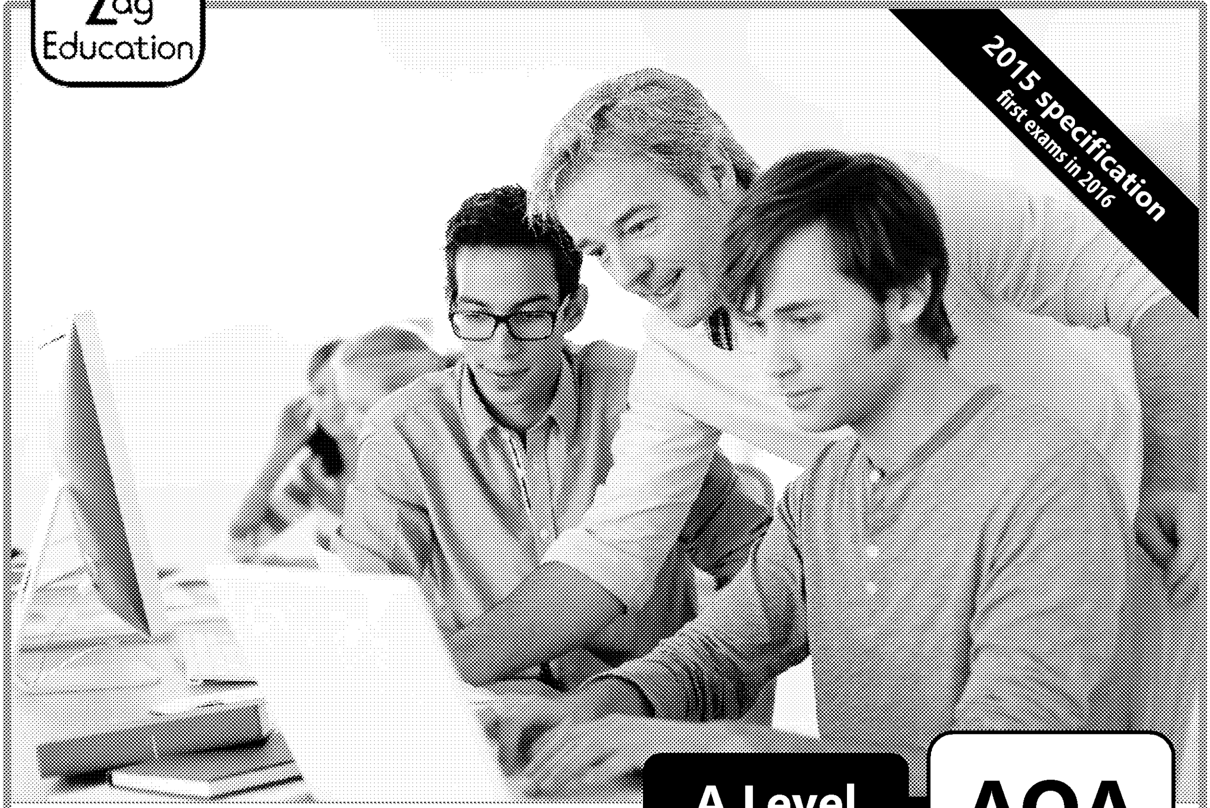
Computer Science A Level | AQA | 7517



A Level AQA

Revision Guide

for A Level AQA Computer Science

Paper 1 –Topics 1-4

zigzageducation.co.uk POD 7759

Contents

TEACHER'S INTRODUCTION

This revision guide has been written to support the AQA A Level Computer Science specification (first teaching from September 2015, first exams in June 2017).

It covers the essential theory required for the A Level Paper 1 examination, including:

1. Fundamentals of programming
2. Fundamentals of data structures
3. Fundamentals of algorithms
4. Theory of computation

Note that part of the Paper 1 examination is based on pre-release material (including skeleton programming code) that is released annually by AQA. For details of resources supporting this pre-release, see the ZigZag Education website.

Each section includes student notes, examples, diagrams and examination-style questions. Example answers are included at back of the resource. *Note that credit should also be given for any valid responses not explicitly included in this resource.* There is also a revision progress grid which students may find useful in the lead up to their exams.

Programming concepts are exemplified throughout using pseudocode and a number of high-level programming languages including Java, C++, Visual Basic and Pascal.

November 2017

REVISION PROGRESS TRACKER: A LEV

Use the grid below to track your progress while revising for your exam. Start by enter the top, and working down the grid, give a rating of between 1 (you really don't know i

This should help you to focus your revision on the areas that require it the most, so tha comes up in the exam. Use the Notes column to record any actions.

Repeat this process until you feel you are confident enough in all areas and are ready f

Specification Topic	Confidence Level (1 – 5)				
	Date:	Date:	Date:	Date:	
1 – Fundamentals of programing					
Data types					
User-defined data types					
Built-in data types					
Assignments					
Iteration					
Selection					
Arithmetic operations					
Relational operations					
Boolean operations					
Variables and constants					
String handling					
Random numbers					
Exception handling					
Subroutines					
Parameters and return values					
Local and global variables					
Stack frames in subroutines					

Specification Topic	Confidence Level (1–5)				
	Date:	Date:	Date:	Date:	
Recursive techniques					
Procedural-oriented programming					
Object-oriented programming					
2 – Fundamentals of data structures					
Data structures					
Arrays					
Text files					
Binary files					
Abstract data structures					
Queues					
Stacks					
Graphs					
Trees					
Hash tables					
Dictionaries					
Vectors					
3 – Fundamentals of algorithms					
Graph-traversal					
Tree-traversal					
Reverse polish					
Linear search					
Binary search					
Binary tree search					
Bubble sort					
Merge sort					
Dijkstra’s shortest path algorithm					

Specification Topic	Confidence Level (1 – 5)				
	Date:	Date:	Date:	Date:	
4 – Theory of computation					
Problem-solving					
Following and writing algorithms					
Abstraction					
Information hiding					
Procedural abstraction					
Functional abstraction					
Data abstraction					
Problem abstraction/reduction					
Decomposition					
Composition					
Automation					
Finite state machines					
State transition diagrams and tables					
Maths for regular expressions					
Regular expressions					
Regular language					
BNF / Syntax diagrams					
Comparing algorithms					
Maths for understanding Big-O notation					
Order of complexity					
Limits of computation					
Classification of algorithmic problems					
Computable and non-computable problems					
Halting problem					
Turing machine					
Universal Turing machines					

TOPIC 1 – PROGRAMMING

1.1 PROGRAMMING

DATA TYPES

Data types

ⓘ **Data types** describe the type of data a variable contains in a computer program.

Type	Description
Integer	An integer is a whole number that can be positive or negative.
Real/Float	A real number contains a decimal point; the position of the decimal point
Boolean	Boolean data types represent two logical states: 'true' (typically 1) or 'fals
Character	A character represents a single alphanumeric item of data, such as a numb
String	A string contains one or more characters, plain text such as: 'Hello World'.
Date/Time	The Date/Time data type contains details of an instant in time, not refere
Pointer/ Reference	A pointer is a data object that contains the memory location of another va Note: Not all languages use pointers or references.
Records / or equivalent	Record data structures are made up of a list of elements where each recor fields with different data types – used in Pascal and Delphi programming Visual Basic uses an equivalent of records, based on user-defined data typ records and C++ and C# use Structs as an equivalent of records.
Arrays	Array data structures are made up of a list of date elements that are the s Arrays can be one-dimensional (similar to a list) or two-dimensional (the e After the array has been declared, the elements in an array can be initialis the program.

Built-in data type

ⓘ A **built-in data type** is one where the programming language offers built-in support. Typical built-in data types for commonly used programs are listed below.

Built-in Types	Visual Basic	Pascal	Java	
Integer	Integer (4 bytes)	Integer (4 bytes)	int (4 bytes)	int (4
Byte	Byte (8 bits)	Byte (8 bits)	byte (8 bits)	byte (
Boolean	Boolean (1 byte)	Boolean (1 byte)	boolean (1 byte)	bool (
Real	Double (8 bytes) Decimal (16 bytes)	Real (8 bytes) Currency (8 bytes)	float (4 bytes) double (8 bytes)	doub decim
Character	Char (2 byte)	Char (1 byte)	char (2 bytes)	char (
Strings (*) as required	String (*)	String (*)	String (*)	string

1.1 – Progress Check

1. Describe with examples the following data types:
   (a) Real/Float (2 marks) (b) Character (2 marks) (c) Array (2

User-defined data types

ⓘ **User-defined data types** can be declared by programmers to meet their specific requi

Data structure declaration examples show a record in Pascal and an equivalent approach (s

Record Example Data Structure in Pascal

```
TYPE
     StudentRecord = RECORD
        StudentName    :STRING(20);
        MobileNumber   :INTEGER;
        EntryYear      :INTEGER;
        FeesOwing      :REAL;
     END;
VAR
        Student   : Array[0..19] of
        StudentRecord;
```

Struct Example Data Str

```
struct  StudentReco
{
     string   Stude
     int      Mobil
     int      Entry
     float    FeesO
 StudentRecord;
}
struct  StudentReco
```

PROGRAMMING CONCEPTS

Programming concepts – variable declarations

ⓘ **Variables** are used in programs to store data that may change when the program is execu require that the data types of the variables in the program are declared before they can b

C#	Java	Pascal

`bool blogic;`
`int intVal;`
`char chVal;`
`double Sum;`
`string stVal;`
	`boolean blogic;`
	`int intVal;`
	`char chVal;`
	`float Sum;`
	`String stVal;`
		`var`
		`blogic : boolean;`
		`intVal : integer;`
		`chVal  : char;`
		`Sum    : real;`
		`stVal  : string;`

In Python, variables are declared when assigned. For example, **Distance = 10.5** will be floa

Programming concepts – constant declarations

ⓘ **Constants** are used where data used in program is preset and does not change. Progra types of the constants in the program are declared and their value initialised before the

C#	Java	Pascal

`const double`
`Pi=3.14;`
`const int x = 12;`
	`static final float Pi=3.14;`
	`static final int x = 12;`
		`Const Pi=3.14;`
		`Const x = 12;`

In Python, constants cannot be declared in, so capitalise the variable and don't change it, e

Programming concepts – assignments

ⓘ An **assignment** is where a value is computed within a program and assigned to a varia

Pseudocode example with data declarations

Area of a Circle Calculation

```
START
      Float Area, Radius, Pi;            // Data Declarations
      Pi ← 3.1416;
      Area ← Pi * (Radius * Radius)      // Area calculation
END
```

In a sequence structure the program performs each action or assignment statement in ord

Programming concepts – subroutines

A **subroutine**, which is identified by name, is a set of instructions that perform a certa called many times within a program.

A **procedure** is a subroutine that is called to perform a task. It may or may not return a

A **function** is a subroutine consisting of a series of instructions to perform a task. Whe returns a value.

Tasks that are repeated within a program are often defined as procedures and functions ar within the program. They consist of a series of statements declared outside of the main pr main program or by other subroutines.

Programming concepts – iteration

Iteration is where a program executes a statement or statements that are repeated w The number of iterations in a loop can be further distinguished into definite and inde

Definite iteration is where the number of iterations that will take place is known before the start of the execution of the main body of the loop.

Indefinite iteration i will take place is no determined by whe

Pseudocode examples below different iteration types available.

FOR LOOP – used when the same instructions / calculations are carried out for a known number of iterations.

The example is based on definite iteration as loop is repeated a known number of times (10).

```
#Example FOR loop
Total ←  0
FOR X = 1 TO 10
       Total ←  T
ENDFOR
```

WHILE LOOP – used when a condition is satisfied at the start of the loop and stops when this is no longer true.

