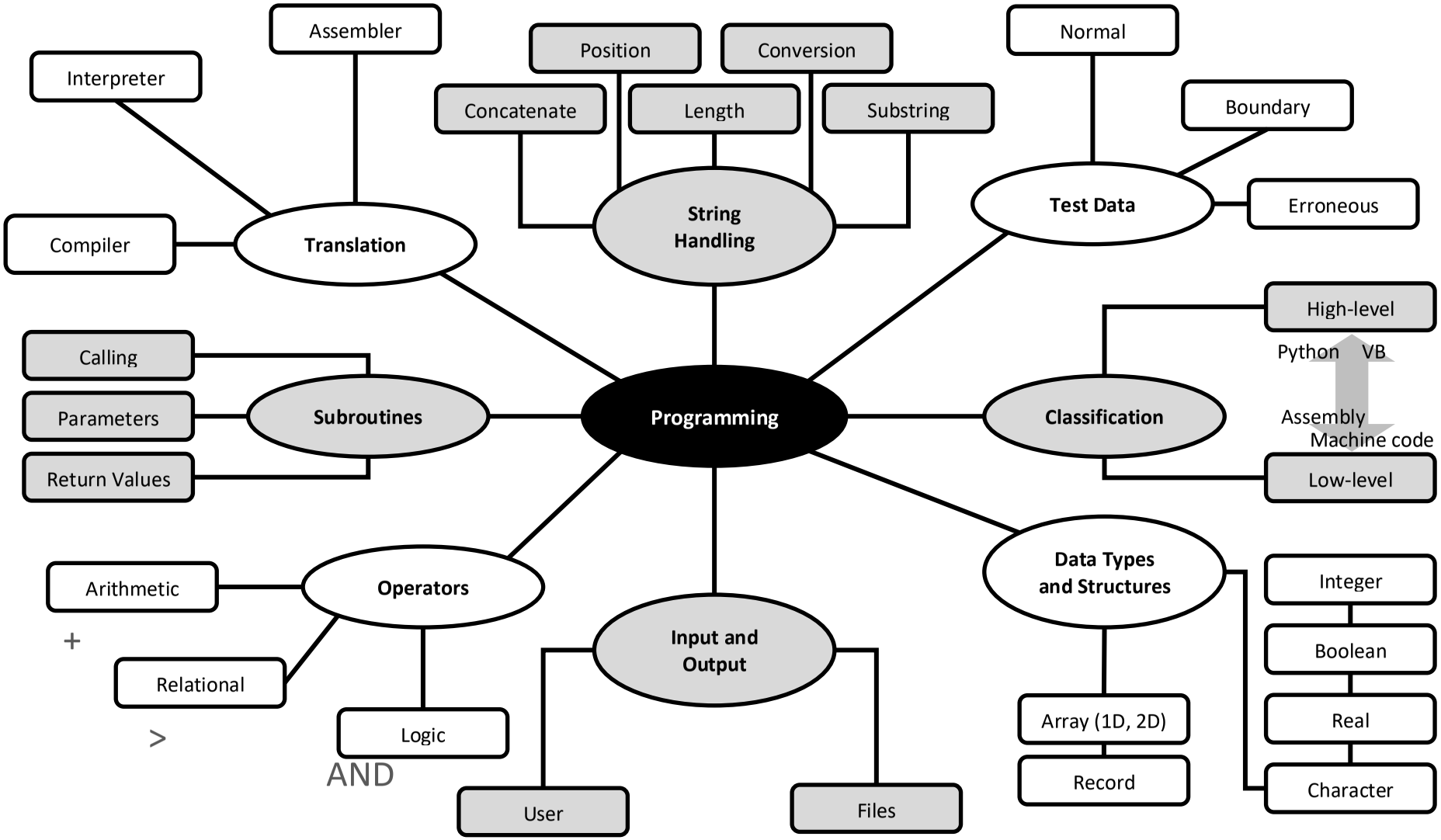
5. A bank stores several pieces of data about each bank account. Identify the most appropriate data type for each of the following pieces of data. Some data types are used more than once.

   Place **one** tick in each row. [6]

Data	Integer	Real	Boolean	String	Char
Money in account					
Account holder's name					
Number of whole years the account has been active					
Account holder's postcode					
Whether or not an overdraft is permitted					
A single-letter code that identifies the account type					

6. An array called DATA contains ten numbers, each being an integer between '0' and '10'. Usingpseudocode or a language programming language with which you are familiar, write an algorithm to count how many numbers in the array are higher than '5'. [4]

Programming Mind Map



3. Fundamentals of Data Representation

Number Bases

i **Number base** – the number of unique digits available in a numbering system. For example, in **decimal** (the numbering system you're most used to), there are 10 individual digits available (0, 1, 2, 3, 4, 5, 6, 7, 8, 9), so it is also known as **base 10**.

Numbering System	Available Digits
Binary (base 2)	0 1
Decimal (base 10)	0 1 2 3 4 5 6 7 8 9
Hexadecimal (base 16)	0 1 2 3 4 5 6 7 8 9 A B C D E F

Computers use binary to represent all data and instructions. Whatever a user enters, be it number, text, image, sound or video, it is ultimately stored as a string of '0's and '1's.

Converting Between Number Bases

Different numbering systems exist for whole numbers, fractions, positive numbers and negative numbers. At GCSE level, you need only to work with binary integers that have a decimal equivalent of between 0 and 255.

Binary → Decimal

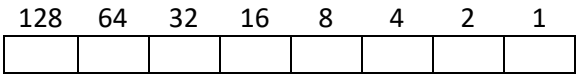
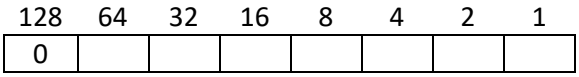
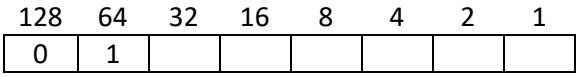
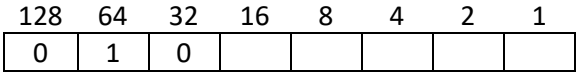
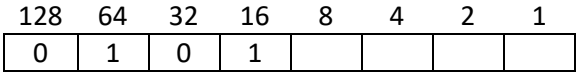
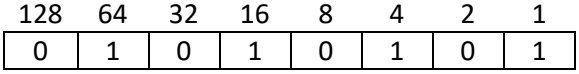
This conversion will use the example binary number `11001010`. The first step is to add placeholders above each binary digit. The placeholder above the rightmost bit is '1', and the value of each placeholder doubles each place to the left:

128	64	32	16	8	4	2	1
1	**1**	**0**	**0**	**1**	**0**	**1**	**0**

Add together the placeholder value that contain a '1':
128 + 64 + 8 + 2 = **202**

Decimal → Binary

The following steps show you how the number **85** is converted, with no need for a calculator:

Instruction	Answer so far	Remaining
We know that there will be eight bits in our answer, so we create a space for eight digits.		85
We can then write in the value of each digit immediately above. Start with '1' on the right-hand side, then double each time you add a new number to the left.		85
Now, we start with the left-most bit. 128 is higher than the number we're trying to convert, so we enter a '0'.		85
Next, we look at 64, which is lower than the number we're trying to convert, so we enter a '1' and subtract 64 from our number.		~~85~~ 21
The next number is 32, which is bigger than the number we're trying to convert (21 at this point, as we've subtracted 64 in our last step). We enter '0' and leave our number unchanged.		21
Our number (21) is larger than the next digit (16), so we enter a '1' and subtract 16.		~~21~~ 5
With only 5 left to convert, which will clearly be made up of a '1' and a '4', we place '1's into each of these columns and '0's into the others.		~~5~~ 0

The binary equivalent of '85' is '01010101'.

The right-most binary digit is always '1', then '2', '4', '8', etc., doubling each time. If the length of the binary number varies from eight bits, the left-most bit will change, but the right-most bit will always be '1'.

Hexadecimal

One hexadecimal digit can always be translated to four binary digits. Hexadecimal is used as shorthand, since it is quicker to write and less prone to being misread.

Binary Representation	Decimal Representation	Hexadecimal Representation
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	8
1001	9	9
1010	10	A
1011	11	B
1100	12	C
1101	13	D
1110	14	E
1111	15	F

To convert from binary to decimal:

1. This is a binary number we will convert to hexadecimal.	011110
2. If the binary number has a number of digits divisible by four (4-digit, 8-digit, 12-digit, etc.), it can be left alone. Otherwise, add '0's **to the left** until you have such a number. Since our number has six digits, we will add two '0's to the left of it.	00011110
3. Next, split the number into 'nibbles' of four bytes each.	0001 1110
4. Finally, convert each nibble separately, using the table above. This table contains every possible value for a binary nibble.	0001 = 1 1110 = E

So '011110' in binary is equivalent to '**1E**' in hexadecimal.

To convert from hexadecimal to binary:

1. This is the hexadecimal number we will convert to binary.	A6
2. Each hexadecimal digit will translate to a binary nibble, according to the table above. Translate each digit separately.	A = 1010 6 = 0110
3. Attach the nibbles together. If you choose to, you may leave a space between them for readability, but you do not have to.	1010 0110

So 'A6' in hexadecimal is equivalent to '**10100110**'.

If you need to convert between decimal and hexadecimal numbers, the best way is to go *through* a binary number, so either **decimal → binary → hexadecimal** or **hexadecimal → binary → decimal**, depending on the conversion you are asked to make.

Dealing with binary and hexadecimal conversions is more a case of practicing than memorising. Try the following:

- Convert numbers, generated at random, from decimal to binary to hexadecimal and back again, making sure you end with the same number that you started with.
- Write multiple choice questions for a revision partner; you'll get plenty of practice as you work out the answers yourselves, and trip them up with convincing (yet incorrect) answers.

Units of Information

Unit Name	Size	Example of Storage
Bit (b)	A single binary digit.	Either a single '1' or a single '0' – nothing else.
Byte (B)	A sequence of eight bits.	An individual keyboard character, such as '#' or 'k'.
Kilobyte (KB)	Approximately 1,000 bytes.	A paragraph of text, containing around 200 words.
Megabyte (MB)	Approximately 1,000 kilobytes or 1,000,000 (one million) bytes.	Around one minute of average quality mp3 music.
Gigabyte (GB)	Approximately 1,000 megabytes or 1,000,000,000 (one billion) bytes.	About ninety minutes of standard definition video.
Terabyte (TB)	Approximately 1,000 gigabytes or 1,000,000,000,000 (one trillion) bytes.	Depending on the quality, several hundred hours of video.

Be aware of the difference between the shorthand form for bits (lower case 'b') and that for bytes (upper case 'B'). If you look at your home broadband speed, it is probably measured in Mbps (megabits per second), not MBps (megabytes per second).

Binary Arithmetic

You need to be able to add two binary integers together as well as to perform binary shifts.

i **Binary shift** – moving the values of a binary number left or right.

a. 000**11**000 → Shift right by one place → 0000**11**00

b. 0000**11**00 → Shift left by two places → 00**11**0000

When you shift left or right, any bits that fall off the end are lost forever; any bits that join the number at the other end are *always* '0':

	Shift Left	Shift Right
Shift 1 place	Multiply by 2	Divide by 2
Shift 2 places	Multiply by 4	Divide by 4
Shift 3 places	Multiply by 8	Divide by 8

Addition of numbers in binary is similar to addition of numbers in decimal. The numbers are stacked, one on top of the other, and each pair is added, going from right to left. In binary, when adding two numbers, there are only five possible combinations of numbers, because we're only dealing with '1's and '0's:

0 + 0 = **0**
0 + 1 = **1**
1 + 0 = **1**
1 + 1 = **10** (seems strange, but '10' is binary for '2')
1 + 1 + 1 = **11** ('11' is binary for '3', and you will only need this one for carried numbers)

Binary Addition Walkthrough

1. The two numbers to be added are stacked one above the other.	10110101 + 00111100
2. Starting from the rightmost digits, the first pair is added together. 1 + 0 = 1.	10110101 + 00111100 1
3. The next pair is just as straightforward; 0 + 0 = 0.	10110101 + 00111100 01
4. As for the next pair, 1 + 1 = 2, which is 10 in binary. Just as we did in adding decimal numbers, we must carry the '1' and place the '0' in the answer.	10110101 + 00111100 001 1
5. In the next column, we're adding 0 + 1, but we also need to add the carried '1'. In binary, 0 + 1 + 1 = 10. This means another '0' in our answer and another carried '1'.	10110101 + 00111100 0001 1 1
6. This time, we're adding three '1's – two from the calculation and another that was carried. In binary, 1 + 1 + 1 = 11.	10110101 + 00111100 10001 1 1 1
7. The next calculation is identical to the previous one. We're carrying out 1 + 1 + 1 = 11.	10110101 + 00111100 110001 1 1 1 1
8. Next, 0 + 0 = 0, but adding the carried '1' gives us a total of '1'.	10110101 + 00111100 1110001 1 1 1 1
9. The final pair of numbers, 1 + 0, is straightforward.	10110101 + 00111100 11110001 1 1 1 1

At GCSE level, you will **not** be required to factor the following into addition:

- Fractions
- Numbers that require more than 8 bits (including the result of addition)

Character Encoding

i **Character** – a single symbol, such as a letter, number, symbol or space. Most keys on a keyboard cause one character to appear on the screen.

i **Character set** – a list of all characters recognised by a computer system. Each character has a corresponding code, and the same codes are used by all computers that use the same character set. **ASCII** (American Standard Code for Information Interchange) and **Unicode** are two common character sets.