The example is based on definite iteration as the loop will be repeated a known number of times (10).

```
#Example WHILE loo
F ←  1
counter ←  10
WHILE counter > 0
      F ←  F + 1
      counter ←  c
ENDWHILE
```

DO WHILE LOOP - used when a condition is satisfied at the start of the loop and stops when this is no longer true.

The example is based on indefinite iteration as the loop will continue an indefinite amount of times dependent upon the number input by the user.

```
#Example DO WHILE l
F ← 1
OUTPUT "Enter a Num
counter ← INPUT
WHILE counter > 0
       F ← F * co
       OUTPUT F
       counter ← c
ENDWHILE
```

REPEAT UNTIL LOOP - the loop operates at least once and then the condition is tested at the end of loop and repeats until the condition is no longer true.

The example is based on indefinite iteration as the loop will continue an indefinite amount of times until the user inputs 10.

```
#Example REPEAT UN
REPEAT
      OUTPUT "Enter
      N ← INPUT +
      OUTPUT N
UNTIL N = 11
```

Programming concepts – selection

A **selection** structure is where the program executes different actions or statements
simple pseudocode examples show typical selection types available.

IF-THEN statements are used to execute one block of code when a Boolean condition is TRUE, there is no alternative branching when the Boolean condition is FALSE.

```
# IF-THEN Example
IF (X > MAX) THEN
      X ←  MAX
END IF
```

IF-THEN-ELSE statements are used to execute one block of code when a Boolean condition is TRUE and an alternative when the Boolean condition is FALSE.

```
# IF-THEN-ELSE Exa
IF (Age >= 18) THE
      OUTPUT “You
ELSE
      OUTPUT “Not
END IF
```

Nested IF-THEN-ELSE statements are used if there is more than one expression to be tested.

In the example shown there are two IF statements required in a row to assign the output sign of a number.

Once an IF or ELSE IF expression is true the OUTPUT is assigned and the program moves onto the next statement.

```
# Nested IF-THEN-E
IF (Score > 0 ) TH
      OUTPUT “Pos
ELSE IF (Score = 0
      OUTPUT “Zer
ELSE
      OUTPUT “Neg
END IF
```

CASE statements (termed switch statements in programs such as JAVA/C) are a variation of an IF-THEN-ELSE statement where several IFs are used in a row.

The Nested IF-THEN-ELSE approach outlined above operates in a similar way to the CASE statement.

```
# CASE or SWITCH E
CASE Weekdays
     1: THEN Day
     2: THEN Day
     3: THEN Day
     4: THEN Day
     5: THEN Day
END CASE
```

Programming concepts – identifiers

Identifiers are symbolic names used for any variable, function or data definition in a
they are normally given meaningful names to make the program understandable.

Examples:

- StudentExamScore is more understandable than x
- TotalCost is more understandable that t

1.1 – Progress Check

2. Describe the following programming concepts:
   (a) Subroutine (4 marks)
   (b) Procedure (3 marks)
   (c) Function (4 marks)
   (d) Iteration (3 marks)
   (e) Selection structure (3 marks)
3. Explain the difference between definite and indefinite iteration. (4 mar

Arithmetic operations in a programming language

Arithmetic Operation	Operator	Explanation
Addition	+	Numbers are added either side of addition **a + b = 16** where a = 12 and b = 4
Subtraction	-	The number on the right side of the subtra number on the left. **a - b = 8** where a = 12 and b = 4
Multiplication	*	Numbers on either side of the operator are **a * b = 48** where a = 12 and b = 4
Real/Float Division	/	The division operator performs floating po operands is a floating-point value, a floati example below. **a / b = 7/2 = 3.5**
		In Python, floating point division is perfor **numbers are integers**, so: **a / b = 7/2 = 3.5**
Integer Division and Remainder	/	The division operator performs integer div integer value is returned from the exampl **a / b = 7 / 2 = 3** and the fractional part of
		In Python, integer division is performed us **a // b = 7 // 2 = 3** and the fractional part o
	MOD or %	The remainder from an integer division op modulus operator. For example **7 MOD 2 = 1** can also be writ
	DIV	The DIV operator performs integer division and calculates the quotient and remainder So **7 DIV 2 = 3r1** (3 remainder 1).
Exponentiation	B**n	B^n the base number B is multiplied repeate exponent. For example: 12^4 = 12 * 12 * 12
Rounding	Rounding replaces a number with an approximate value languages have built-in functions to round data, for exa available to round a value to the nearest multiple of 10. So 34.5674 can be rounded up to 34.57 using the functi where 2 is the number of decimal digits required.	
Truncation	Truncation is the process of limiting the number of digit programming languages have built-in functions to trunc For example in Java the truncate library function is used number of decimal places. So 21.7546 is truncated to 21.75 using the function: **tru**	

1.1 – Progress Check

4. (a) Explain the difference between the truncation and rounding arithme
   (b) Calculate 11 MOD 3 (1 mark)
   (c) Calculate 3**2 (1 mark)

Relational operations in a programming language

Relational operators use different symbols in some programming languages, see exar

Operator	Java / C# / Python	Pascal / Visual Basic	True exa
equal to	==	=	7 == 7 or
not equal to	!=	<>	5 != 4 or
less than	<	<	6 < 1
greater than	>	>	15 >
less than or equal to	<=	<=	6 <=
greater than or equal to	>=	>=	10 >=

Boolean operations in a programming language

The relational operations listed above can be used in conjunction with the Boolean lo complete range of complex Boolean expressions needed by programmers.

<table>
<tr><th>Operator</th><th>Truth Tables</th><th>Pseudocode Examples</th><th></th></tr>
<tr><td>NOT</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><td>NOT (Year > 2014)</td><td>The log
Jav
Pyt
The exa
greate</td></tr>
<tr><td>AND</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>1</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table></td><td>(Age > 5) AND (Age < 15)</td><td>The log
Jav
Pyt
The exa
is olde</td></tr>
<tr><td>OR</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>1</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table></td><td>(Time < 9) OR (Time >5)</td><td>The log
Jav
Pyt
The exa
before</td></tr>
<tr><td>XOR</td><td><table><tr><th>A</th><th>B</th><th>A XOR B</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table></td><td>(Code =1) XOR (Code = 7)</td><td>The log
Pyt
The exa
code is</td></tr>
<tr><td colspan="4">Note the programming languages Pascal, Delphi and Visual Basic use similar symbols to P</td></tr>
</table>

1.1 – Progress Check

5. Design a truth table for the exclusive or function using variables A and B

Constant and Variable Data

A constant is data that uses a literal value, which doesn't change in the computer program. Also a constant can be given a meaningful identifier such as Pi, which would be set at 3.1416.

Variables are normally referenced by an identifier, such as Total, Height and Group. Unlike constants the data stored in variables can change as the program is executed. For example, the variable Total may be modified as further terms are added to it.

Advantages of using an identifier or name for a constant rather than a literal value:

- The program is more understandable when reading words or symbols than reading
- If you need to change the accuracy of **Pi** you only need to change it once even if it
- Constants with well-chosen labels or identifiers are easy to change for business re from time to time, so using a label makes it easy to change it only once in a progra

Extract from coding example using C#

```
const float Pi=3.1416;            // Initialise Pi
Circumference = 2 * Pi * Radius;  // Pi more meaningful than 3.
```

Extract from coding example using Pascal

```
CONST Pi=3.1416;                  {Initialise a constant value
Circumference = 2 * Pi * Radius;  { Pi more meaningful than 3.14
```

1.1– Progress Check

6. Explain the difference between a constant and a variable in a programm

String-handling operations in a programming language

The programming languages used on this course use a string datatype and have a series o information contained in a string. The syntax for string handling varies between programm handling operations are described in general terms below.

Operation	Description with examples
Length	The length of a string is a commonly available function such as: Len(s the length of a string. For example **Len("Computer Science")** would return a value of 16 an
Position	Items in a string are numbered from 0 to the length of the string. Cha based on their position in the string using the [ ] in Python. For example where s ="Computer", **s[2]** extracts character 'o' and **s[:3**
Substring	A substring is a string contained within another string. The substring c a position range input. For example, in C#: **"computer".Substring(0,0);** returns 'c' and **"comp**
Concatenation	The concatenation operation is used to join two strings together whe For example, in Pascal: **"Computer" + "Science";** returns 'Computer Sc
Character and Character Codes	A character code is a binary representation of the characters used in a keyboard characters which have a character code based on either ASC Conversion from a **character code to a character** can be carried out us For example, in Java the conversion from Unicode to the character is: **var character = String.fromCharCode(88)**; which returns 'X'.

String Conversion Operation Functions

Conversion	Visual Basic	Pascal	Java	
String to integer	CInt(s)	StrToInt(s)	Integer.ValueOf(s)	C
String to float	CDbl(s)	StrToFloat(s)	Float.ValueOf(s)	Co
Integer to string	CStr(n)	IntToStr(n)	Integer.ToString(n)	Co
Float to string	CStr(r)	FloatToStr(r)	Double.ToString(r)	Co
Date/time to string	CDate(d)	DateToStr(d)	SimpleDateFormat(d)	Co
String to date/time	CStr(s)	StrToDate(s)	SimpleDateFormat(s)	Co
The conversion operations listed are specific functions to carry out these string conversion could be used to convert 2 into a string '2'.				

1.1– Progress Check

7. Explain with an example, the string handling term 'concatenation'. (2 ma

Random number generation in a programming language

ⓘ **Random numbers** lack any sort of pattern; built-in functions can generate random nu

Program	Example to create a random number in the ran	

Visual Basic	`Dim rn As New Random`
	`Dim answer As Integer`
	`answer = rn.Next(1,10)`
		Declare a new object typ
		Declare an integer varia
		Next (1, 10) generates a

Python	`import random`
	`answer = random.radrange(1,11)`
		import random loads the
		random.randrange(1,11)
		10 as endpoint not inclu

Pascal	`Randomize ;`
	`answer := Random(10) +1 ;`
		Call Randomize procedu
		Random(10) generates a
		range 1 to 10

C#	`Random rn = new Random();`
	`int answer = rn.Next(1,10) ;`
		Declare an object type Ra
		Next (1, 10) generates a

Java	`Random rn = new Random();`
	`int answer = rn.nextInt(10) + 1;`
		Declare an object type Ra
		nextInt (10) generates a
		range 1 to 10

Exception handling

Exception handling is a technique used by the programmer to deal with error conditi
code will allow the program execution to continue under error conditions.

The simple pseudocode example indicates the principle of exception handling which
programming language chosen by the teacher.

```
ConsoleInput ← USERINPUT                          // Variable to be
TRY
       R ← ConvertToReal(ConsoleInput);           // Variable conve
                                                  can continue with
CATCH
       OUTPUT “Input was not a real variable”     // Error caught an
END TRY
```

Subroutines (procedures and functions)

Procedures and **functions** are useful in helping to provide a structure to create logica
There is a similarity between procedures and functions, in that they both support the
calling the function or procedure many times, rather than continually writing out or co

This approach reduces the amount of code and creates a more readable solution; they con
blocks that are executed by using an identifier in one or more places throughout the progr
the runtime version of the code at compile time.

Built-in functions create efficient code to solve commonly encountered problems. The
computer program developers. Software documentation gives a list of the built-in fun
to use them.

A typical example is the square root function or specifically **sqrt(val)** which is availab
Python.