Character Value	ASCII Value	Unicode Value
A	100 0001	0000 0000 0100 0001
a	110 0001	0000 0000 0110 0001
#	010 0011	0000 0000 0010 0011
3	011 0011	0000 0000 0011 0011

Characters are commonly grouped and run in the order that you would expect, i.e. 'A' in ASCII has a decimal representation of 65. 'B' is 66, 'C' is 67, 'D' is 68, etc. In lower case, 'a' is 97, 'b' is 98, 'c' is 99, etc. Numerals (0, 1, 2, 3, etc.) are similarly grouped together.

ASCII uses seven bits, meaning 128 different characters (2^7) can be represented.

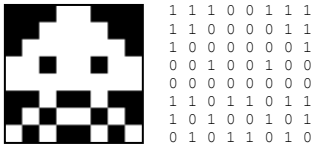
Unicode uses sixteen bits, meaning 65,536 different characters (2^{16}) can be represented. Using Unicode instead of ASCII gives you access to far more alphabets, including Chinese, Japanese, Arabic and Russian, but more storage space is required.

ASCII and Unicode use the same codes for characters up to code 127 (`111 1111` in binary), e.g. 'A' is `100 0001` in ASCII and `0000 0000 0100 0001` in Unicode.

Representing Images

An image is divided into pixels, each of which is a tiny dot on the screen. A pixel can only be one colour at a time, and when a picture is saved, the colour of each individual pixel must be stored in binary. The more bits that are used to store each pixel, the more colours are potentially available.

i **Pixel** – short for *picture element*, this term refers to the smallest possible unit within an image or on a screen. A pixel cannot be divided up into smaller units, and a pixel can only ever be one colour at a time. VDUs display output using millions of pixels.



When working with **monochrome** images, pixels need only be turned on or off, so one bit can represent one pixel.

The image above (which has been enlarged) is eight pixels by eight, so sixty-four pixels in total. Only black and white are used, so a single bit would be enough to store each pixel, set to '0' for black or '1' for white. This means sixty-four bits, or eight bytes, would be enough storage space for the pixels of this image.

If more colours are needed, more bits are needed. Many images store twenty-four bits, three bytes, *for each pixel*, in this way:

First byte	Second byte	Third byte
11111111	10001011	00000000
Red	Green	Blue

The 'red' value in the first byte is as high as it can be, so there will be lots of red in the colour of this pixel. There will be some green, but not as much as red, and there will be no blue at all, as the value of the last byte is zero. Over sixteen million colours are available, but a sixty-four pixel image saved in this format would need 192 bytes to store all the pixels, compared with eight for the image above.

The amount of storage required for an image depends on a number of factors, including:

- **Colour depth** – a measure of how many colours are available; the more colours that are available, the more bits that must be assigned to store each pixel
- **Size in pixels** – the number of pixels in *height* and *width* for an image. More pixels require more storage space than a lower-resolution image, but result in an image of higher quality.

Calculating Image Size

W	The width of an image, measured in pixels
H	The height of an image, measured in pixels
D	The colour depth; the number of bits used to store each pixel

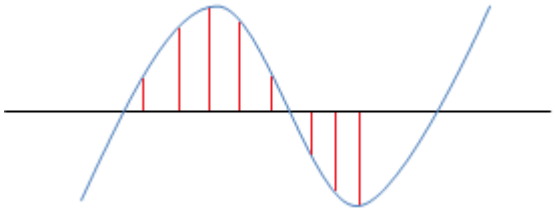
File size in **bits**: W × H × D
File size in **bytes**: (W × H × D) / 8

To convert file size from **bytes** to **kilobytes**, divide the number of bytes by 1,000. To convert from **bytes** to **megabytes**, divide the number of bytes by 1,000,000.

Representing Sound

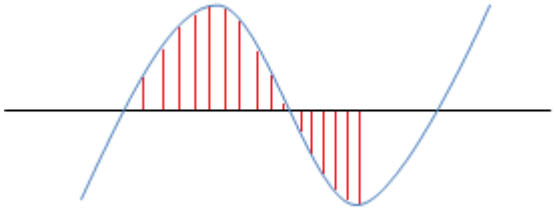
i **Analogue** – sound is an analogue signal, which is the opposite of digital. A computer usually works with digital signals, processing '0's and '1's – nothing else, nothing in between. An analogue signal is continuously variable. It might be one of two values, such as '0' or '1', or anything in between, such as '0.1879'. Sound is an analogue signal, since frequencies do not occur at set points.

Analogue data needs to be converted to digital in order to be stored and processed. This is done by taking regular **samples** of the sound's **amplitude**. With sound, thousands of samples are taken per second, each being a measure of the amplitude at a specific point in time:



(each vertical line represents a sample)

i **Sampling rate** – a measure of how often a sample is taken, measured in Hertz (Hz). 1 Hz means once per second. 1 MHz (megahertz) means one million times per second. The image below represents a higher sampling frequency than in the one above.



A higher sampling rate results in higher accuracy but requires more storage space.

i **Sample resolution** – not to be confused with screen resolution, which is a measure of the number of *pixels*. Resolution in sound sampling refers to how many bits are required to store each sample.

Calculating Sound File Size

Sampling rate – the number of samples per second
Sample resolution – the number of bits used to store each sample
Seconds – the length, in seconds, of the whole sound file

File size in **bits**: Rate x Resolution x Seconds
File size in **bytes**: (Rate x Resolution x Seconds) / 8

Data Compression

i **Compression** – techniques to reduce the size of a file, so that it takes up less storage space and can be transmitted across a network more quickly. There are different types of compression, far beyond those covered at GCSE level.

Choice of compression type might depend on a number of factors:

- If data needs to be precise, such as in money transactions, lossless compression would be used
- If data does not need to be precise, typically as in photographs or music, lossy compression might be considered to save disk space or transmission time.

Run Length Encoding

i **Run length encoding** – a form of compression that is effective when dealing with repeating data. Instead of storing each item of repeated data, the data is stored once, along with how many times it repeats.

A run-length encoding algorithm is a straightforward means of lossless compression. It would be effective in storing or transmitting a file that contained lots of repeated data, such as:

- A text file that contains repeated characters, e.g. 'zzzzzzzzzzzzz'
- An image file where lots of adjacent pixels are an identical colour
- A sound file containing stretches of complete silence.

If we go with the example of a text file containing repeated characters, compression using 'run length encoding' might operate in the following way:

Uncompressed: `QQQQQQQQWWWWWWEEEEEEEEEEEEEEEQQQQQQQQ`
Compressed: `8Q6W15E8Q`

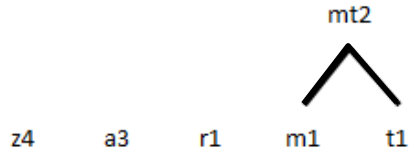
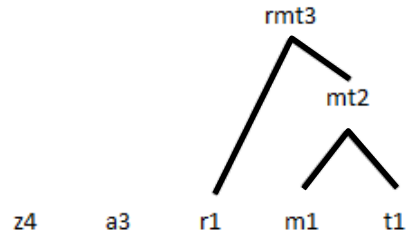
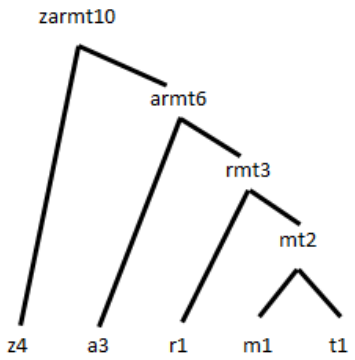
Each block of one letter is now reduced to that letter, once, alongside a number representing how many consecutive instances exist. Eight individual 'Q's has now become 8Q. This would not work for every file you attempt to compress:

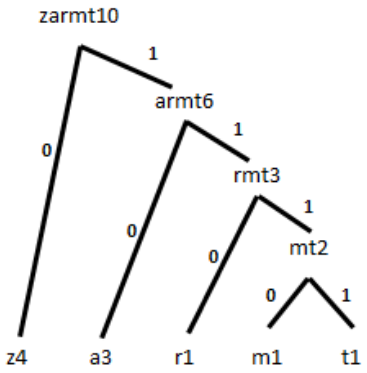
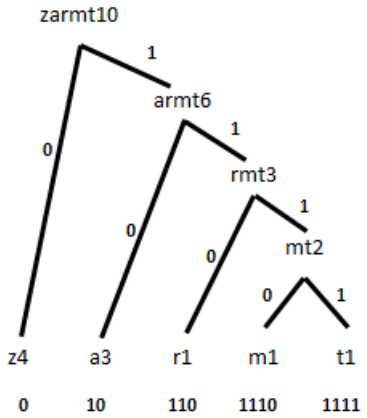
Uncompressed: `QWERTY`
Compressed: `1Q1W1E1R1T1Y`

Because no letters were repeated in the uncompressed file, the 'compression' process has actually resulted in a larger file.

Huffman Encoding

i **Huffman encoding** – a form of compression that represents commonly-used characters with smaller bit patterns, and rarely-used characters with larger bit patterns.

Begin with the text, which would, in most cases, be more than a single word	`razzmatazz`
Identify all unique letters	`r a z m t`
Count how often each letter appears	`r1 a2 z4 m1 t1`
At the bottom of your page, write the letters down in descending order of how often they appear	`z4 a3 r1 m1 t1`
Create a 'V' linking the two **lowest-frequency** letters, and write the combination of the letters, and their total frequency, at the peak of the 'V'	
Repeat the process, this time remembering that 'mt2' is now a possible option and that 'm1' and 't1' are not	
Continue until all letters have a path, however long, to the top of the tree	

Label each left branch with a ‘0’ and each right branch with a ‘1’	
Give each letter a binary code made up of all the ‘1’s and ‘0’s encountered, in order, from the top of the tree	

This is a **Huffman tree**

To write ‘razzmatazz’, using this encoding, the following bit strings would be used in order:

`110 10 0 0 1110 10 1111 10 0 0`

In fact, the spaces can be removed, since there is no way one code could be mistaken for another:

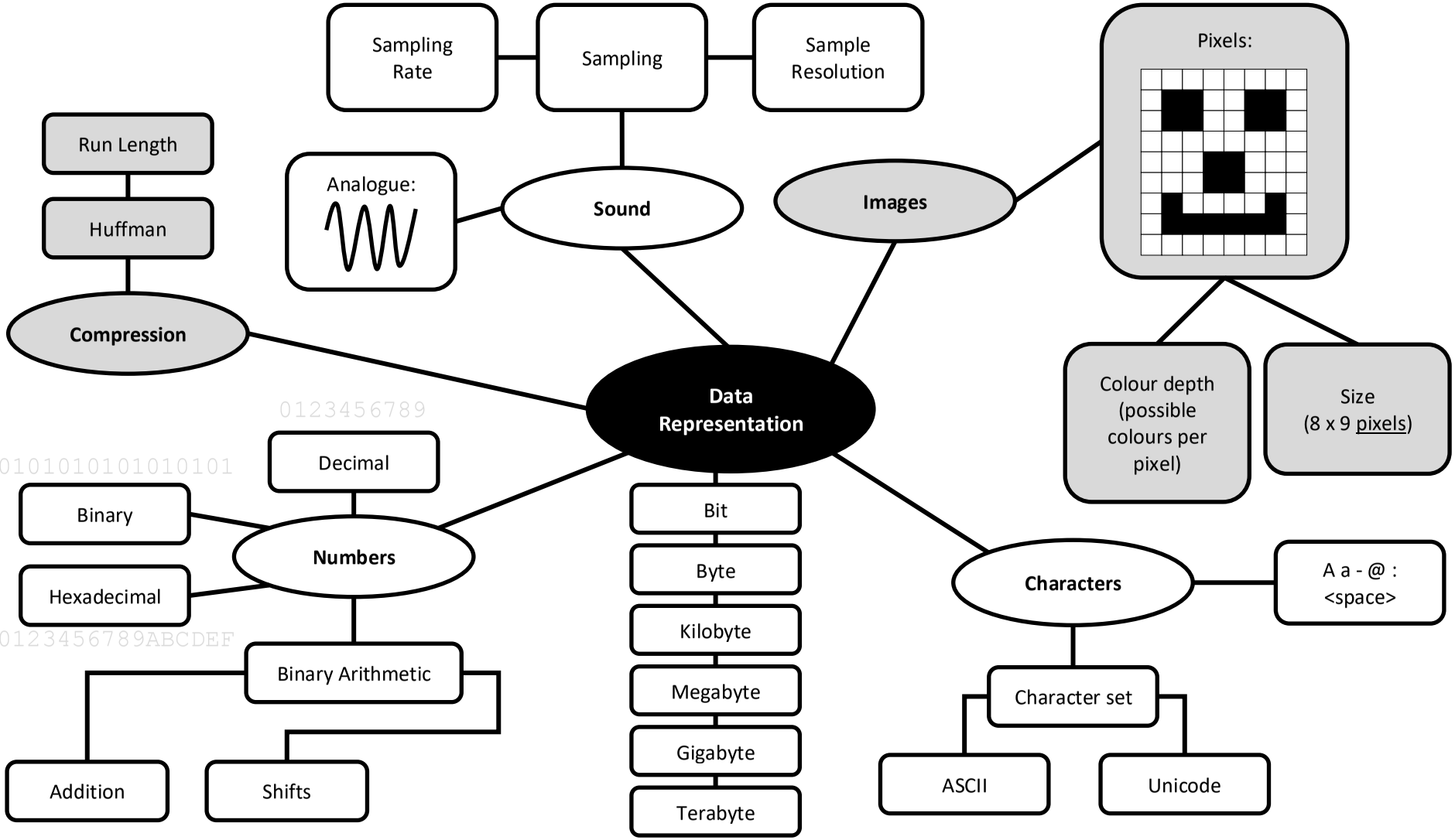
`11010001110101111 1000`

An ASCII representation of the word ‘razzmatazz’ would have required 70 bits (10 characters, multiplied by seven bits per character). The Huffman encoded representation, as you can see, requires only 21 bits. This is because the most commonly used character, ‘z’, is given the shortest code, ‘0’. Of course, much of this saving would be lost due to the fact that the Huffman tree itself would also need to be transmitted. However, for larger amounts of text, a substantial saving can be made in terms of file size, even when the size of the Huffman tree is factored in.

Sample Examination Questions

7. The binary number `10011011` can have several values when translated into other number systems.
   a) Convert it to decimal. [1]
   b) Convert it to hexadecimal. [1]
8. a) Carry out an arithmetic shift left by two places on the binary number 00011000.
   b) State the effect of an arithmetic shift by two places.
9. a) State what is meant by the term **character set**. [1]
   b) The ASCII code for ‘P’, in decimal, is 80. What is the corresponding code for ‘M’? [1]
10. Each pixel in an image uses 8 bits of storage, and the resolution of the image is 640x480 pixels.
   a) i) What is meant by the term **colour depth**? [1]
      ii) State the colour depth used by this image. [1]
   b) How much storage space is required for this image? Provide your answer in **kilobytes**. [3]

Fundamentals of Data Representation Mind Map



4. Computer Systems

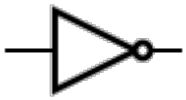
Hardware and Software

i **Hardware** – the physical components that make up a computer system, including processors, memory, storage, and input and output devices.

i **Software** – the programs that run on a computer, including operating systems, utility programs and application software (which includes mobile apps).

Boolean Logic

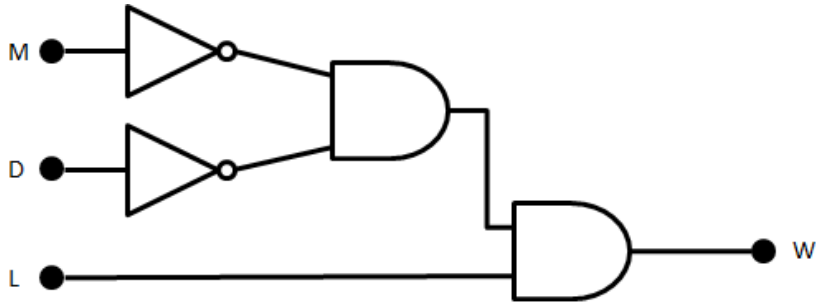
i **Logic** – in computer science, 'logic' refers to producing **Boolean** outputs (0 or 1; true or false) based on combinations of Boolean inputs.

<table>
<tr><th>Logic Expression</th><th>Gate</th><th>Truth Table</th></tr>
<tr><td>AND
all inputs must be '1' for the output to be '1'</td><td>  </td><td>
<table>
<tr><th>A</th><th>B</th><th>A AND B</th></tr>
<tr><td>0</td><td>0</td><td>0</td></tr>
<tr><td>0</td><td>1</td><td>0</td></tr>
<tr><td>1</td><td>0</td><td>0</td></tr>
<tr><td>1</td><td>1</td><td>1</td></tr>
</table>
</td></tr>
<tr><td>OR
if either input, or both inputs, is '1', the output is '1'</td><td>  </td><td>
<table>
<tr><th>A</th><th>B</th><th>A OR B</th></tr>
<tr><td>0</td><td>0</td><td>0</td></tr>
<tr><td>0</td><td>1</td><td>1</td></tr>
<tr><td>1</td><td>0</td><td>1</td></tr>
<tr><td>1</td><td>1</td><td>1</td></tr>
</table>
</td></tr>
<tr><td>NOT
the output is simply the opposite of the input</td><td>  </td><td>
<table>
<tr><th>A</th><th>NOT A</th></tr>
<tr><td>0</td><td>1</td></tr>
<tr><td>1</td><td>0</td></tr>
</table>
</td></tr>
</table>

A truth table can be more complex. If we consider a variation on the greenhouse example described in the 'embedded systems' section above, we might want to turn on an automatic sprinkler (water = 1) only if the soil is dry (moisture = 0), it is daytime (light = 1) and the door is closed (door = 0). In all other events, the sprinkler is turned off (water = 0).

Inputs			Output
Moisture M	*Light L*	*Door D*	*Water W*
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0

One way to draw the logic circuit for this truth table would be as follows:



Software Classification

System software – programs that are needed for effective communication with the hardware, and for launching application software. Examples include:

- Operating system, such as Windows and Android
- Utility software, such as antivirus and compression software

Application software – programs launched from the operating system, by the user, to perform a specific task or group of tasks. Examples include:

- Desktop applications such as browsers and word processing packages
- Mobile apps

The Operating System

Operating system – a piece of system software that acts as an interface between the user and the hardware, managing all hardware and all other software. If another piece of software is to be launched, it will be launched from the operating system.

Operating systems are complex pieces of software, often requiring many years and many people to develop them. The reason that they are complex is simply that computers are complex, with much hardware and software that requires management. Among many things managed by the operating system are:

Processor(s)	The operating system decides… • which processes will be carried out by which processors • if multiple processes are running, which one the processor should handle next • how long a **time slice** a process should be given, i.e. how long before the processor's attention switches to the next process
Memory	The operating system… • loads programs and data from backup store to main memory • removes unneeded programs and data to make room for more • manages **virtual memory** where part of secondary storage is used as an overflow area for main memory
I/O devices	regarding **input/output** devices, the operating system… • acts as a go-between, passing data from input → application software, or application software → output • manages device drivers, which are programs telling the operating system how to communicate with attached input/output devices
Applications	The operating system… • communicates between application software and hardware • processes requests from application software for resources, such as a network connection or a remotely stored file
Security	The operating system can… • manage multiple user accounts, keeping users' data separate • automatically back up data, thereby increasing its security • handle usernames and passwords to prevent unauthorised access • recognise one user as an administrator, who would have greater access rights

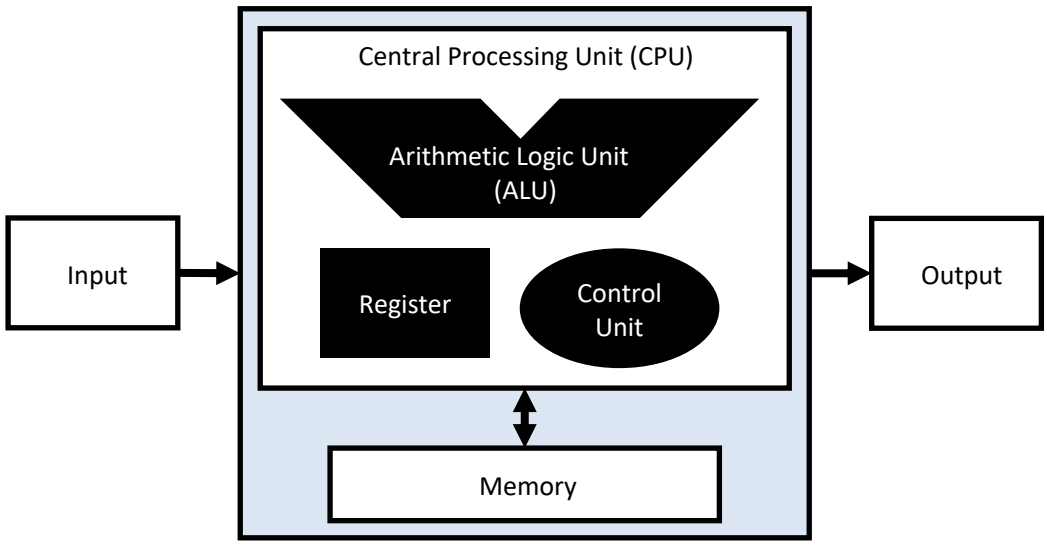
Uility Software

i **Utility software** – programs that keep the computer functioning efficiently, perhaps by freeing up storage space, removing viruses or ensuring the files are backed up.

Utility	Purpose
Compression	Reducing the size of a file so that it can be stored using less space or transmitted more quickly.
Defragmentation	Moving separate parts of a file physically together, to speed up disk access.
Backing up	Creating a copy of files, either on the same disk, on a backup device or in the cloud. Backing up can be either full or incremental: • **Full** back-up involves creating a copy of all files • **Incremental** back-up involves creating a copy only of files that have been created or edited sine the last back-up.
Encryption	Allowing for data to be scrambled in order to prevent unauthorised individuals from understanding any files that they see. This might be for secure storage or secure transmission.

Systems Architecture

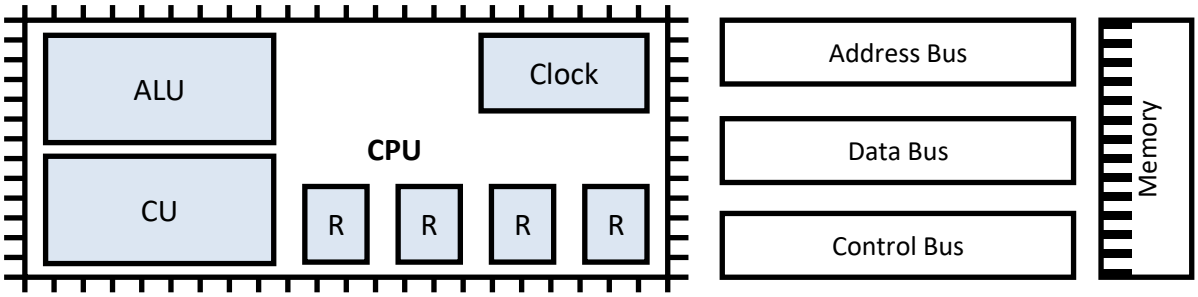
The Von Neumann Architecture



Data is input, processed by the CPU, which has several components of its own, and output. There is also memory, allowing data to be stored. The arrows in this diagram indicate **buses**.

This model defines the behaviour of many computers. The most advanced gaming PC has a processor (or several), an input in the form of a keyboard and mouse, an output in the form of a display and speakers, and vast memory to store the state of the game. A digital alarm clock, which is also a computer, has an input (the user can set the alarm), outputs (the current time, and the alarm sound), processing (keeping track of what time it is) and memory (it needs to remember the time of the alarm).

i **CPU** – the central processing unit executes program instructions, performing calculations and comparisons, as well as coordinating the behaviour of other hardware. The CPU comprises several different components.