So sqrt(9) returns the square root of 9 which is 3

Programming languages also allow the user the option to create their own functions
square function:

```
{Define a Function to square a number}
FUNCTION Square (Val)
       S ← Val * Val;
       Return S
END FUNCTION
```

Advantages of using procedures and functions

1. The same code is re-used, resulting in less code being needed (this makes the pro
2. The solution easier to understand and update when maintaining or fixing errors in

Parameters of subroutines

A **parameter** is a variable that is used as a data input or an argume

The definition for a subroutine normally includes a list of parameters, a
the arguments are assigned to the relevant parameters.

Returning a value / values from a subroutine

An example of a simple subroutine (VAT) is shown below; when the subroutine is called it based on the input parameter.

VAT example in C programming language	**Subroutine defined**

The function is named VAT and contains one input parameter named 'purchase_price'.
The parameter is data type double (floating point) and the data type returned is also double.
The function calculation details are within the {} brackets
	`double VAT(double purchase_p`
	`{`
	`    return 0.2 * purchase_pr`
	`}`

Once the function has been defined it can be called as shown on the right.	`total_price = purchase_price`

If the purchase price is 100 then **VAT** function returns 20.	`purchase_price      = 100`
	`total_price         = 100 +`
	`                    = 100 +`

Local variables in subroutines

ⓘ Where **local variables** are declared and used in a subroutine they are only in existenc and are only accessible or in scope within that subroutine.

It is good practice to use local variables rather than global variables in subroutines and fu content of a local variable in a large program and the subroutine retains modularity where variables are passed to it for execution.

Local variables are more efficient than global variables as you free up memory each time y

Global variables in a programming language

ⓘ Where **global variable**s are declared at the beginning of a program they can be acces program in contrast to local variables which are only accessible in the program block

Global variables should only be used if they are needed throughout the complete pro programming languages where variables are global unless declared in a function or s

Note this is not the case in Python where all variables are local unless declared explic

1.1 – Progress Check

8. Explain why it is good practice to use local variables rather than global

Recursive techniques

Recursion is where a subroutine or function is defined in terms of itself. The functi
as it calls itself during execution; the function is terminated when a logical conditi

The **general case** in recursion has a solution in terms of itself for a given input.

In recursion the **base case** has a solution that has no reference to the general case; the
to terminate the function.

Factorial example using recursion

```
Function Factorial ( N )
        If (N==1)
                Return 1
        Else
                Return (N * Factorial(N-1)
        End
```

Recursion example explained

If N is input at 4 then output will be b

4! = 4 * 3! = 4 * 3 * 2

In the example the **base case** is when N

So when the base case is reached, the

Factorial example using iteration

```
Function Factorial ( N )
        Answer = 1;
        FOR (K = 1; K<= N: K++)
        {
                Answer = Answer * K
        }
        Return Answer;
        End
```

Iteration example explained

Initialise answer to 1.

Loop through factorial calculation betw
If input N is 4, each pass of the loop w

Iteration Number	
1	
2	
3	
4	

So when the when K = N, the iterative

The recursive solution uses fewer variables than the iterative approach, but can be mor
makes it more difficult to maintain and can take more time to execute the code.

Role of stack frames in subroutine calls

A computer **stack** is an area of computer memory where data is stored, and that da
first-out (**LIFO**) fashion.

When a subroutine is called, an area of the stack known as a **stack frame** is allocated to store the following collection of data:

- Return address
- Parameters or arguments – used in the subroutine as an input variable
- Local variables
- Register contents

When the subroutine ends, control is passed to the saved return address and the stack frame is cleared from the stack.

Stack example for simple f

```
Function (in
        Float

        Return
End
```

Stack Frame Address
1
2
3
4
5

Note that **recursive function calls** are placed on a stack.

Once you hit the **base case**, each function call is then removed from the stack.

1.1 – Progress Check

9. Explain the term 'recursion'. (4 marks)
10. Compare recursion to an iterative approach in problem-solving. (4 mark

1.2 PROGRAMMING PARADIGMS

Programming paradigms

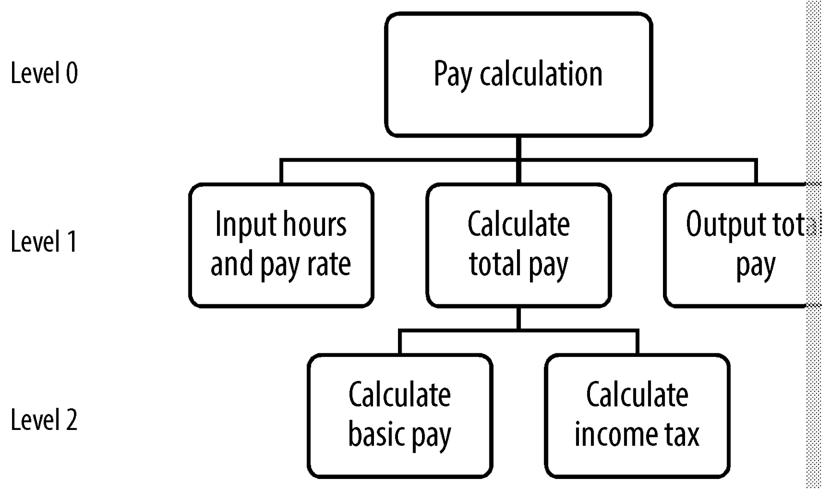
Procedural-oriented programming is a paradigm based on the use of procedure calls (also functions in some computer programming languages). In procedural programming languag and the programs follow a list of computational steps to be carried out and the order in wh

Object-oriented programming (OOP) languages combine the data and code for a routine to classes. OOP languages are based on objects that interact with each other and may inherit

Procedural-oriented programming

ⓘ **Structured programming** is a type of procedural-oriented programming where the pro to help reduce development time and ensure that the program is easier to understand

ⓘ **Hierarchy charts** are used to show the details of the modular structure in the program shows the modules involved in designing a simple pay calculation.



Advantages of structured approach

The structured approach to programming reduces the complexity of the task and has the f

1. Breaking down large programming tasks into manageable subtasks means that th several programmers, which saves development time.
2. Program test and debug time is reduced, where modularity helps to reduce the ti the errors that may occur.
3. Programs are easier to understand and, therefore, easier to maintain; also a new f introduction of an additional module.

1.2 – Progress Check

11. Describe the advantages of using a structured approach to programming

Object-oriented programming

Object-oriented programming includes the following concepts:

- **Class** describes shared attributes and methods, and is a template that can be used t
- **Object** is one instance of a class, such as a real-world entity.
- **Instantiation** is where instances of a class, known as objects, are declared.
- **Encapsulation** is the combining of data and code to form a new object or class.
- **Inheritance** is where a new class that is created (termed subclass) retains the attribu class it was based on (termed parent class).
- **Association aggregation** is where an object that is created can contain other objects then the composed objects will continue to exist.
- **Composition aggregation** is where an object that is created can be composed of oth deleted, then the composed objects will also be deleted.
- **Polymorphism** is where different objects in a class can be processed differently, ma
- **Overriding** is where a subclass can have a different execution for a method compare the base class.

Object-oriented techniques

An object-oriented technique is a design approach to create systems based on data structu that reflect real-world objects.

A **class** is a template that refers to the attributes and methods shared in each object, wher **attributes** are the properties or variables within a class and **methods** is the code or routines that operate within a class.

An **object** is the basis unit of object-oriented programming (OOP); in OOP, the programme specifies objects that contain data and the operations (or methods) performed on this data.

In OOP, the data and the operations that work on that data are contained in an object; this feature is termed encapsulation.

In procedural languages, it maybe unclear as to which variables operate in a function; this i not the case in OOP, where **encapsulation** provides a structure that arranges data and rela operations together in the same object.

Inheritance is a feature that allows the creation of a new class (or subclass) from an existi class; the new class shares some or all of the attributes of the original class. This is a useful feature in object-oriented programming as it makes use of code reusability, which saves th programmer time and reduces the length of coding.

In object-oriented programming, **polymorphism** allows objects in a class or subclass to be processed differently, making use of the same method.

An example of polymorphism could be: where shape is the original class, a method to calculate area will exist in subclass objects such as rectangle and circle; but the programm will define the area method to suit the particular subclass, such as length × breadth for rectan and π × radius^2 for circle.

Uses of object-oriented programming (OOP)

This technique relies on the creation of objects that contain attributes or data, and the ope
on a particular object. Examples of objects can include a range of items such as:

- people
- accounts
- shapes
- buildings

Initially, the programmer will identify all of the objects that they need to manipulate and t
objects. This concept is known as data modelling and is used to represent data flow in an e

There are many advantages in using an object-oriented programming approach, including t

- Programs are written as a series of modules based on objects or groups of objects, maintain software should a fault develop or a specification change be necessary.
- The modular nature of using OOP techniques is useful in developing a system wh to develop independently.
- New functionality can be added to a program by simply creating a new class or ob
- A class is only concerned with the data defined within it, so it is unlikely to access accidentally.
- Data can be hidden within the class that accesses it, providing greater system sec
- Objects can be set up to inherit attributes and methods to create reusable code in developed for as well as in other object-oriented programs.
- OOP code can be used to create code in software libraries for easy reuse.

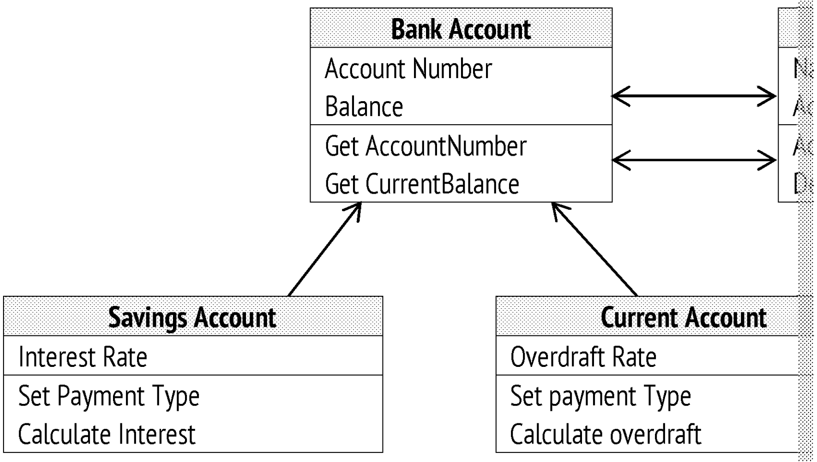
1.2 – Progress Check

12. Describe four advantages of using object-oriented techniques to solve pro
13. Explain the term 'polymorphism'. Illustrate your answer with examples. (5

Drawing and interpreting class diagrams

Bank account example – Inheritance with class and subclasses

The simplified example below shows a typical account system; the base class is a bank acc
of savings account and current account which can be represented by the generalised inherit



Using encapsulation, the properties are at the top of each class diagram and the methods a

Superclass (or base class)

The base class stores common properties; in this case, account number and balance are by t

Subclasses

Sometimes referred to as derived classes.