Component	Function
Arithmetic Logic Unit (ALU)	Performs various operations: • Arithmetic operations (+ - * /) • Relational operations (< > =) • Logic operations (AND, NOT, OR).
Control Unit (CU)	Manages the execution of instructions by coordinating the activities of the other hardware.
Buses (collections of wires that transmit data between computer components)	**Data bus** – moves data back and forth between the CPU and memory.
	Address bus – transmits memory locations. Any data retrieved from or placed into memory belongs in a specific address or memory location.
	Control bus – transmits commands to other components, such as READ (get data from memory) or WRITE (put data in memory).
Registers	Short-term storage for small, specific pieces of data, within the CPU itself. One register would store data just retrieved from memory; another would store the memory location from which that data came.
Clock	Every action performed by the CPU must begin during a clock **pulse**. A clock generates a pulse billions of times each second (in modern PCs), synchronising the activities within the CPU.

Main Memory

i **Main memory** – the main working area for data currently being used and programs currently running.

There are different types of main memory:

Term	Definition
RAM	Random Access Memory. When a program is loaded from a computer's hard disk, its data and instructions are loaded into RAM, which is generally much smaller but much faster than a hard disk. When a computer is turned off or subject to a loss of power, the content of RAM is lost. This means that RAM is **volatile**. If a computer has more RAM, it will be able to run more applications at the same time.
ROM	Read Only Memory. 'Read Only' means that the content cannot be edited or deleted (meaning it is not volatile). As such, ROM stores data or instructions that will not need to be updated. ROM will typically store **bootstrapping** instructions, which tell the computer the initial steps in finding and initialising the operating system when the computer is turned on.
Cache Memory	Cache memory stores copies of data or instructions from RAM that are accessed very regularly. This means that these data or instructions can be accessed very quickly, although a computer's cache memory will usually be very small, measured in kilobytes or megabytes rather than gigabytes (see 'Data Representation').

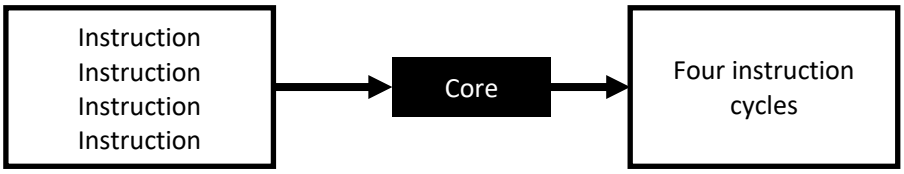
i **CPU performance** – a CPU with a higher rate of performance can execute instructions more quickly than a CPU with a lower rate of performance. Several factors can affect a CPU's performance:

- Number of cores
- Clock speed
- Size of cache
- Type of cache

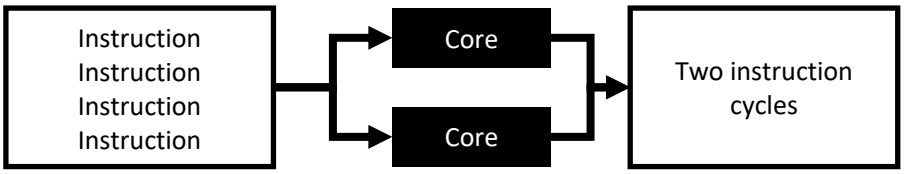
Number of Cores

i **Core** – a single unit, comprising an ALU and a control unit, which can execute instructions. Two cores can execute instructions at the same time, so more cores means more instructions per second that can be executed.

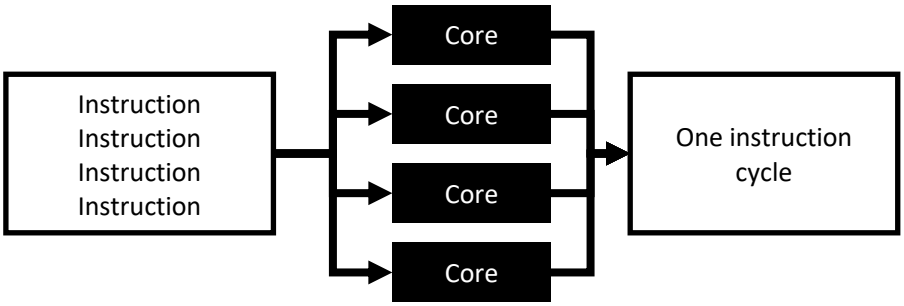
Four instructions with once core:



Four instructions with two cores (a **dual core** processor):



Four instructions with four cores (a **quad core** processor):



A dual core processor is not quite twice as fast as a single core processor, because some time is typically lost to organising which core will follow which instructions (this time is called the **overhead**). The overhead is, however, usually small enough that a dual core processor can be considered twice as fast as a single core processor with the same clock speed.

Clock Speed

i **Clock speed** – the number of clock pulses per second, typically measured in gigahertz (GHz). A 3GHz processor has a clock that pulses three billion times per second, meaning there are three billion opportunities, each second, for an instruction to begin. Note that instructions might take longer than a single clock pulse to complete.

Cache size

Cache memory, as stated above, is used to store data and instructions that will be accessed repeatedly. If more cache memory is available, a larger amount of data and instructions can be stored there. Since cache memory is faster than any other form of memory, this would improve CPU performance.

Cache type

Cache memory can be L1 (level 1), L2 or L3. L1 cache is the smallest in terms of capacity, but it is also the quickest, as it can be found directly on the CPU. If the CPU needs a piece of data, it will first look in L1 cache. Failing that, it will check L2 cache (which is larger but slightly slower), then L3 cache (larger and slower still), then main memory (larger and slower than L3 cache).

To summarise, CPU performance can be enhanced by:

- increasing the number of cores
- increasing clock speed
- increasing the size of cache memory
- maximising use of L1 cache

The Fetch-Execute Cycle

i **Fetch-execute cycle** – a continual sequence of tasks that results in instructions being *fetched* from main memory, *decoded* so that the CPU knows what to do with them, then *executes* them, i.e. carries them out. This cycle happens many millions of times per second in a modern PC.

Fetch

1. The clock pulses. On a 3 GHz processor, this would happen three billion times a second. The processor can only begin a task as the clock pulses, although a task may take more than one clock pulse to complete
2. One of the registers contains the location (in memory) where the next instruction is to be found; this register is usually called the '**program counter**'
3. This location, also called an 'address', is transmitted along the address bus to memory
4. Memory responds by sending the content of that address along the data bus, back to the processor. In this case, an instruction is being transmitted, and it is stored in a register called the '**Instruction Register**'

Decode

5. The instruction is read by the control unit, which prepares the registers for whatever data they will be storing; the nature of this preparation depends on what the instruction was

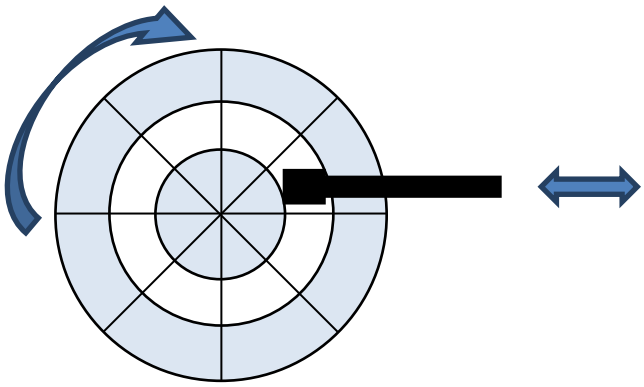
Execute

6. Here, the instruction is carried out. It might be one of the following:
   - Retrieve a piece of data from memory and store it in a register
   - Perform some operation on data that is already in a register; if this is any kind of calculation or comparison, the ALU will carry out this part of the task
   - Store something currently in a register, perhaps that has been processed in some way, in memory
7. The program counter, which indicates the location of the next instruction, is updated because the current instruction has been executed. *Now, the entire cycle begins again*.

The terms 'fetch-execute cycle' and 'fetch-decode-execute cycle' refer to the same process. There is a 'decode' phase, whichever term is used.

Secondary Storage

i **Secondary storage** – long-term storage in a computer system, necessary because main memory is volatile (loses its contents when powered down) and will also run out of storage space. Three types of backing store are **optical**, **magnetic** and **solid state**. All of these devices are required to store vast numbers of '1's and '0's, but they do so in different ways.

<table>
<tr><th>Device</th><th colspan="2">How it Works</th></tr>
<tr><td rowspan="2">Optical</td><td colspan="2">Optical storage media, including CDs, DVDs and Blu-Ray disks, are written to and read from using lasers. Just like magnetic disks, optical disks spin to allow the laser to read the data from the correct location.
 
The surface of these disks have billions of locations in which holes can be physically burned by a laser, or not burned, to represent either a '0' or a '1':</td></tr>
<tr><td>+ One optical disc is very cheap
+ Often read-only, so difficult to accidentally overwrite data</td><td>- An unprotected disk is vulnerable to being scratched
- Low data capacity</td></tr>
<tr><td rowspan="2">Magnetic</td><td colspan="2">Magnetic disks are round, although they are usually built into rectangular housing. The surface of such a disk is divided into tracks and sectors:
 
Here, there are three tracks that are circular in shape, and each track is divided into eight segments. The disk will spin to allow a read-write head (displayed in black) to access the data in a particular segment. This will happen quickly with internal hard disks, which spin at 7,200 revolutions per minute (rpm). Within each sector, billions of magnetic particles exist that are either magnetised (1) or not magnetised (0).</td></tr>
<tr><td>+ The cheapest storage medium per megabyte
+ Less cumbersome than multiple optical discs</td><td>- Slower access speeds than solid state
- Unlike optical discs, data is all on one device, which can be lost</td></tr>
<tr><td rowspan="2">Solid State</td><td colspan="2">Solid state storage devices (of which flash drives are one type) use electronic circuits, and no moving parts, to store data electronically. With no moving parts, these devices are both sturdier and quieter than magnetic or optical devices. SSDs (solid state disks) use electrical circuits which, unlike those of RAM, are non-volatile, meaning they retain data without a power supply.</td></tr>
<tr><td>+ Faster access speeds than both optical and magnetic
+ With no moving parts, harder to damage</td><td>- The most expensive form of storage per megabyte
- Limited number of times each bit can be written to</td></tr>
</table>

i **Cloud storage** – this involves storage on remote computers, usually managed by other organisations. When a file is saved or loaded, it is transmitted across the Internet, and multiple backups of files often exist around the world. Cloud storage uses magnetic and, increasingly, solid state storage.

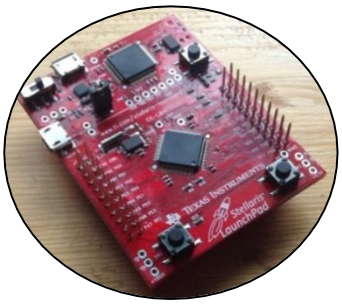
+ With data stored remotely, and usually in multiple physical locations, it is less likely to be damaged by fire, etc. or misplaced
+ Capacity on your local machine is freed up, granting you more storage space

– Cloud storage often comes with a subscription fee
– The speed at which you can access your data is limited by your Internet connection; it might take days to access a large video file

Embedded Systems

i **Embedded system** – a computer that exists within a larger mechanical or electrical device such as a microwave oven or a guided missile. Embedded systems are used when a whole computer is not viable (as in a microwave) or when they have only a single, specific purpose (as in a guided missile).

At the heart of an embedded system is a **microcontroller**, which is a chip that contains both storage and processing capabilities. To the right, you can see a microcontroller board. In the centre is a black square, the microcontroller itself. On both the left and right edges are pins, which can be set to input or output mode.



The microcontroller's code usually runs in a loop, continually monitoring inputs, processing them and producing outputs. The processing power in such a device is usually considerably less than a modern PC, but these devices are much cheaper.