Savings account subclass contains details of savings accounts that they are responsible to

Current account subclass contains details of all current account transactions and associate

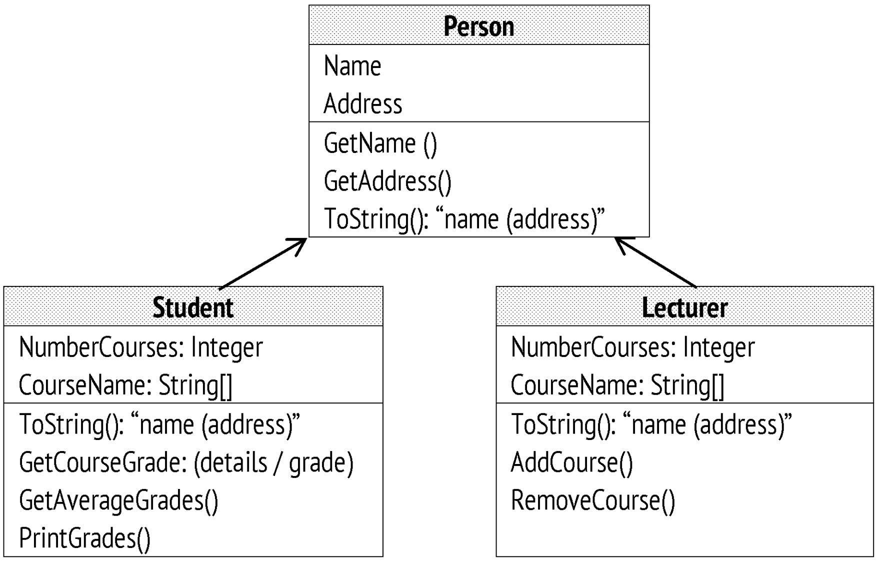
Information hiding is the principle that the details of the implementation of a class o not accessible by the user; the user simply needs the essential details of how to initia

Example – car manufacturers break down the process into a series of modules. Some elect model as well as the luxury model. Teams work on particular modules, and the detail of th modules is hidden from them.

So, for example, the type of entertainment system installed (basic or luxury) is hidden fro output module (speakers). This approach creates flexibility whereby the car manufacturer across the range of cars it produces.

Inheritance with class and subclasses

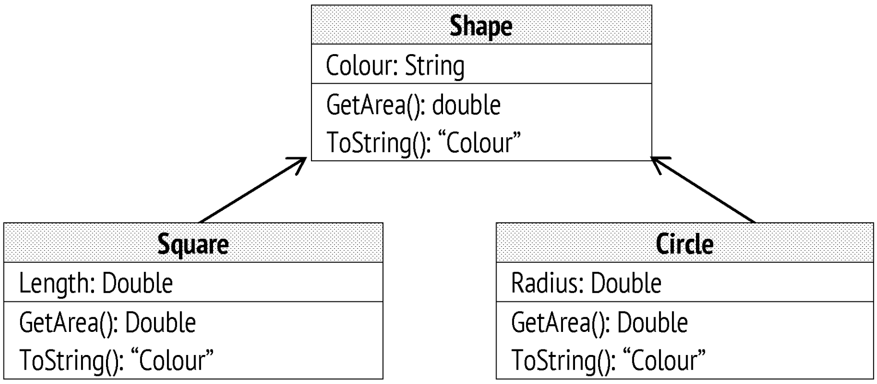
In a college there could be a class known as person which could include subclasses of stu and lecturer which can be represented by the generalised inheritance diagram below:



Polymorphism – Shapes example

The following example can be applied where a program makes use of different shapes, such as square and circle.

The superclass contains details of the shape colour and defines the behaviour of the other shapes; in this case, **all** shapes need to have an area method included.



1.2 – Progress Check

14. Define the following concepts in object-oriented programming:
    (a) Class (1 mark)
    (b) Object (1 mark)
    (c) Inheritance (1 mark)
    (d) Attributes (1 mark)
    (e) Encapsulation (1 mark)

TOPIC 2 – FUNDAMENTALS OF DATA S

2.1 DATA STRUCTURES AND ABSTRACT DATA TYPES

Data structures

ⓘ A **data structure** is the format used to efficiently store and organise a collection of rel
structures are chosen in computer programming for specific tasks; commonly used stru

Arrays

Single-dimensional arrays

ⓘ **Array** data structures are made up of a list of data elements that are the same data type and size.

Arrays can be one-dimensional, similar to a list as shown below:

The 'Subject' one-dimensional array has 6 elements;
Note that for most programming languages the indexes are addressed between 0 and 5 rather than between 1 and 6:
So Subject[3] = "Computing"

Index	Subject
0	English
1	Maths
2	Physics
3	Computing
4	Chemistry
5	French

Array Declaration & Assignments:

```
string Subject[6]
Subject[0] = "English";
Subject[1] = "Maths";
Subject[2] = "Physics";
```

Pseudocode Basic Examples:

Output to printer (Physics):

```
Output (Subject[2]);
```

Clear all string data from the array:

```
For Index = 0 To 5
   Subject[Index] = "";
Next
```

Array elements are often assigned in repetitive program code.

Multi

Arrays can also be two-di
a matrix).

The StudentTest array b
students (rows) and 4 sc

Rows = Students		0
	0	34
	1	12
	2	67
	3	46
	4	55

Array Declaration:

```
int StudentTest
```

Pseudocode Basic Exam

Output to printer (56):

```
Output (Student
```

Set all elements of the a

```
For Student = 0
     For Score =
          Student
     Next
Next
```

2.1 – Progress Check

1. Describe the term 'data structure'. (2 marks)
2. Describe the array data structure. (1 mark)
3. Explain the difference between a one-dimensional and two-dimensional a

Fields, records and files

ⓘ **Text files** store data in humanly readable format (usually ASCII) using a text editor.

Read/write to text files
Text files normally consist of a series of characters, separated by an end of line character, such as: \n.

The basic commands used for reading and writing to text files are shown on the right.

The commands shown are based on Python2, but different commands exist for different programming languages.

Create a file	`file = c`
Add a line of text	`f`
Close the file	`file.clo`
Open a file to read	`file = o`
Read 1st line of file	`p`
Read all lines of file	`p`
Append a file	`file = o`

ⓘ **Binary files** are not humanly readable and must be interpreted by the computer program or hardware.

```
0000    FF D8
0010    08 00
```

Read/write to binary files
The data returned when reading a binary file is presented in byte strings and not in readable text strings.

Some basic commands used for reading and writing to binary files are shown on the right. The commands shown are based on Python2, but different commands exist for different programming languages.

Note the modes are slightly different t becomes 'rb', etc.

Open a binary file to read	`file =`
Open a binary file to write	`file =`
Append a binary file	`file =`
Write bytes into a binary file using hexadecimal	`file.w`

2.1 – Progress Check

4. Explain the difference between a text file and a binary file. (2 marks)

Static and dynamic structures

ⓘ **Static data structure** is a data storage method where the amount of data and memor an array.

ⓘ **Dynamic data structure** is a data storage method where the amount of data and mem needs of the application using it; a typical example is a vector.

Key features of static and dynamic data structures are compared below:

Dynamic data structure	St
Memory allocation is varied to exactly meet the need of the application, which is an efficient use of memory.	Memory allocation is fi may assigned than is re
Memory addresses will be allocated during run-time, and could be fragmented, causing delays in access.	Memory addresses will will be fixed and contig
Memory addresses vary as the program is executed, so a changing relationship between data elements is inefficient.	Memory addresses are f data elements does not

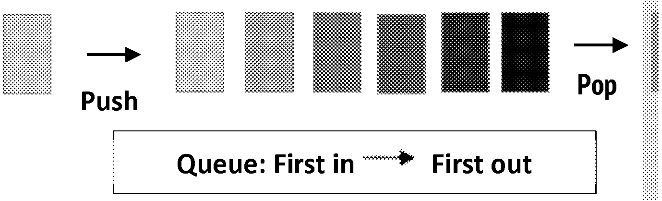
2.1 – Progress Check

5. Define the terms 'static data structure' and 'dynamic data structure', giv (2 marks).

2.2 QUEUES

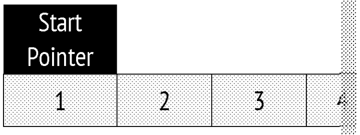
Queues

A queue is used as a temporary storage space for data. It is defined as an abstract data stru **first out (FIFO)** basis, where data is pushed into the queue and popped from it.



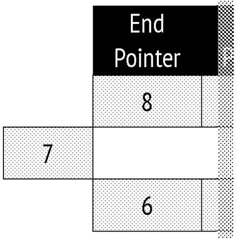
Queues can be implemented as **linear**, **circular** and **priority** structures, as described below.

ⓘ **Linear queue** is a FIFO data structure organised as list or line of data, as shown in the diagram on the right.



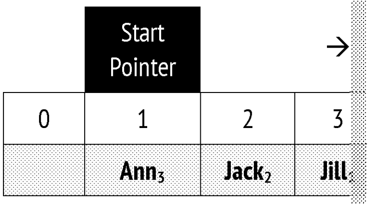
ⓘ **Circular queue** is a FIFO data structure organised in a ring where the start and end pointers wrap around from the last element of the array to the first element of the array.

See diagram.



ⓘ **Priority queue** is similar to a regular queue except that each element is assigned a priority and the highest-priority elements are served first. Where elements have the same priority, they are served according to a FIFO order.

See example of priority in diagram.



In the queue above, Jill will be serv highest priory.

Next Jack will be served, and finall lowest priority.

Queues are used to buffer data; for example, a file sent to a printer is stored in queue unti

A queue is used as a temporary storage space for data. It is defined as an abstract data stru **first out (FIFO)** basis.

The queue uses a two pointer (index) to keep track of the data in the queue, where:

- the **start** pointer indicates the data item that was first entered into the queue
- the **end** pointer indicates the data item that was last entered into the queue

Linear Queue Pseudocode

Assume initially Queue is empty so
Start pointer is set to 0 and
Queue has 'Max' elements

Push Example – '

	Start Pointer	
0	1	2
	Ann	Jack

Queue Push Algorithm

```
if start_pointer = 1 and end_pointer = MAX then
            return queue full message
else if start_pointer = end_pointer + 1 then
            return queue full message
else begin
            queue [end_pointer] = data;
            end_pointer = end_pointer + 1;
end
```

	Start Pointer	
0	1	2
	Ann	Jack

Queue Pop Algorithm

```
if start_pointer =1 then
            return queue empty message
else begin
            data = queue[start_pointer];
            start_pointer = start_pointer + 1
end
```

Pop Example – 'An

	→	Start Pointer
0	1	2
		Jack

Circular queue implementation is similar to the linear approach; this method allows new d
queue and data to be read from the front of the queue; this technique can be used efficien
buffer or in writing to a DVD.

2.3 STACKS

Stacks

A **stack** is an abstract data type where the last item added to the stack is the first to leave (LIFO) and that contains the following operations:

- **Push** – this function adds new items of data to the top of the stack. In the pseudocode function shown, the stack is first checked for overflow before the stack pointer is incremented and a new value written into the stack.
- **Pop** – this function removes the last item of data from the stack. In the pseudocode function shown, the stack is first checked to ensure it isn't empty, in which case there is no data to pop; otherwise, data is popped from the top of the stack and stack pointer is decremented.
- **Peek (or top)** which means to read the top element in the stack without removing it.

The diagram below shows an example of a stack, where element 1 (Smith) was entered first.

	4	
TopOfStack →	3	**Khan**
	2	**Jones**
	1	**Smith**

The pseudocode shows the practical application of testing for stack full or stack empty.

Pseudocode for pu
Stack is a String Arr
TopOfStack is an In

```
FUNCTION PushT
    # Test st
    data item
    IF (TopOf
          OUT
    available
    ELSE
          INP
          Top
          Sta
    NewStackI
    END IF
END FUNCTION

FUNCTION PopFr
    # Test st
    removing
    IF (TopOf
          OUT
    data to p
    ELSE
          OUT
          Top
    END IF
END FUNCTION
```

2.3 – Progress Check

6. Differentiate between the following types of queue:
   (a) linear queue (1 mark)
   (b) circular queue (1 mark)
   (c) priority queue (1 mark)

7. A stack contains the values 7, 6, 9, where 7 is the first value stored and 9
   List the final values in the stack after the following operations are made
   1. Pop
   2. Pop
   3. Push 12
   4. Push 16
   5. Pop
   6. Push 11

2.4 GRAPHS

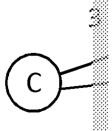
Graphs

Graphs are used to represent complex relationships such as computer interconnection dra used to represent complex relationships, where a graph consists of a series of nodes or ve vertices together.