Embedded systems	Non-embedded systems
• Automated teller machines (ATMs) • Printers • Utility smart meters • Petrol pumps	• Personal computers • Laptops • Tablets • Smartphones

Sample Examination Questions

11. A system that automatically waters a lawn will turn on sprinklers if **both** of these conditions are true:
    - The temperature is above20 degrees Celsius
    - The lawn was last watered more than 5 days ago

    a) State the name of the logic operator that applies here. [1]

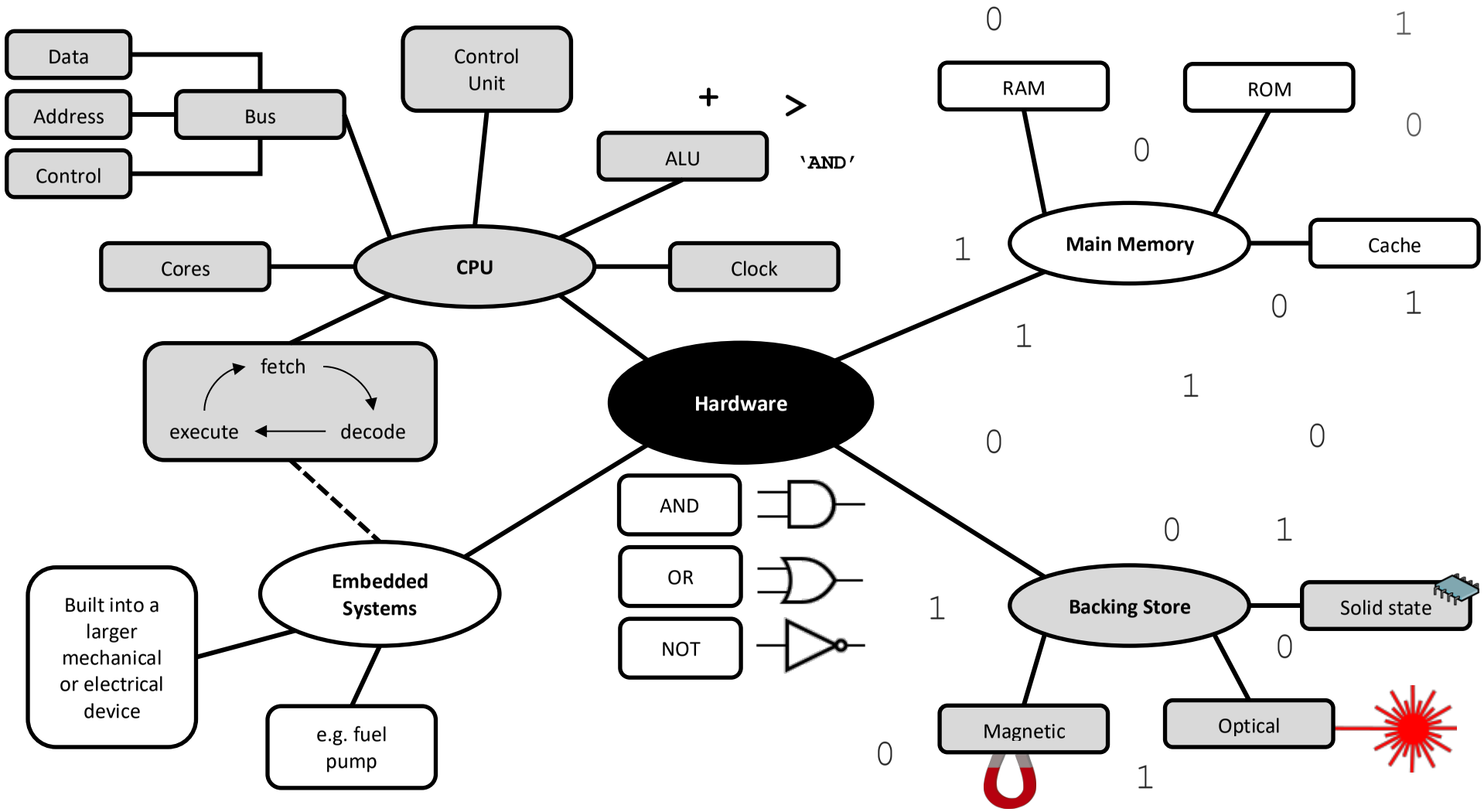
    b) Draw the truth table for this logic operation. [2]

12. Describe, with an example, what is meant by the term **embedded system**. [3]

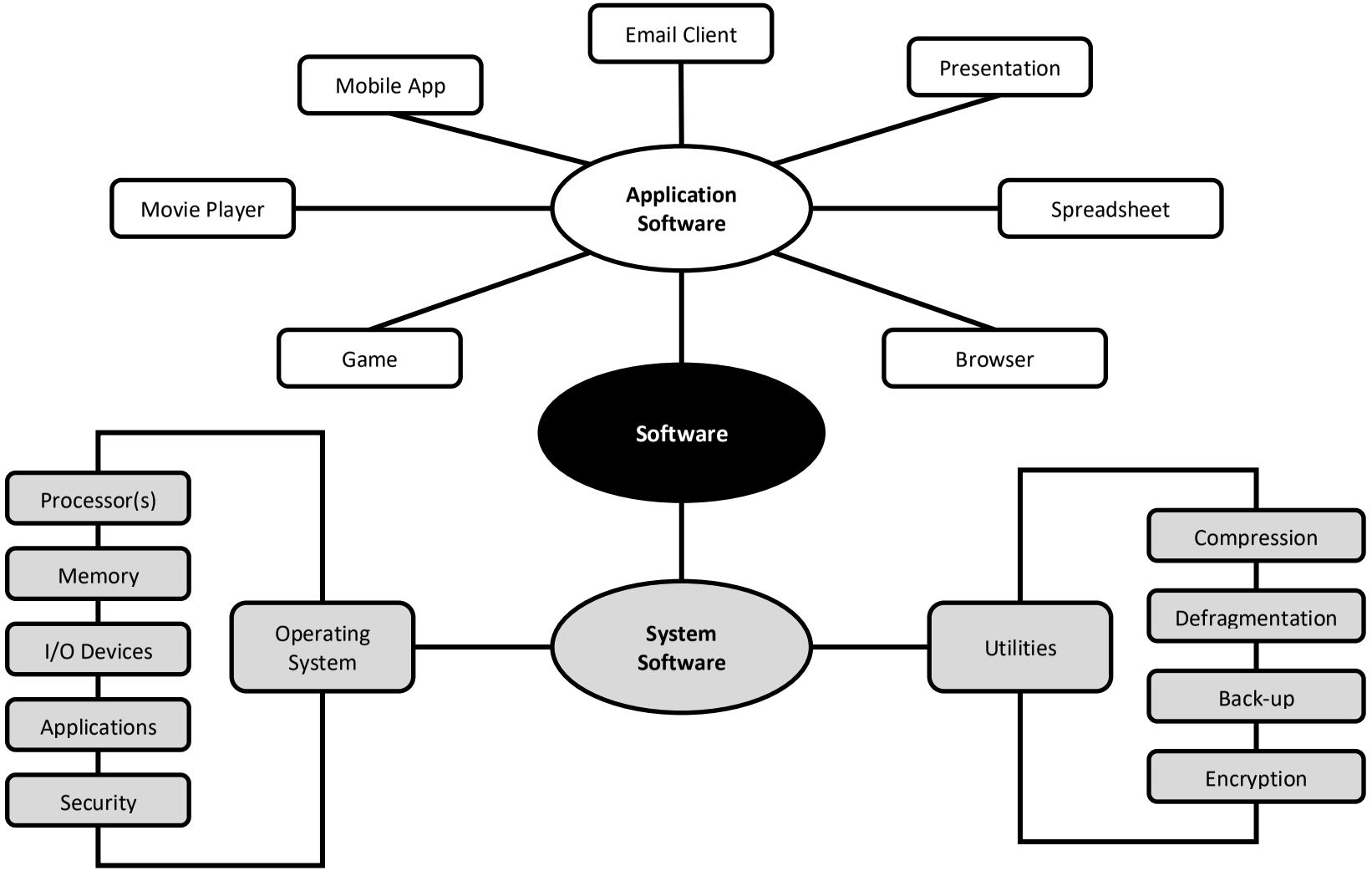
13. Copy the table below and add a description of how individual bits are stored within each of the following storage technologies. [3]

Component	Description
Optical	
Magnetic	
Solid state	

Computer Systems Mind Map (Hardware)



Computer Systems Mind Map (Software)

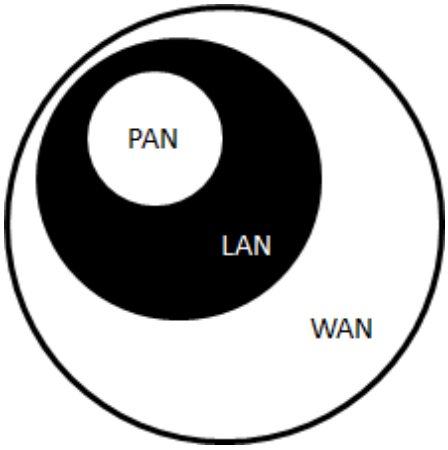


5. Fundamentals of Computer Networks

i **Network** – an interconnection of computers and other pieces of hardware to facilitate communication and the sharing of resources

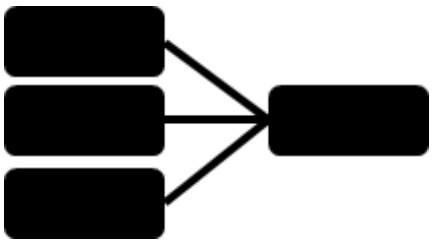
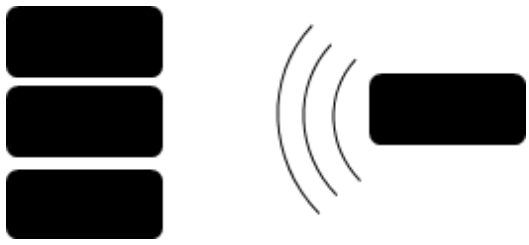
+ Resources such as printers, Internet connections and files can be shared, saving both money and effort
+ Communication can take place via email or instant messaging
+ Backing up data to a different computer is far more straightforward
- Management of a network is a specialist skill, which can be quite expensive
- Security procedures are needed to prevent unauthorised access to a potentially large number of computers
- Network hardware, including cables and switches, can be expensive

Types of Network



PAN – Personal Area Network. This describes devices owned by a single user, connected via Bluetooth, such as a phone, tablet and headset.
LAN – Local Area Network. This is a network covering a single building or perhaps a campus. Usually, a single organisation owns all hardware that belongs to a LAN.
WAN – Wide Area Network. This would be spread over a large geographic area, with shared ownership of some hardware (a telecommunications company might own much of the cabling). The Internet is the largest example of a WAN.

Networks can be either **wired** or **wireless**:

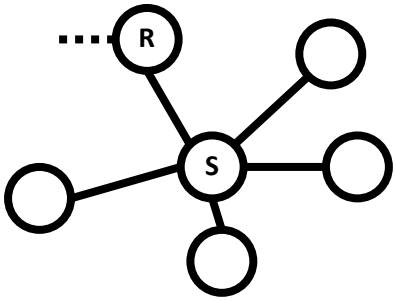
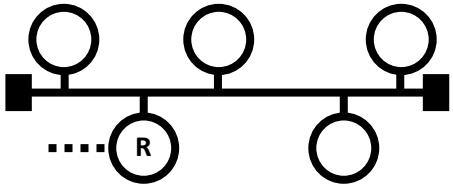
Wired	Wireless
	
+ More secure, as a physical connection is needed, so a hacker would need to be in the building + Less prone to interference	+ Much easier to add a device + No requirement to stay in the same place

Connection Media

Wireless	**Radio waves**
Wired	**Optical fibre** can be used for very fast connections, or if lots of computers are going to share a single connection **Copper** cabling is cheaper and slower, although it is quick enough for most activities that a single computer would perform online

Topologies

i **Network topology** – the pattern in which the hardware on a network is positioned, including connections. Common topologies include star and bus.

Topology	Explanation
 **R = router (connection to Internet)** **S = switch**	**Star** Every device is connected to a switch at the centre of the network. All communication travels via this switch. + Very few data collisions, since each device has a dedicated line to the switch + Strong, centralised security. - Lots of cabling needed - If the switch has no spare ports, adding another device can be difficult.
 	**Bus** A central cable, called the **backbone**, running between two **terminators** (black rectangles), connects all devices. + Uses relatively little cable, making it quite inexpensive + Additional devices can be easily added. - Collisions can occur, as multiple transmissions will attempt to use the shared backbone - If a large number of devices are connected, network transmission can be slow (due to collisions).

Network Protocols

i **Protocol** – a set of rules that governs how a computer communicates on a network. Computers have many protocols, each necessary for a different purpose (email, accessing web pages, moving files, etc.). Without protocols, communication between computers would be impossible.

- **Ethernet** – (a family of protocols rather than a single protocol). This is a set of rules that governs how data is formatted for transmission across a local area network.
- **Wi-Fi** – these letters actually aren't short for anything but are a brand name. These rules control how data is transmitted on a wireless local area network (WLAN).
- **TCP/IP** – These are two protocols that often work together – *Transfer Control Protocol* and *Internet Protocol*. Their collective role is to break up data into **packets**, each of which is a chunk of data that knows where it has been sent from and where it is to be delivered to.