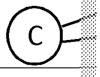
Vertex/Node – is a connector; for example, the two nodes shown could be used on a map represent towns A and B.

Edge/Arc – is used to connect two nodes or vertices together to form a simple graph, as shown.

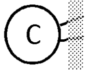
Weighted graph – is a graph that has labels on each edge to indicate a data value.
In the example, A, B and C could be towns and the data on each edge (3, 6 and 9) could represent a distance between the towns.



Undirected graph – is a graph where the direction between vertices can be either way. In the example, it is possible to travel in the direction of A to B or to travel from B to A.



Directed graph – is a graph where the direction of travel between vertices can only be one-way, as indicated by the arrow.
In the example, it is only possible to travel from A to C, from C to B and from B to A.



Graphs – adjacency list and matrix

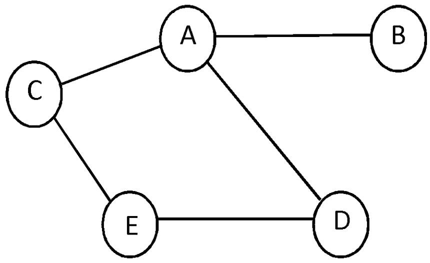
Adjacency list – used to represent a graph by listing each node and the nodes directly connected to it.

Adjacency matrix – used to and indicating where there

Node	Adjacent Nodes
A	C, D, B
B	A
C	A, E
D	A, E
E	C, D

List shows each node and its connected nodes.

Undirected graph



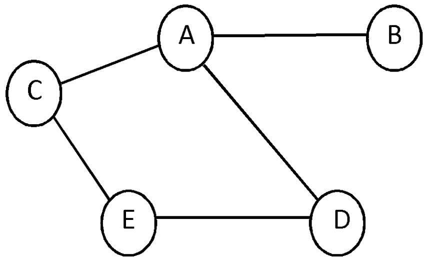
A
B
C
D
E

Matrix

Node	Adjacent Nodes
A	C, B
B	
C	E
D	A
E	D

List only shows the one-way relationship between nodes.

Directed graph



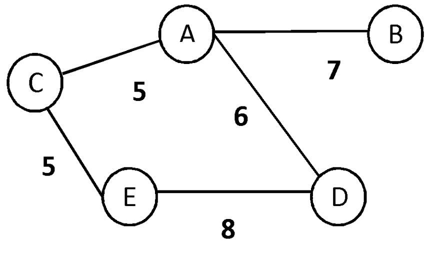
A
B
C
D
E

Matri

Node	Adjacent Nodes
A	B,7 C,5 D,6
B	A,7
C	A,5 E,8
D	A,6 E,8
E	C,5 D8

List shows weighted value between nodes, so the weight between A and B = 7.

Undirected weighted graph



A
B
C
D
E

Matrix
e

Adjacency list compared with adjacency matrix

Adjacency list only stores details of graph where an edge exists; consequently, it uses less storage/memory than an adjacency matrix.	**Adjacency matrix** store an edge exists as a 1 an uses more memory/stor
A **sparse** graph is one with few edges and, in this case, it is faster and saves memory to use an adjacency list to store the data set.	A **dense** graph is one w permutations of adjacen the data set as an adjac

2.4 – Progress Check

8. Explain the difference between a directed graph and an undirected graph
9. Explain the difference between an adjacency list and an adjacency matrix

2.5 TREES

Trees

A **tree** is an abstract data type based on an undirected graph that doesn't contain any cycle reached through one path. The following terms apply to tree structures:

- **Root** – this is the starting node that has no parents, and all other nodes branch o
- **Parent** – this node in the tree has further nodes branching from it, which are cal
- **Child** – this node in the tree has nodes above it in the structure, which are called
- **Leaf** – this node does not have any further nodes beneath it.

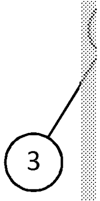
A rooted tree example is shown on the right, where:

1. the root is at the node A
2. node A is parent of nodes B and C
3. nodes B, D and E are leaves as they have no child nodes
4. there are no cycles

Binary tree – a tree structure where each node has a maximum of two child nodes; the binary tree is unordered since there is no relationship between a parent node and its child nodes.

Binary search tree – this is an ordered or sorted binary tree with the recursive relationship that for each node the left child is less than the node and the right child is more than the node.

This search method is covered in detail in section 3.4 Searching Algorithms.



The exa

Binary trees are used for:

- routing the best path in a network, such as a telephone network or electronic da
- storing data in an easy-to-find, searchable format using standard tree traversal m
- storing hierarchical data, such as directory tree for file management purposes

2.5 – Progress Check

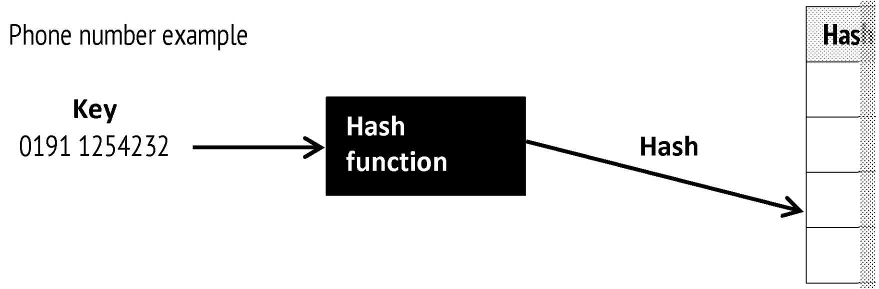
10. Describe the tree data structure and the terms that apply to it. (6 marks)
11. Explain the difference between a binary tree and a binary search tree. (4 m
12. Draw a binary search tree based on the following sequence of numbers: 8,

2.6 HASH TABLES

Hash table

ⓘ **Hash table** (or hash map) is a data structure that stores data in an associative way; function which maps a key to an index in an array to find the value of an array ele

ⓘ **Hash function** takes a data input known as a key and outputs an integer known as to a particular index in the hash table.



The diagram above shows the relationship between the key, the hash function and the function is used to store and to search for data.

Hash function algorithm
A simple example is based on using an array of seven elements to store five-digit mem
A common method uses modulo arithmetic to generate the addresses, so Address = (K

With seven elements in the array, use modulo 7 to generate the hash value
Address = (Key field) Mod 7

For membership number 81923
Address = (81923) Mod 7

Add the digits in the membership number **8 + 1 + 9 + 2 + 3 = 23**
Implement modulo 7 calculation so **23 / 7 = 3 remainder 2**
So the hash table index is 2 and the data is stored in location 2

Using the same method for membership number 41552
So the hash table index is 3 and the data is stored in location 3

H

Note that hashing can be applied to text fields by using the ASCII values of the text

ⓘ **Collision** is where two keys are applied to the hash function and create an identica chosen with the aim of minimising the chance of collisions.

The example used with Mod 7 can be used to demonstrate how collisions can occur.

Hash table index	0	1	2	3	
			81923	41552	
			24996		
			19231		

Calculation of the following membership numbers creates a collision in table index lo

Obviously, a hash function using modulo 7 is not sufficient to minimise collisions with
If we had 1,000 memory locations, we could have used Mod 1000 to produce “0 to 99
minimised the ease with which collisions occur.

Two methods available to deal with collisions are linear probing and separate chaining.

Linear probing is where a key is assigned to the next available slot in the hash tab with this technique is that it can create clustering, which is where more collision c index is not randomly distributed.

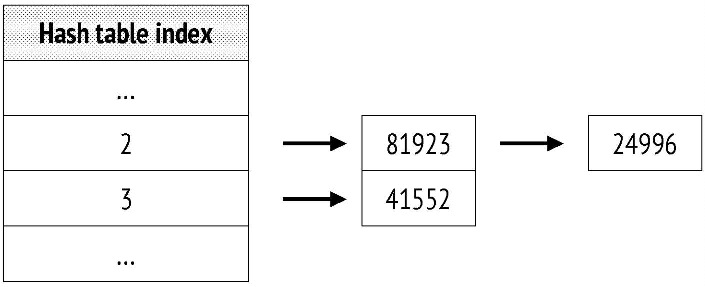
The Mod 7 example above can make use of linear probing, as shown below, where the da 4 (24996) and 5 (19231) of the table.

Hash table index	0	1	2	3	
			81923	41552	2

A **linked list** is a linear data structure in which a group of nodes contain data and m next data element.

Separate chaining is where the hash table index is a pointer to a linked list data st there are no collisions the linked list contains one element and the linked list will c particular slot.

The Mod 7 example above can make use of separate chaining, as shown below, where the linked list.



2.6 – Progress Check

13. Define the terms 'hash table', 'hash function' and 'collisions'. (6 marks)
14. Compare the suitability of linear probing and separate chaining as method (4 marks)

2.7 DICTIONARIES

Dictionaries

ⓘ **Dictionary** is a data structure that contains a collection of 'key-value' data elements w associated key.

The dictionary data structure is also known as an associative array and operates like a con or string is a key used to map to the definition or value. The following operations can be u

- **Add** or insert a key-value pair; note that the data in the dictionary is unordered.
- **Find** or search for a key returns the value associated with a specific key.
- **Delete** removes the key and its associated value.

One use of **information retrieval** in dictionaries is based on an algorithm for counting wor
For example:

The text "Row, row, row your boat" could be represented by the dictionary:
{ 'row' : 3, 'your' : 1, 'boat' : 1} ignoring letter case

A dictionary data structure can be viewed as a simple **two-dimensional array** and can be u look-up table:
For example:

In the 2D array shown for greetings card, the key value is the card type string an
{ 'Birthday': £3.25, ' Congratulations': £3.75, 'Driving Pass'; £2.95, 'Graduatio

There are similarities between hash tables and dictionaries; hash tables are widely used a dictionary data structures.

2.8 VECTORS

Vectors

ⓘ **Vectors** in programming data structures are similar to arrays; however, the size of an a automatically grow to store additional elements. The individual elements of a vector c index.

A vector can be represented in several ways; for example:

List of real numbers

In this case, a 4-vector over $\mathbb{R}$ (can be written as $\mathbb{R}^4$) can be expressed as:
[1.2, 5.25, -4.3, 1.75]

Note: R^4 means a vector that contains four real numbers.

A 4-vector over $\mathbb{R}$, can be written as $\mathbb{R}^2$ when it contains 2 real numbers and can be e

Functional interpretation is based on the list above, so: 0 → 1.2 1 → 5.25 2 → -4.3 3 → 1.75	**f: S → R** So the function f is Set **S = {0, 1, 2, 3}** mapping to the codomain of real nu

Geometric vector
A geometric vector can be drawn as a line with an arrow, to show magnitude and dire

A **dictionary representation** is useful for describing the functional interpretation of a vecto

So, the function **f: S → R** described above can be represented as a dictionary:

{0: 1.2,   1: 5.25,   2: -4.3,   3: 1.75}

1D array representation of an vector

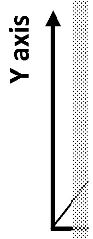
Vectors are similar to arrays, but can be declared differently dependent upon the programm

For example, a 4-vector over ℝ can be expressed as:

- Dim V(3) As Single (VB.NET)
- var V = new FloatVector(3); (C#)

Visualising a vector as an arrow

In geometry, vectors can be visualised as arrows, and they consist of two components: magnitude and direction.