- **UDP** – *User Datagram Protocol*. This protocol transmits data packets very quickly, but without checking to see whether each packet has arrived, so it is not always reliable.
- **HTTP** – *Hypertext Transfer Protocol*. This is the set of rules governing how hypertext (the language of the worldwide web) is moved around the Internet, from device to device.
- **HTTPS** – *HTTP Secure*. This protocol encrypts data that is sent across the Internet. Encrypted data cannot be read if intercepted, so is favoured when sending passwords or credit card numbers.
- **FTP** – *File Transfer Protocol*. This is how files are moved from one computer to another across the Internet. This protocol is heavily relied upon in building websites, moving files from the developer's computer to a server, from where they can be accessed publicly.
- **SMTP** – *Simple Mail Transfer Protocol*. While POP3 might be used to retrieve emails, SMTP is used to send them from one server to another.
- **IMAP** - *Internet Message Access Protocol*. This email protocol is used to allow multiple devices (laptops, tablets, phones, etc.) to access the same email account.

Network Security

i **Internet Protocol Address** – a unique number, used to identify every device connected to the Internet. No two IP addresses are the same. If your computer requests a web page, your computer's IP address specifies where that web page should ultimately be delivered.

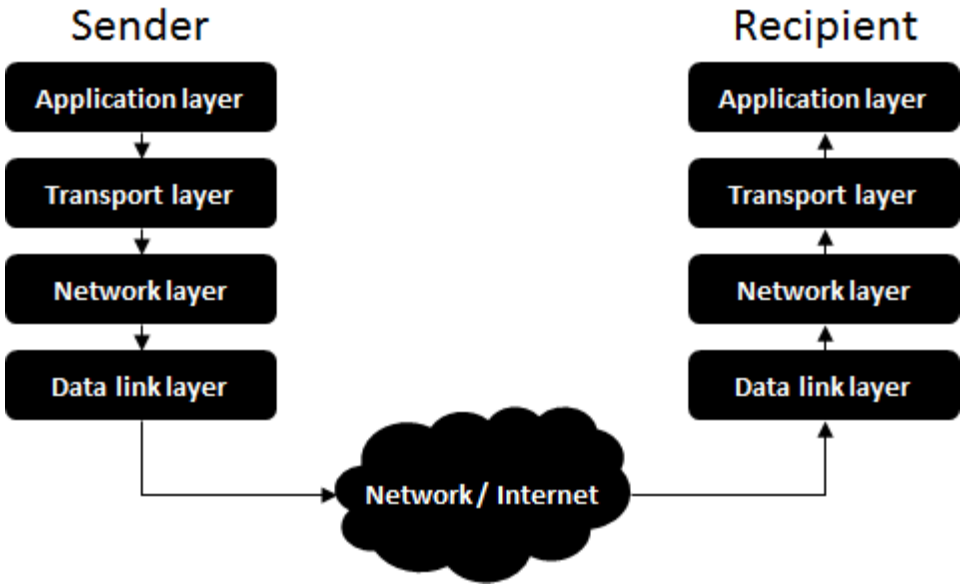
With more data being shared, more data is falling into the wrong hands. Organisations now hire dedicated professionals to keep data secure, as data breaches can damage an organisation's profitability and reputation. Even an individual can lose out financially if their credit card details are stolen.

Authentication	Measures to make sure that a person trying to access data is who they say they are. This can take place by way of: • Usernames and passwords • Memorable information, such as their mother's maiden name • Checking that they are using a recognised IP address
Encryption	Scrambling data using a key to ensure that it makes no sense to anyone who intercepts it. When it is received, the recipient also has a key, which can be used to decrypt the data, returning it to its readable form.
Firewalls	These can be either hardware, software or both. A firewall can be told to block certain traffic (such as all emails or any traffic from a suspect IP address) or to only allow certain traffic (such as from a single, trusted device), blocking everything else.
MAC address filtering	Each computer has a MAC (media access control) address which, unlike an IP address, cannot be changed. Based on this unique identifier, specific devices can be either permitted onto, or blocked from, a network.

The TCP/IP Model

i **TCP/IP model** – a series of protocols. When they work together, they transmit data from one computer, through any number of pieces of network hardware, to another computer. The TCP/IP model is a *concept*, not a physical thing.

The model has four layers, each containing a number of protocols. When data is sent, protocols at one layer break down and re-package the data into smaller units, before passing the data to the layer below. When data is received, those units are reassembled as they move up the model.



Layer	Description	Protocols at this Layer
Application Layer	Where network applications, including browsers and email applications, operate.	HTTP, HTTPS, FTP, SMTP, IMAP
Transport Layer	Establishes communication between the sender and the recipient, agreeing on how communication will take place.	TCP, UDP
Network layer	Packages data for transmission by breaking it into units called **packets**, which are sent across the network.	IP
Network Interface Layer	The physical components of the computer, such as the network interface card, operate at this level.	Ethernet, Wi-Fi

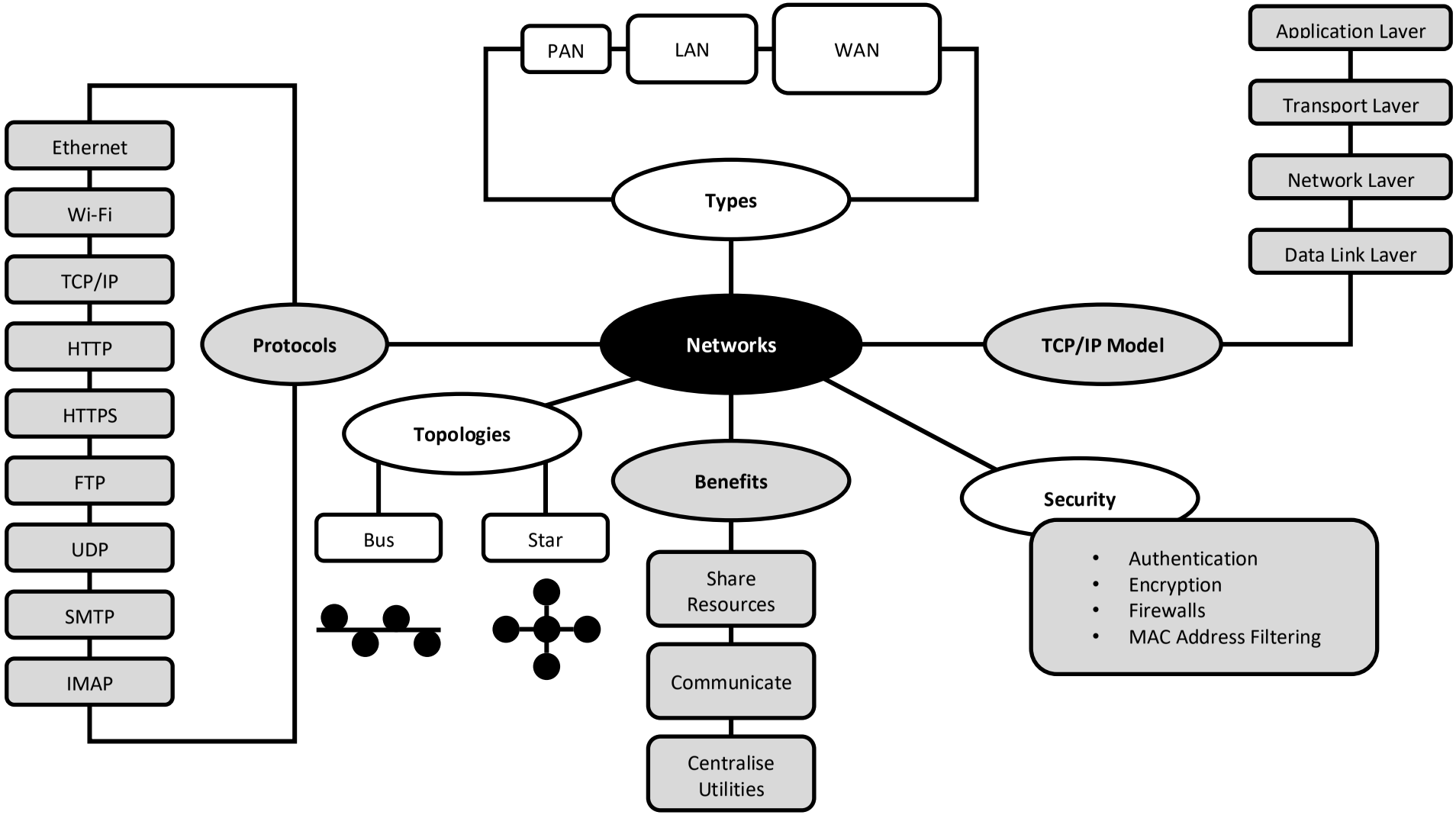
Sample Examination Questions

14. a) Define the term **protocol**. [1]

    b) Give the names of **two** different protocols whose primary purpose is handling emails in some way. Describe the difference between these protocols. [4]

15. Describe **three** advantages of a networked computer over a standalone computer. [3]

Fundamentals of Computer Networks Mind Map



6. Fundamentals of Cybersecurity

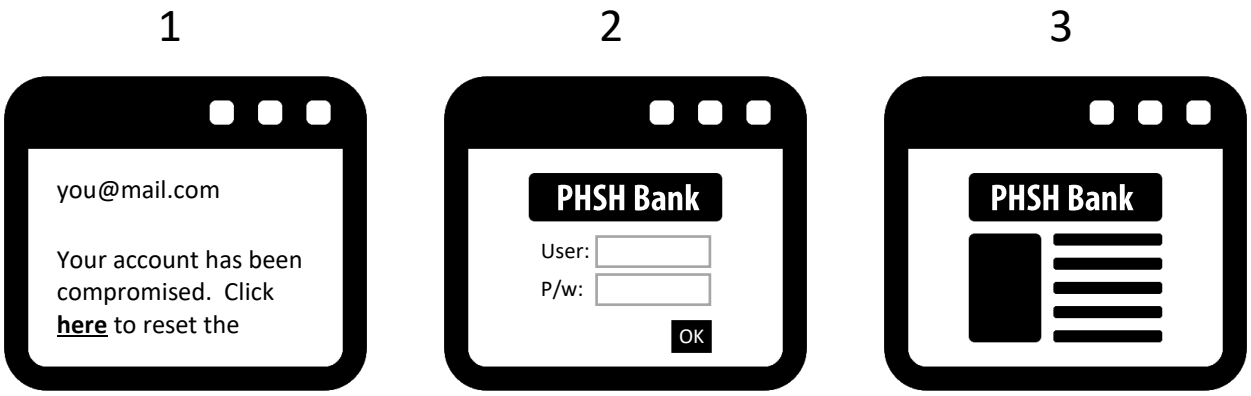
i **Cybersecurity** – a series of processes, practices and technologies that protect networks, computers, software and data from damage, loss and unauthorised access.

Cybersecurity Threats

Social Engineering

i **Social engineering** – forms of cyberattack that focus on people, rather than pieces of technology, as the weak point in any system. There are different ways to manipulate people to surrender confidential information.

- **Pretexting** or **blagging** involves fabricating a scenario in order to gain unauthorised access to a system. A person might pretend to be from IT support in order to persuade an employee of a company to divulge their password
- **Shoulder surfing** or **shouldering** is simply watching, over someone's shoulder, as they type in their password or PIN
- **Pharming** involves redirecting users to an unsafe website. They might type in the web address of, say, their bank, and be taken to a website that looks very much like their bank's website. The fake website will then collect the user's login credentials.
- **Phishing** uses emails to lure people to convincing but fake web pages. They believe they're logging in, but they're really transmitting their login details to an unknown person. Here's how it works:



1. The victim receives an email with a hyperlink. The email tells the user they need to click on the hyperlink, often saying that their security has been compromised in order to motivate them.
2. They will be taken to a screen that asks them to enter personal information. This screen will usually look identical to a screen with which they are familiar.
3. When they have entered the information, they are usually forwarded to the genuine page. In the meantime, the information they entered has been forwarded to a hacker.

Malicious Code

i **Malware** – any program that works against the interests of you or your computer. Viruses, Trojans, adware and spyware are types of malware, although there are others.

- **Computer viruses** are self-replicating pieces of code that can damage data or software. They are often spread via email attachments or removable media such as USB flash drives
- **Trojans** or **Trojan horses** are legitimate programs developed with the intention of hiding malicious code within. Since they are largely legitimate, they are often not recognised as malware
- **Adware** downloads unwanted Internet adverts, often observing your online behaviour in order to target specific adverts
- **Spyware** covertly obtains sensitive data, such as credit card numbers and passwords, transmitting the data to a hacker across the Internet.