Magnitude is the length of the line and **direction** is indicated by the arrowhead.

The tail of the vector is sited at the origin (0, 0) and the head is at a geometric point in space value (x, y), where x and y are known as the components of the vector.

For example, the arrow of the vector A = (5, 6) so the vector distance from the origin (0, 0) gives an x-component = 5 and a y-component = 6.

Note that three-dimension vectors can be represented using the A component for the z ax so using an arrow for a 3D vector, A = (x, y, z).

Vector addition

Vectors can be added by drawing them to scale and joining the head of one to the tail of another, as shown in the diagram on the right.

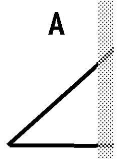
An alternative method is to sum all the x-components of each vector and all the y-components of each vector.

For example: A = (9, 11) and B = (3, 7)

So A + B = ((9+3), (11+7)) = **(12, 18)**

Vector addition achieves a translation of the resultant vector.

The vecto



It is not al vectors us

Vector addition (3D)

Vector addition can be calculated using a similar technique for three dimensions, as each v (x, y, z)

For example A = (9, 11, 4) and B = (3, 7, 6)

So A + B = ((9+3), (11+7), (4+6)) = **(12, 18, 10)**

Scalar–vector multiplication

A vector can be multiplied by a number known as a scalar to rescale the magnitude of the v

For example: Calculate the magnitude of vector B when vector A = (9, 11) is multiplie

So A = (9, 11) and vector B = **(27, 33)**

Note that the direction of the vector B is the same as vector A; it is only the magnitude of t

Dot product of two vectors

The dot product is obtained by multiplying two vectors together to obtain a number or sca dot product operation is not a vector.

The dot product for two vectors a(x,y) and b(x,y) is determined using the formula $\mathbf{a \cdot b = a}_x$

Example 1: calculate the dot product of vectors a(7,4) and b(9,2)

$a \cdot b = a_x \times b_x + a_y \times b_y = 7 \times 9 + 4 \times 2 = 63 + 8$

$\mathbf{a \cdot b = 71}$

The same principle can be applied to vectors in three dimensions using the formula: **a · b**

Example 2: calculate the dot product of vectors a(7,4,-5) and b(9,2,4)

$a \cdot b = a_x \times b_x + a_y \times b_y + a_z \times b_z = 7 \times 9 + 4 \times 2 + (-5) \times 4 = 63 + 8 - 20$

$\mathbf{a \cdot b = 51}$

Applications of dot product

The specification lists the application of generating a parity bit between two vectors **u** and **v** over GF(2) as an application of dot products.

GF(2) refers to finite field arithmetic that contains the binary elements {0,1}, where multiplication is the equivalent of AND, and addition is the equivalent XOR.

Even parity can be calculated for vectors u and v using the dot product with bitwise AND (multiply) and with XOR (addition) operations:

u · v = [1, 1, 1, 1] · [1, 1, 1, 0]
= 1 AND 1 XOR 1 AND 1 XOR 1 AND 1 XOR 1 AND 0
= 1 XOR 1 XOR 1 XOR 0 = 1

The even parity bit is one as there is an odd number of bits in vector v.

GF(2) arithm

Mult

Multipl

A

Addi

2.8 – Progress Check

15. Define the ‘dictionary’ data structure and describe one use of this algorithm
16. Define the vector data structure. (2 marks)
17. Use vector addition to calculate A + B in the following cases:
    (a) A = (4, 13) and B = (5, 2) (1 mark)
    (b) A = (19, 12, 14) and B = (13, 16, 5) (1 mark)
18. Calculate the magnitude of vector B when vector A = (5, 12) is multiplied b
19. Calculate the dot product of vectors A(7,4) and B(9,2). (1 mark)

TOPIC 3 – FUNDAMENTALS OF ALG

3.1 GRAPH TRAVERSAL

Graph traversal

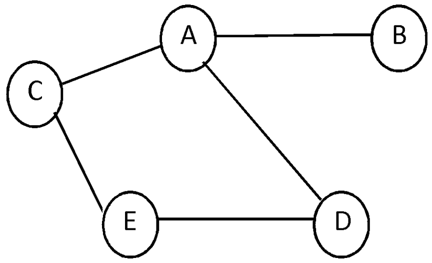
Graph traversal is the process of visiting each vertex in a graph. The vertices can be first algorithm.

Graph traversal is similar to tree traversal except that graphs can contain cycles, so the sa
Tree traversal algorithms are covered later in this section with associated pseudocode.
The undirected graph implemented below is used to demonstrate these traversal methods

Node	Adjacent Nodes
A	C, D, B
B	A
C	A, E
D	A, E
E	C, D

List shows each node and its connected nodes.

Undirected graph



A
B
C
D
E

Matrix

Depth-first graph traversal

Depth-first graph traversal starts at the chosen node and explores each branch in de backtracking to the remaining unvisited nodes.

This algorithm uses a list of visited nodes to ensure that each node is only processed once

Node	Visited Node	Description
A	-	Start at node A
A	A	Choose node connected to A, which is node C
A, C	C	Mark C as visited; traverse to node E
A, C, E	E	Mark E as visited; traverse to node D
A, C, E, D	D	Mark D as visited; no further nodes on this trave
A	-	Choose unvisited node connected to A, which is
A,B	B	Mark B as visited; no further nodes to visit so th

Breadth-first graph traversal

Breadth-first graph traversal visits the neighbouring nodes first and then continues queue is used to note the nodes that have been visited.

Apply the following rules:

1. Visit the adjacent unvisited node, note it and add it to the queue.
2. When there is no adjacent vertex, remove the first vertex from the queue.
3. Repeat these rules until the queue is empty.

Queue	Description
A	Start at node A as before, and add A to the queue
ABCD	Add nodes B, C and D as they are adjacent to A
CD	Remove nodes A and B from queue as fully explored
CDE	Add node E as it is adjacent to nodes C and D
E	Remove nodes C and D from queue as fully explored
	Remove node E as it fully explored – queue is empty so the graph has

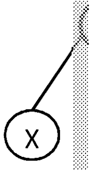
3.2 TREE TRAVERSAL

Tree traversal

Tree traversal is a process that ensures that each node in a tree data structure is visited just once. The output from a tree traversal is a list of the nodes in the order or sequence visited.

There are three methods of tree traversal:

- **Pre-order** (root, left, right)
- **Post-order** (left, right, root)
- **In-order** (left, root, right)



Pre-order traversal

Pre-order traversal is a method where the parent node (or root) is visited first, followed by

The pre-order sequence in the example above is: **A, F, X, G, D, E**.
Pre-order traversal uses include:

- creating a copy of the tree
- creating a prefix expession from an expression tree; where prefix expression + A precedes the operands

Post-order traversal

Post-order traversal is a method where the left and right children (or subtrees) are visited fir

The post-order sequence in the example above is: **X, F, D, E, G, A**.
Post-order traversal is used to convert infix to RPN (Reverse Polish Notation). The postfix B, as the operands precede the operator.

In-order traversal

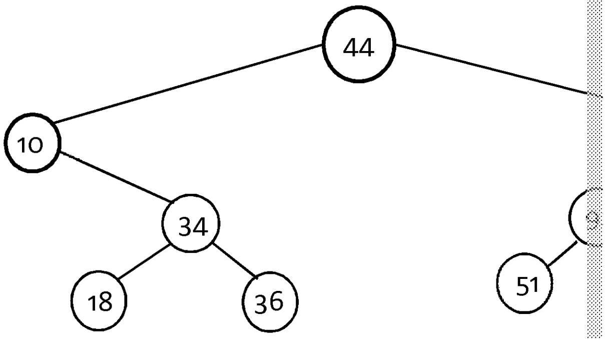
In-order traversal is a method where the left child (or subtree) is visited first, then the pare the right child (or subtree).

The in-order sequence in the example above is: **X, F, A, D, G, E**.

In-order traversal is used in binary search trees, which is described in the search algorithm

3.2 – Progress Check

1. Compare the terms 'depth-first graph traversal' and 'breadth-first graph t
2. Trace the tree traversal in the diagram below using pre-order and post-o



3.3 REVERSE POLISH

Reverse Polish Notation

Reverse Polish Notation (or RPN) is used in mathematical expressions where the m the operands. RPN is frequently referred to as **postfix notation** and it eliminates the parentheses in expressions.

Infix notation is commonly used in mathematical statements where the operator is notation makes use of brackets to force precedence between mathematical operand Typical conversion examples between infix and postfix are shown in the table below

Infix	Postfix (or RPN)
A + B	A B +
(A + B) * C	A B + C *
(A + B) * (C + D)	A B + C D + *

RPN expression evaluations

Infix expressions can be converted to postfix expressions and can be evaluated using a stack. So **infix (3+4)*(1+3)** can be converted to **postfix 3 4 + 1 3 + ***

Stack	Evaluation explanation
3	Push 3 onto stack
3 4	Push 4 onto stack
3 4 +	Push + (sum) onto stack
	+ is an operator, so Pop the operands and evaluate 3 4 +
7	Push the answer (7) back onto the stack
7 1	Push 1 onto stack
7 1 3	Push 3 onto stack
7 1 3 +	Push + (suml) onto stack
7	+ is an operator, so Pop the operands and evaluate 1 3 +
7 4	Push the answer (4) back onto the stack
7 4 *	Push * (multiply) onto stack
	* is an operator, so Pop the operands and evaluate 7 4 *
28	Push the result (28) back onto the stack

3.3 – Progress Check

3. Convert the following expressions into infix: (2 marks)
   (a) 7 4 /
   (b) 7 3 + 2 *

4. Convert the following expressions into postfix: (2 marks)
   (a) (7 - 3) / 4
   (b) (3 * 4) + (6 / 2)

3.4 SEARCHING ALGORITHMS

Big-O notation

Big-O notation is used to determine the time complexity or run-time of the code as the vo are references to Big-O in this section.

The theory of Big-O notation is covered in section 4.4 Classification of Algorithms.

Linear search

Linear search

In this method it is not necessary for the search data to be sorted.

```
Linear Search (Int A[ ], int Target)
Found ← False
FOR (i=1 TO N)
    IF (Target == A[i]) THEN
            OUTPUT "Item Found" Item 'At Location' i
            Found ← True
            Break /* Item Found so break from loop
    END IF
END FOR
IF (Found == False) THEN
    OUTPUT "Item does not exist in array"
END IF
```

Refer to solution

A linear techniqu search c compare list until

If the ta element message

Worked example: Use linear search to find the following numbers from the unordered list needed to find each number:

(a) 902
(b) 370
(c) 41

1	2	3	4	5	6	7	8	9	10	11	Index
											Comments
12	14	6	89	41	56	378	370	657	905	902	Data set nu
12	14	6	89	41	56	378	370	657	905	902	(a) 902 = 11
12	14	6	89	41	56	378	370	657	905	902	(b) 370 = 8th
12	14	6	89	41	56	378	370	657	905	902	(a) 41 = 5th

Note that the number of comparisons needed to find the target is directly related to the ta

Time complexity for a linear search is **O(n)**, so if the list is doubled, it will take double the

3.4 – Progress Check

5. Explain the difference between searching and sorting in computing. (2 mar
6. Describe the method used by the linear search. (2 marks)

Binary search

In this method, the search data needs to be sorted.

```
Binary Search (Int A[ ], int Target)
Left ← 1
Right ← Size[A]
WHILE (Left <= Right)
    Middle ← (Left + Right) / 2
    IF (A[Middle] = Target) THEN
      Output “Target found at “ A[Middle]
    ELSE IF (A[Middle] < Target) THEN
      Left ← Middle + 1
    ELSE
      Right ← Middle – 1
    END IF
END WHILE
```

Refer to the binary search p
A binary search operates a
search criterion.

1. First, compare the ta list.
2. Stop if the target = t
3. If the target is less t left-hand list.
4. If the target is greate right-hand list.
5. Repeat the process o the target is found.

Worked example: Trace the stages of a binary search to find the number 4 from the list be

1	2	3	4	5	6	7	8	9	10	11	Index
											Comments
3	4	12	18	21	36	37	49	61	200	702	Data set nu
3	4	12	18	21	36	37	49	61	200	702	INT(1+11)/2 = 36 > 4 so ch
3	4	12	18	21	36	37	49	61	200	702	INT(1+6)/2 = 12 > 4 so ch
3	4	12	18	21	36	37	49	61	200	702	INT(1+3)/2 = Number 4 fo

Note that the search was successful with three comparisons.

Binary search characteristics

Advantage – the binary search is more efficient than the linear search, as elements can be found with few comparisons. See table on right.

Disadvantage – the data that has to be searched needs to be in order for the binary search.

Time complexity for a binary search is **O(log n)**, so if the list is doubled, the difference in run-time is much lower than the linear search.

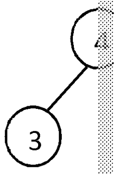
Total eleme
2
4
8
16

Binary tree search

Binary search tree is an ordered or sorted binary tree with the recursive relationship that for each node

- the left child is less than the node
- the right child is more than the node

The binary search tree shown on the right is ordered or sorted based on the following sequence of numbers: 6, 4, 9, 8, 11, 3, 5.
Time complexity for a binary search is **O(log n)**, so if the list is doubled, the differen



3.4 – Progress Check

7. Explain why a binary search method is more efficient than a linear search m
8. Trace the stages of a binary search for the number 18 from the list {1,4,6,7,
9. Describe one advantage and one disadvantage of using a binary search tech
10. Explain the binary search tree algorithm using a recursive process. (3 marks

3.5 SORTING ALGORITHMS

Bubble sort

Sort algorithms are normally used to organise data in an array or a list into order.

Bubble sort

In the following algorithm an array ‘S’ of ‘N’ elements will be sorted using the bubble sort method.

```
BubbleSort(int S[ ], int N)
Swapped ← True
j ←  0
WHILE (Swapped ==True)
        Swapped ← False
        j ← j + 1
        FOR (i=1 TO N-j)
                IF (S[i-1] > S[i]) THEN
                        Temp ← S[i-1]
                        S[i-1] ← S[ i ]
                        S[ i ] ← Temp
                        Swapped ← True
                END IF
        END FOR
END WHILE
```

Refer to the bub

The algorithm o

- Initially the that when bubbled) to
- The inside in the array
- If the first next (S[i]),
- Once the F incremente element is

Note that for an will repeat N-1 t

Worked example: Sort Array S with six elements (8,9,3,6,2,7)

Bubble sort hand traced to help explain the process.
Note that for array of six elements, the outside loop j will repeat five times.

j	Swapped	8	9	3	6	2	7	Origi
1	False	8	9	3	6	2	7	No ch
	True	8	3	9	6	2	7	9 > 3 s
	True	8	3	6	9	2	7	9 > 6 s
	True	8	3	6	2	9	7	9 > 2 s
	True	8	3	6	2	7	9	9 > 7 s
2	True	3	8	6	2	7		8 > 3 s
	True	3	6	8	2	7		8 > 6 s
	True	3	6	2	8	7		8 > 2 s
	True	3	6	2	7	8		8 > 7 s
3	False	3	6	2	7			No ch
	True	3	2	6	7			6 > 2 s
	False	3	2	6	7			No ch
4	True	2	3	6				3 > 2 s
	False	2	3	6				No ch
5	False	2	3					No ch

2	3	6	7	8	9	Array sorted using

Advantages – The bubble sort is a useful tool where there is very little memory available, as it only uses the memory which the array or list occupies. It requires a small amount of code and so is simple to understand and implement.

Disadavantges – The bubble fully sort, where n is the size out-of-position item is only m Time consuming for large da

Merge sort

Merge sort is an example of a divide and conquer algorithm.

The pseudocode below shows a merge sort for an integer array 'S' of 'N' elements.

MergeSort is based on the fo
- Divide the array or list in
- Merge sort the first half o
- Merge sort the second ha
- Once both halves of the a together to complete the

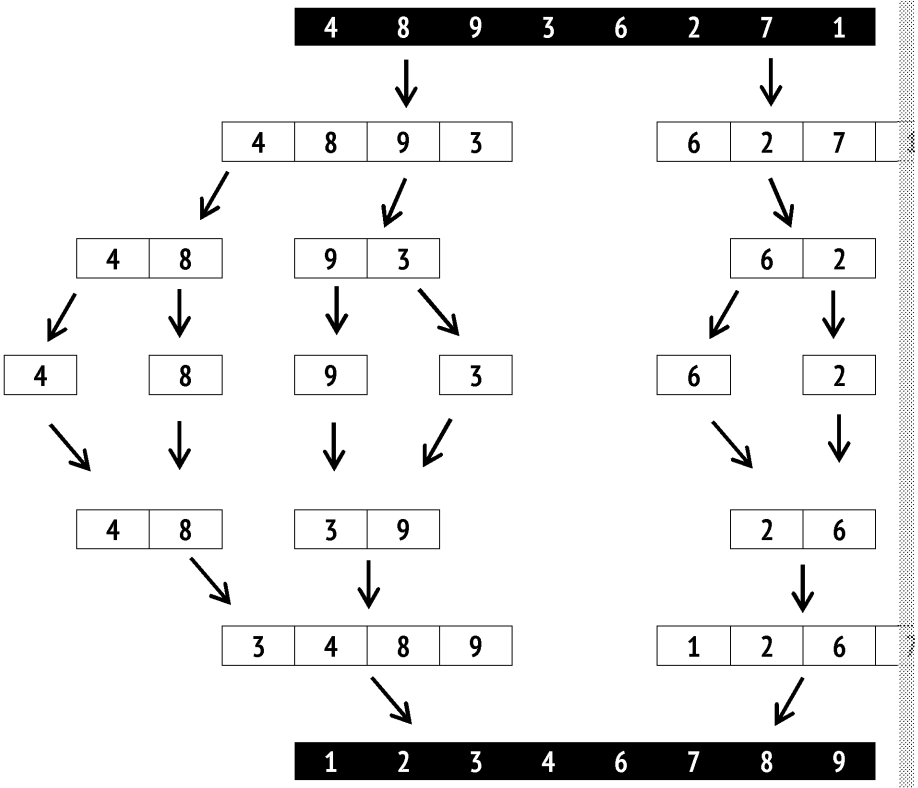
```
FUNCTION MergeSort (S[1:N])
BEGIN
      IF (N = 0 OR N = 1)
            RETURN S                                  // Return array
      ELSE
            Mid ← ((1 + N)/2)                         // Calculate mid
            Left ← MergeSort(S[1:Mid])                // Sort Left an
            Right ← MergeSort(S[Mid:N])
            Result ← Merge(Left, Right)               // Finally merge
            RETURN Result                             // Return the
      END IF
END MergeSort
```

The Merge Function pseudocode to merge the Left and Right arrays is shown below.

```
FUNCTION Merge (Left, Right, Result)
BEGIN
      i, j ←1                                                 // In
      WHILE (Left is not empty AND Right is not empty)        // Co
            FOR (K =1 TO N)                                   // Lo
                  IF Left(i) < Right(j)                       // Le
                        Result(K) ← Left(i)
                        i++                                   // In
                  ELSE  IF Left(i) > Right(j)                 // Ri
                        Result(K) ← Right(j)
                        j++                                   // In
                  END IF
            END FOR
      END WHILE
      IF (Left = empty)                                       // Le
            Insert Right remaining elements to Result         // So
      ELSE
            Insert Left remaining elements to Result          // Ad
      END IF
END Merge
```

Worked example:

Trace the stages in sorting the list using a merge sort with eight elements (4,8,9,3,6,2,7,1)



The diagram above shows how the divide and conquer principle is used in the merge sort.

Divide: The first stage is to divide each list into a series of smaller lists.

Conquer: Once the lists are subdivided, they are sorted and successively merged until ther

Advantages	
Merge sort is an algorithm that is used to rearrange data in lists or to sort sequential data from file streams. Merge sort is a fast technique with average and worst case performance of **O(n log (n))**.	Merge sort uses additi the array of merged da lists.

3.5- Progress Check

11. (a) Briefly describe in words the bubble sort process. (2 marks)
    (b) Explain a disadvantage of using the bubble sort approach. (2 mark

12. Trace the stages in sorting the list using a merge sort with eight elemen (4 marks)

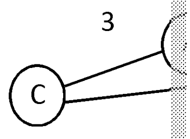
3.6 OPTIMISATION ALGORITHMS

Dijkstra's shortest path algorithm

Dijkstra's shortest path algorithm is used to determine the shortest distance between two vertices on a weighted graph.

Shortest path is the shortest distance between two vertices (or nodes) on a graph based on the weight of their edges.

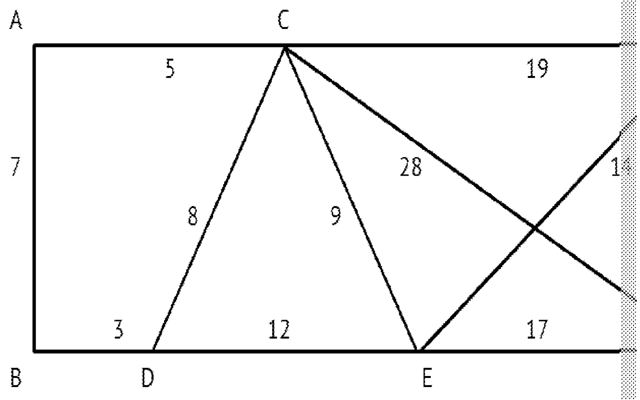
Weighted graph is a graph that has labels on each edge to indicate a data value.



In the weighted graph
and the data on each e
distance between the t

The shortest distance f
by trial and error; by in
is 8.

Tracing Dijkstra's algorithm worked example



The task is to trace the shortest path between vertex A and vertex G on the weighted graph
The trace technique is broken down into a series of steps, as outlined below.