Other threats

i **Weak passwords** – passwords that are easy to guess. $tR0nG p@S$worDs contain combinations of upper-case and lower-case letters as well as numbers and symbols. Similarly, **default passwords** can be a problem. A router's new owner might not change the password from 'admin' or 'password' when they buy it, leaving their network vulnerable.

i **Misconfigured access rights** – access rights are rules that tell a computer system which user should have access to which files and other resources. If these access rights are not set up properly, employees and other users could access data that they should not be able to access.

i **Removable media** – any storage device that is highly portable can easily be used to steal data or introduce malware onto a system.

i **Unpatched software** – when a security risk is identified in a program, the developer will release a **patch**, which is an add-on program that fixes the security risk. If a user does not install the patch, their computer is not secure.

Threats will often exist in combination. A username acquired via phishing could be used in collaboration with a weak, easy-to-guess password to introduce a virus that specifically seeks out unpatched software.

Detecting and Preventing Cyberattack

Biometric measures	Using some part of a person's biology to access a system instead of a password. For example: • Mobile phones and tablets that unlock on scanning a fingerprint • Doors that unlock when a person's iris or retina is scanned • Voice recognition • Face recognition
Password systems	Automated procedures that ensure that sound password policies are followed: • $tR0ng p@s$worD$ that include many different character types • Passwords that must be changed on a regular basis Users that try not to adhere to the policies are simply not allowed into the system until they do.
CAPTCHA	Blurry text is presented to the reader, which is easy for a human to read but difficult for a computer. This technology is used to ensure that a human is using the system, not simply a computer program trying to guess a password at a rate of millions of attempts per second.
Email confirmation	Often, when a password is changed, a user must verify this change by clicking on a link sent to a registered email address. This can prevent third parties changing passwords unnoticed.
Automatic software updates	When a new version of software, which might have updated security measures, is released, a computer can be configured to automatically download this new version.
Penetration testing	Someone tries to hack into a system, but as an employee or contractor of the system's owner. Their aim is not to steal or corrupt data, but to identify weaknesses so that they can be resolved.

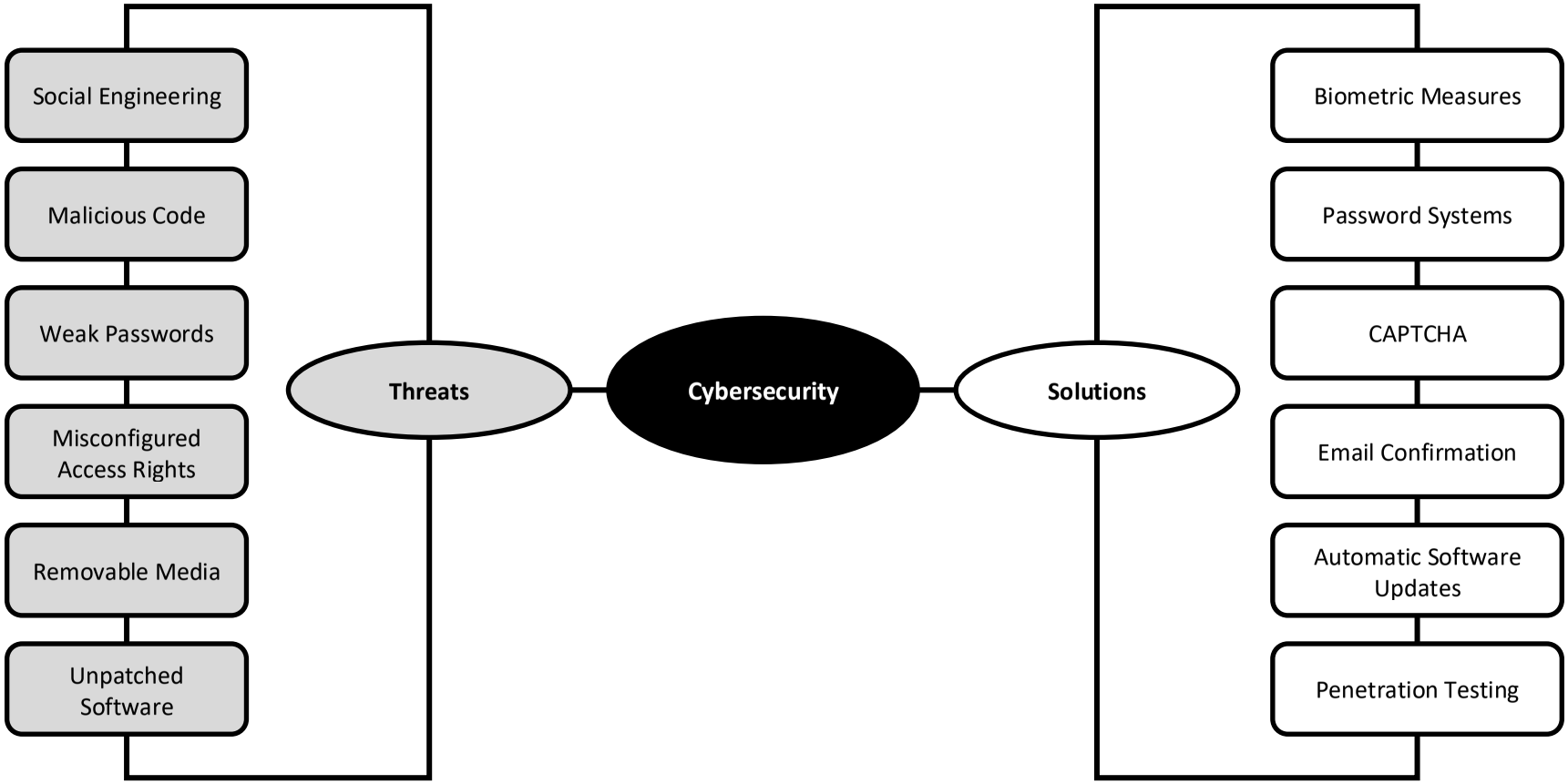
Sample Examination Questions

16. The NHS stores a large amount of confidential data about patients. This data needs to be available at healthcare facilities across the country, so it is available online. Security is in place, however, and staff need to enter a password in order to view any of the data.

    State three cybersecurity threats that could apply to this data, and describe a different security measure to counter each threat. [6]

17. Define the term **social engineering**. State **two** activities associated with social engineering. [3]

Fundamentals of Cybersecurity Mind Map



7. Ethical, Legal and Environmental Impacts

Ethical Issues

i **Ethics** – this term refers to what is right and what is wrong, although ethical issues are rarely so straightforward that they have a single 'right' answer. Often what is right for the individual is wrong for society as a whole, and vice-versa.

Topics	Issues
Privacy	• People expect to go online and find out everything about other people (freedom of expression), but they might prefer to keep some element of secrecy about their own lives (expectation of privacy), especially from the government. The government, on the other hand, argues that it needs access to personal data to guard against terrorist threats • Once something is shared online, particularly in the world of social media, it is next to impossible to delete it completely • Through social media, people unwittingly share details of their purchases, the schools attended by their children, where they go in the evening, etc.
Inclusion	• Among some groups, especially those of school age, there can be a social stigma around not having the latest technology • Not everyone in the country has access to the Internet, so not everyone has equal access to information, to job listings or to a society that increasingly exists online • Government could commit money to solving this inequality, but in doing so, they would be spending money unequally. This is an instance of there being no 'right' answer.
Professionalism	• There is increasingly an expectation of people to be available, by email, outside of working hours; this is a direct impact of technology • Although you can apply for jobs internationally, employers can recruit internationally, making the process far more competitive • Social media can be seen by anyone, including prospective employers, blurring the line between private and professional life.
Artificial Intelligence	• Driverless cars are now a real possibility, but it might be unclear whether the car's occupant or its programmer would be to blame in the event of a crash • Computers can read CVs to filter out certain types of job candidate, but the 'type' might be people from particular postcode areas or ethnic backgrounds if an unscrupulous developer decides upon this

Environmental Issues

i **Environment** – a broad term with several meanings relating to the physical world in which we live. Global air pollution is an environmental issue, but so is the distance between where you live and where you work. The following table provides categorised examples:

Health Issues	
+ Proliferation of health-tracking apps allow people to monitor exercise and calorie intake – people are better informed + Medical technology is continually advancing in predicting and diagnosing illness + Sharing of health-related data across the Internet helps research.	- An increase in computer-based jobs is met by an increase in sedentary lifestyles - Some people are addicted to games and social media, and they can become withdrawn - Technology means people are never truly away from work, increasing stress levels.
Energy Use	
+ Computer technology can be used to reduce consumption of fossil fuels; they can turn off lights in empty offices and cause cars to run more fuel economically + Smart meters allow people to track and control their use of electricity and gas at home and at work + People can work from home more, so they commute less.	- Computers, tablets, smartphones, e-readers, etc. all consume electricity - The manufacture of such devices also requires electricity - Many devices that consume electricity are unseen, including vast server farms for hosting cloud storage and online gaming.
Resources	
+ In theory at least, less paper needs to be used, so fewer trees should be cut down + Some products, such as books and music, can be delivered electronically, with no physical transport needed + One delivery driver, delivering ten Internet-ordered products on a single delivery run, requires less fuel than ten people each driving to a shop for one item.	- In reality, people often print unnecessarily, so it may not really be true that less paper is used - Computers require resources for their production, such as gold, which are not in infinite supply and have to be mined - Not all obsolete technology is recycled, and much of it ends its life, irretrievably, in landfill.

These bullet points are just the beginnings of arguments. Be prepared to weigh two conflicting ideas against each other to see which one carries the most weight.

Legal Issues

Copyright, Designs and Patents Act (1988)

This law protects **intellectual property**, meaning it is a criminal offence to copy certain things without the permission of the owner of the **intellectual property rights**. Different types of protection exist:

- Copyright applies to anything that can be written (such as web pages, books, music and program code) as well as images. Once something has been written, copyright exists immediately, without the need to apply for it.

> i **Creative Commons license** – a copyright holder may issue a creative commons license to allow others to use, build on or distribute their copyrighted work without fear of being prosecuted. Sometimes, such a license may include limitations, such as permitting free distribution, but not sale, of the copyrighted work.

- A registered design (applicable to computer science) would apply to logos, icons or similar. As the name suggests, such images need to be registered, and they need to have been unique prior to being registered
- Patents can be used to protect inventions. This would apply to a piece of hardware or a process (such as a new method of printing), but not to program code.

Computer Misuse Act (1990)

This law makes hacking a criminal offence. The following activities are recognised in this particular law:

- Accessing material on a computer that you are not authorised to access (for example, logging into a system using someone else's credentials)
- Modifying material on a computer that you are not authorised to modify. You could break this law even if you are allowed to access the data

Data Protection Act (1998)

This law applies to personal data of living individuals. If an organisation stores personal data of living individuals, that data must be…

- processed fairly and lawfully
- held for specified purposes
- adequate, relevant and not excessive
- kept up-to-date
- not kept for longer than necessary
- processed according to the rights of the individual data subject
- kept securely
- not transferred outside the EU unless to a country with a similar law to this one.

Freedom of Information Act (2000)

This law gives members of the public the right to access information under the following circumstances:

- The information must relate to either a public body (local council, government department, police force) or an organisation that provides a service to a public body
- Providing the data would not cause another law (probably the Data Protection Act) to be breached

Exam questions on this area may require you to draw together an understanding of all three of **ethical**, **legal** and **environmental** impacts in a single answer. While you are expected to have an understanding of the areas mentioned in the table below, you are also expected to know when a particular issue is relevant. You might be asked about, say, wireless networking, and you would be expected to recognise that unauthorised access and invasion of privacy are relevant issues. Again, these bullet points are starting points – not whole answers.

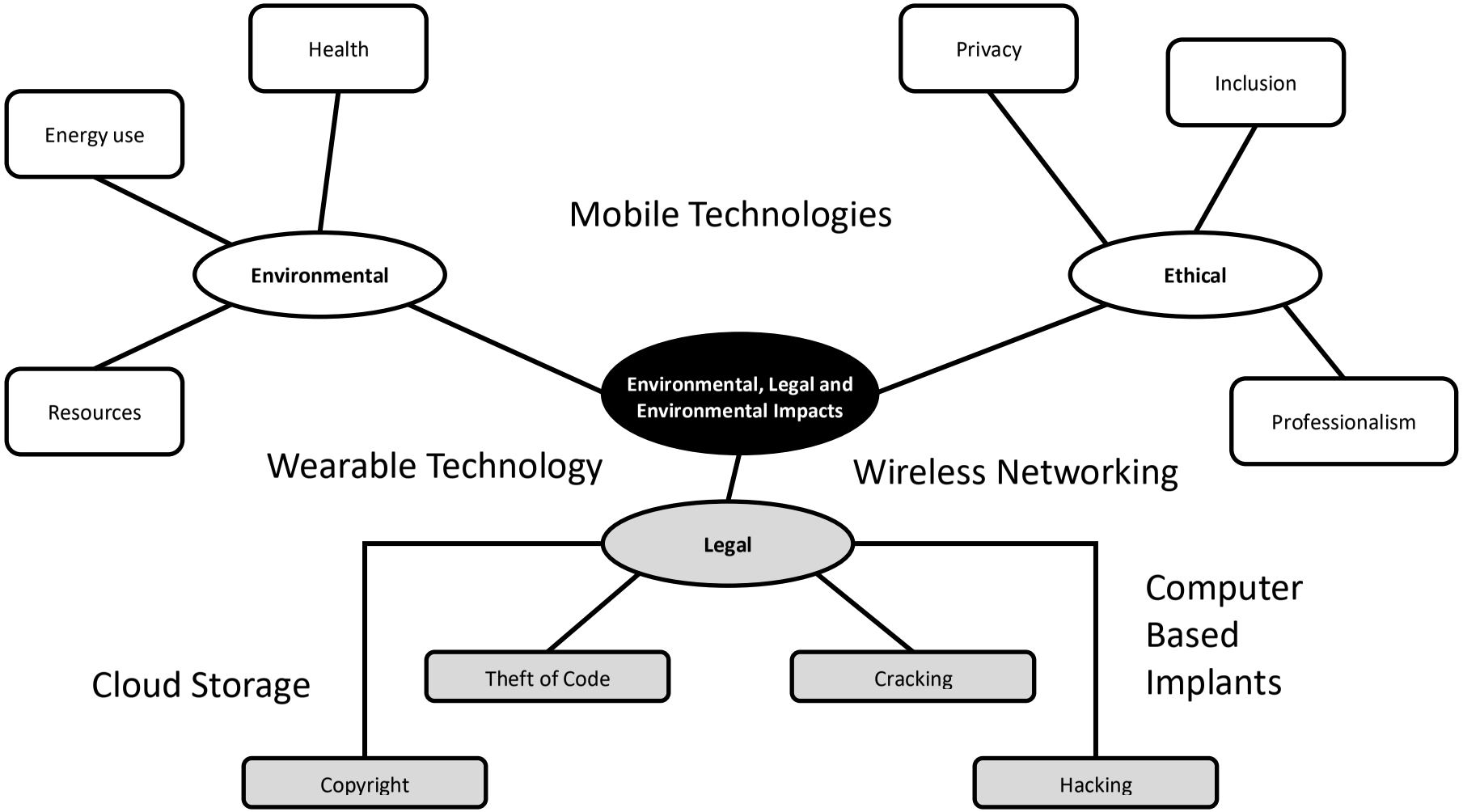
Cybersecurity	• A government cybersecurity policy may actually weaken your own privacy if it grants the government access to personal data • A company may be held liable if it does not protect data from unauthorised access; it can be sued for a data breach • The issue of cybersecurity overlaps with many others in this table, including mobile technologies, wireless networking and cloud storage
Mobile technologies	• The amount that your phone knows about you, in terms of likes, locations, friends, etc. is already a breach of privacy • Constant replacement of mobile phones, and their batteries, has a negative impact on the environment • Illegal copies of MP3s are commonly found on people's phones
Wireless networking	• Easier access for unauthorised users than in a wired network • Wireless network is better in terms of the environment as far as production of cables and network hardware are concerned • Privacy can be breached as others might access personal data
Cloud storage	• Other people may gain access to your data when stored remotely • Providers must adhere to the Data Protection Act • Fewer, more centralised data stores are better for the environment as they reduce transport costs associated with distribution
Theft of computer code / copyright of algorithms	• Theft of computer code breaches the Computer Misuse Act and the Copyright, Designs and Patents Act • Stealing code, as well as being illegal, is ethically unsound, as it takes unfair advantage of the work of others • One company could gain an unfair advantage over another by stealing code that could have taken months or years to write. Unlike physical products, a program can be assembled instantly using stolen code.
Cracking	• Cracking involves gaining access to part of a program that you do not have access to, usually to bypass payment • This is a breach of the Computer Misuse Act • It is ethically unsound, as it takes advantage of the work of others, and increases costs for people who *are* willing to pay
Hacking	• Unauthorised access potentially breaches people's privacy • It also breaches the Computer Misuse Act and probably the Data Protection Act • Hacking often results in additional crimes, such as stalking or breach of copyright
Wearable technologies / Computer-based implants	• Wearable technologies carry the same ethical, legal and environmental impacts as mobile technologies (see above) • Implants carry the additional complication of replacement; it is not as straightforward as simply buying a new one • Looking to the future, an implant may be a potential major breach of privacy, as it could contain biometric data, information about illnesses, etc., and hacking such a device could be dangerous

Sample Examination Questions

18. In recent years, there has been a dramatic increase in the number of people who own a smartphone. Discuss the benefits and drawbacks of the widespread use of mobile phone technology.

    In your answer, you should consider any legal, ethical and environmental issues relating to the widespread use of mobile phone technology. [9]

Ethical, Legal and Environmental Impacts Mind Map



Sample Answers

Fundamentals of Algorithms

1. An algorithm is a series of instructions that describes how to solve a specific problem.

i You've seen by this point, this guide is full of definitions. It is highly likely that some of these definitions will be required in the exam, although it's impossible to determine which ones. Definitions, assuming you have revised well, are the easiest marks available. One way to study definitions is to produce flashcards, with the word on one side and its definition on the other. Making the cards is beneficial on its own, and it's easy for others to help you to study by testing you on the definitions.

2. Terminator
   Process
   Decision

3. Pairs of numbers are compared (1st with 2nd, 2nd with 3rd, 3rd with 4th, etc.), with values being switched if they are not in order. Once all pairs have been compared, that constitutes one pass. More passes will occur until either n-1 passes have occurred, or an entire pass occurs without any values being switched.

i This explanation is fairly complex. In circumstances like this, you might find it helpful to supplement your written answer with a diagram. While it probably wouldn't be worth any marks on its own, it might help to clarify any points that you haven't worded as well as you would have liked.

Programming

4. a) i) total
      ii) {keeps a running total}
      iii) +
      iv) >
      v) while
   b) They would need to input a value of '-1'.

i The best way to prepare for questions like this one is to write code – lots of code. Whenever you write a program, you should make an effort to understand every part of every line you write. Don't be happy with simply having a program that works; make sure you can understand *why* it works. Ask your teacher about any lines of code that you don't understand.

5.

Data	Integer	Real	Boolean	String	Char
Money in account		X			
Account holder's name				X	
Number of whole years the account has been active	X				
Account holder's postcode				X	
Whether or not an overdraft is permitted			X		
A single-letter code that identifies the account type					X

6.
```
highCount as integer = 0
  for loopCount = 0 to 9
  if DATA[loopCount] > 5
       highCount = highCount + 1
  end if
end for
```


Fundamentals of Data Representation

7. a) 128 + 16 + 8 + 4 + 1 = **157**
   b) 1001 = 9; 1101 = D; **9D**

> *i* You will probably be asked to convert between binary, decimal and hexadecimal numbering systems. At the end of the exam, if you have time, a good way to check your answers is to perform the conversion in reverse. Did you convert a binary number to hexadecimal? Convert it back to binary and check that the result of this conversion matches the question.

8. a) 01100000
   b) Number is multiplied by four.

9. a) A collection of every possible letter, number, symbol, etc. available to a computer.
   b) 77

10. a) i) The number of different colours possible within a particular image.
       ii) 256
    b) 640 * 480 * 8 = 2,457,600 bits
       2,457,600 / 8 = 307,200 bytes
       307,200 / 1000 = **307.2 kilobytes**

> *i* When you try exam questions that require calculations, make sure you show a result at every stage, no matter how trivial (in the above example, we multiplied by 8, then divided by 8). Not only is it a comprehensive attempt at every single mark, it increases the likelihood of picking up some of the marks, even if your calculations contain an error.

Computer Systems

11. a) AND
    b)

Temp	Time	Output
0	0	0
0	1	0
1	0	0
1	1	1

12. An embedded system is a computer that forms part of a larger electrical or mechanical system. Inputs are received via pins that go into the microprocessor, and outputs are sent via different pins. Inputs might be sensors, or buttons; outputs might be actuators or LCD screens.

13.

Component	Description
Optical	'1's and '0's are represented by pits being burned or not burned at predetermined locations on a disc
Magnetic	'1's and '0's are represented by magnetising individual magnetic fragments, which can be either North-aligned or South-aligned
Solid state	Transistors can either hold a charge or not hold a charge, which translates to either a '1' or a '0'

Fundamentals of Computer Networks

14. a) A protocol is a set of rules governing how one device communicates with others.
    b) SMTP – Simple Mail Transfer Protocol. This manages the relaying of email messages from one server to the next, towards their ultimate destination. IMAP – Internet Message Access Protocol. This allows email messages to be retrieved by several devices, each accessing the same email account.

i Always spell out acronyms in full. Writing 'simple mail transfer protocol' as well as 'SMTP' might not gain you a mark, but it may prevent you from losing one.

15. Files can be stored centrally, allowing data to be accessed on multiple workstations
    Files can be backed up more easily when stored centrally, rather than across multiple workstations
    Networks enable security to be established, ensuring that users can only access certain directories/files
    Networks enable shared access to hardware such as printers
    Networks enable shared access to applications

i This was a three-mark question that required three pieces of information. If you can easily identify where each mark is coming from, you should address each mark separately. Since the examiner will be looking for three distinct points, three distinct sentences are the best way to present your answer.

As you've seen by this point, this guide is full of definitions. It is highly likely that some of these definitions will be required in the exam, although it's impossible to determine which ones. Definitions, assuming you have revised well, are the easiest marks available. One way to study definitions is to produce flashcards, with the word on one side and its definition on the other. Making the cards is beneficial on its own, and it's easy for others to help you to study by testing you on the definitions.

Fundamentals of Cybersecurity

16. People can read staff members' passwords over their shoulders. This can be countered by complex passwords, using a range of different character types, which are more difficult to memorise than words.

    Data can be intercepted as it is transmitted between devices. Encryption can overcome this threat, scrambling data so that it only makes sense when accessed on the intended recipient.

    Malware might be installed on a system, which could steal or corrupt patient data. Ensuring that anti-malware software is installed and kept up-to-date is a means of countering this threat.

i Again, the origin of each mark is easy to identify. Each paragraph is an attempt at two marks, so each paragraph needs to make two distinct points. Within each paragraph, the first sentence describes the problem (first mark) and the second sentence describes the solution (second mark).

17. Social engineering is the act of manipulating people in order to get confidential information from them.
    One example of this is phishing.
    Another example is pretexting.

i The keyword 'state' means only minimal information is required. In 'state' questions, a single word or phrase will be enough to get you the mark.

Ethical, Legal and Environmental Impacts

18. The main benefit is that many more people are now able to communicate with a much larger world. Through social media, communication is possible between countries, and people have access to a wealth of information via the Internet wherever they go. There are also benefits to safety – it is easier than it used to be for a person who is in danger to seek help. While some may question the pace at which people acquire new and improved mobile phones, whenever someone gets a new phone, someone else is likely to acquire their old phone from them. This increases the number of people who have access to the communication and safety benefits.

    However, there are some negative points. Mobile phones make it much easier for people to put their own privacy at risk, since photos and personal information shared on social media can be impossible to completely remove from the Internet. There is also an ethical issue of equal access to technology – the more advanced mobile phones become, the more people who do not have access to them are left behind. The environment also takes a hit, since the materials needed to construct mobile phones, and to dispose of them when they are no longer needed, require mining.

i There will probably be one question like this, which will be worth a large number of marks, some of which depend on good-quality written communication. The question is most likely to be on ethical, legal and environmental issues.

There are some guidelines you can follow for such questions:

- Plan your answer, rather than jumping straight in. The model answer above contains separate paragraphs for each side of the discussion, and each paragraph makes several points.
- Know your subject – it is unlikely that you will score highly on a topic you have not studied, however well written your answer is.
- Read/watch the news – this question is likely to be quite current, and being aware of the latest developments might give you more to write about.
- Provide examples to support any points you make (examples above include motion sensors and satellite navigation).
- Proofread your finished answer. A single spelling error (or even a hurriedly scribbled word that cannot clearly be made out) could cost a mark, and that mark could put you on the wrong side of a grade boundary.