Step 1 – Start trace from vertex A and tabulate results:

- Distance between A and A is 0; record this with subscript A
- Distance between A and B is 7; record in table
- Distance between A and C is 5; record in table
- All other vertices cannot be reached directly from A, so they are unreachable; record this using **∞** symbol

Vertex	A
A	0

Shade in cells
in this case, **0_A**

Step 2 – Next, trace from vertex C which was smallest number in table:

- Distance from A to D via C = 5 + 8 = 13; record this with subscript C
- Distance between A to E via C = 5 + 9 = 14; record this with subscript C
- Distance between A to F via C = 5 + 19 = 24; record this with subscript C
- Distance between A to G via C = 5 + 28 = 33; record this with subscript C

Vertex	A
A	0
C	0

Shade in cells
in this case, **0_A**

Step 3 – Next, trace from vertex B:

- Distance from A to D via B = 7 + 3 = 10; this is a shorter distance than before, so record this value with subscript B
- Distance between A to C via B = 7 + 3 + 8 = 18; larger distance, so don't record this value

Vertex	A
A	0
C	0
B	0

Shade in cells
in this case **0_A**

Step 4 – Next, trace from vertex D:

- Distance from A to E via D = 7 + 3 +12 = 22; larger distance, so don't record this value
- Distance between A to C via B = 7 + 3 + 8 = 18; larger distance, so don't record this value

Vertex	A
A	0
C	0
B	0
D	0

Shade in cells
in this case **0_A**

Step 5 – Next trace from vertex E:

- Distance from A to F via E = 7 + 3 +12 +14 = 36, larger distance so don't record this value
- Distance between A to G via E = 7 + 3 + 12 + 17 = 39, larger distance so don't record this value

Vertex	A
A	0
C	0
B	0
D	0
E	0

Shade in cells
in this case **0_A**

Step 6 – Next trace from vertex F:

- Distance from A to G via F = 5 + 19 + 5 = 29, this is a shorter distance than before so record this value with subscript F

No more improvements can be made so the trace is completed.
The shortest distances between A and G is 29.

The final row of the table contains the shortest distance between A and each of the other vertices.

Vertex	A
A	0
C	0
B	0
D	0
E	0
F	0

Shade in cells
in this case **0_A**

The pseudocode for the Dijkstra shortest path algorithm is shown below.

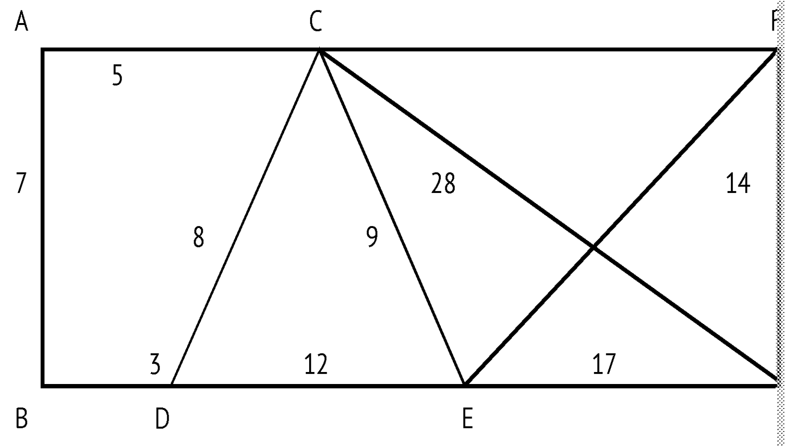
```
FUNCTION Dijkstra (Graph, source)
BEGIN
      FOR (each vertex V in graph)                      // Initialisation loo
            Distance(V) ← Infinity                      // Set initial distan
            Previous(V) ← Undefined                     // Set previous nod
            Q ← Set of all nodes in the graph           // Assign all nodes in
      END FOR

      Distance(Source) ← 0.0                            // Distance from so
       WHILE (Q NOT empty)                              // Loop through as
            U ← Node in Q with min Distance(U)          // Source node sele
            Remove U from Q                             // Assign as minimum

            FOR (each neighbour V of U)                 // Loop through no

                  // Calculate alternative distance between U and V
                  Alternative ← Distance(U) + Distance between
                  IF (Alternative < Distance(V) THEN          // Shorte
                        Distance(V) ← Alternative             // Update
                        Previous(V) ← U                       // Update
                  END IF
            END FOR
      END WHILE
RETURN (Distance(), Previous())
```

Uses of Dijkstra's shortest path algorithm
There is a variety of uses for this shortest path algorithm, including:

- determining the shortest route between two places on a map
- determining the shortest route for data packets in network switching

3.6 – Progress Check

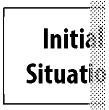
13. Use Dijkstra's algorithm to work out the shortest distance between B and F. (



TOPIC 4 – THEORY OF COMPUTA

4.1 ABSTRACTION AND AUTOMATION

Problem-solving



Problem-solving involves reaching a desired outcome from an initial situation.

The first stage in problem-solving is to gain a good understanding of the problem that is to be solved.

The next stage is to create a definition of the problem, which involves the following components:

- The initial situation
- The desired outcome
- The resources available to solve the problem
- Responsibility for planning and implementing

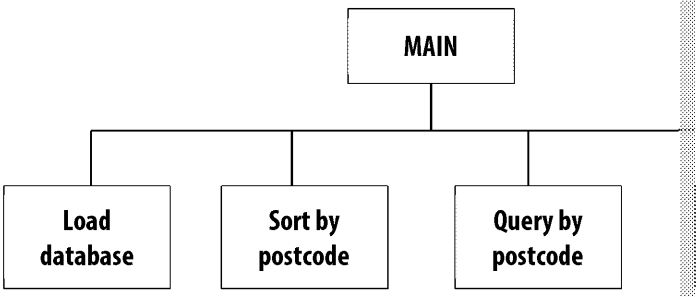
Planning a solution involves deciding upon a plan of action or strategy to solve the problem. solve the problem on paper using a range of assumptions to simplify the problem. Most pr series of smaller problems or sub-problems that are easier to solve, which is known as a top

ⓘ **Top-down design** or stepwise refinement is used to plan a solution based on a top-dow

Using this approach a complex problem can be solved by breaking it down into a series of sm down further into even smaller steps. The smaller the steps the easier it is to both understa and eventually the whole problem.

Example: Addressing labels
A travel agent wishes to send a brochure to each of their customers in a certain area, based

Solution: Load the database and create a query for the customer file based on a selected po with the relevant address. The hierarchy chart below shows the top level of the design.



Each of the steps can then be broken down further to add more detail for the solution, so fo function includes additional detail:

1. Switch on printer
2. Load envelopes into printer tray
3. Run postcode query on database
4. Select print query from database

The other functions: 'load database', 'sort by postcode' and query by postcode' can also be b
The top-down design should include enough steps for the designer to create the algori then further steps need to be added, hence the term 'stepwise refinement'.

There are several **advantages of top-down design** as listed below:

- Breaking the problem into parts helps to clarify exactly what needs to be achieved
- Each stage of the refinement process creates smaller sub-problems that are easier to
- Some functions or parts of the solution might be reusable
- Breaking a design into parts allows more than one person to work on the solution.

4.1 – Progress Check

1. Explain the term 'problem-solving'. (2 marks)
2. Describe top-down design. (4 marks)
3. Describe the advantages of using top-down design. (4 marks)

Following and writing algorithms

ⓘ An **algorithm** is a step-by-step approach to solving a problem. It is normally written in any particular programming language.

ⓘ An algorithm can be expressed using **pseudocode**; this is a written list of steps that a connected to any particular programming language.

Note that pseudocode can be written in many styles, but it should be in sufficient detail to based on it. In the examples below, comments are made using the **'#'** Symbol; for example

The solutions to simple problems can be written in pseudocode using one or more of these selection, and iterations (see Topic 1 for information and examples).

4.1 – Progress Check

4. Describe the term 'algorithm'. (2 marks)
5. Create algorithms in pseudocode to solve the following problems:
   (a) Calculate the area of a square from a value entered by the user. (3
   (b) Calculate the area of all the squares with values 2, 3, 4 and 5. (5 m

Hand-trace algorithms

Hand tracing is a form of dry run testing where a program is tested and variables are recor table can be used which contains columns for the expected answers. The variables used in error it can be detected and the code corrected.

The simple example below demonstrates the dry run process to calculate the mileage charge for a car hire company.

```
# Luxury Car Hire Mileage Solution – the car hire firm makes
a charge for hiring the car and also charges for the mileage
driven by the hirer.

START
   # Data declarations
   INTEGER Mileage
   FLOAT Cost

   # Input / Output
   OUTPUT "Enter Total Mileage used"
   Mileage ← USERINPUT

   # Car hire mileage cost calculation
   IF (Mileage < 50) THEN
      # First 50 miles charged at 5p per mile
      Cost ← Mileage * 0.05
   ELSE IF (Mileage < 200) THEN
      # Next 150 miles charged at 25p per mile
      Cost ← (Mileage – 50) * 0.25 + (50 * 0.05)
   ELSE
      # Mileage above 200 charged at 40p per mile
      Cost ← (Mileage – 200) * 0.40 + (150 * 0.25) + (50 * 0.05)
   END IF
      OUTPUT "Mileage Cost for car hire = £" Cost
END
```

Test No.	Mileage	Cost
1	0	
2	45	= 4
3	160	= 50 *0.05 + 90
4	280	= 50 *0.05 + 150 *0.25 + 80

Final comments

It can be seen from the test results that the program above works as expected.

The program code is efficient making use of a nested IF-THEN-ELSE statement; it is easy t
indentation and a range of comments.

Test data was chosen logically to ensure that all parts of the code were accessed.

User feedback would be necessary if the program had been created for a real client.

4.1 – Progress Check

6. Hand-trace the following algorithm using the data in the array. (4 marks):

```
Total ← 0
FOR X = 1 TO 5
    Total ← Total + Array[X]
    Average ← Total / X
    Output "X", "Average"
ENDFOR
```

Element	Data
1	9
2	7
3	1
4	5
5	6
6	2
7	6
8	1
9	1
10	4

Pseudocode to program code

Pseudocode is not program-language-specific and so cannot be understood by a programm
code written to aid understanding of a problem.

Well-written pseudocode can be converted into the program language of your choice; for t
produced in any of the following languages: C#, Java, Pascal / Delphi, Python, VB6 or VB.N

It is suggested that pseudocode should include the following to ensure it is straightforwar
programming languages.

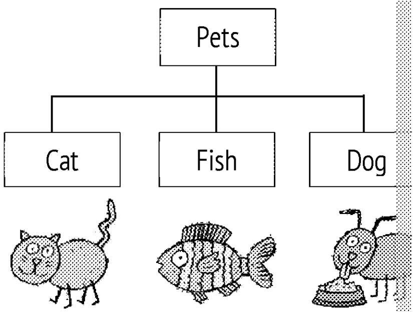
- The use of meaningful variable names
- Naming procedures and functions using understandable titles
- Structure pseudocode making use of white space and indentations to aid understa

Abstraction

Abstraction is the process of including only the important features when solving a problem; this reduces the complexity of the system by removing unnecessary details.

The benefits of using abstraction techniques are that it is easier for the programmer or user to view, to modify and to maintain the solution as they are not distracted by excessive detail which is hidden from them.

The hierarchy diagram below is an **abstrac**
by generalisation for pets



Information hiding

ⓘ **Information hiding** is the principle that the details of the implementation of a class o accessible by the user; the user simply needs the essential details of how to initialise a

Example – car manufacturers break the process down into a series of modules. Some elect model as well as the luxury model. Teams work on particular modules and the detail of th is hidden from them.

So the type of entertainment system installed (basic or luxury) is hidden from the team re (speakers). This approach creates flexibility where the car manufacturer is able to use man cars it produces.

Procedural abstraction

ⓘ **Procedural abstraction** is based on programming where large programs are written by subprograms; this approach applies to any programming language although the units a methods in Java and functions in C and many other languages.

The procedure is a named block of code, where the actual data and values used in the com use of local variables declared within a procedure helps to ensure that the block of code ca be used by the operator without an understanding of its process.

Note that the actual computational method used is not hidden with procedural abstraction aid understanding.

Functional abstraction

ⓘ In **function abstraction** the exact computational method used is hidden, unlike proced

The use of built-in or library functions is an example of functional abstraction, the user sim returned with no knowledge of the internal code within the function.

Example – using a built-in square root function: **SQRT(16)** will return the value 4, so **x =** S

Data abstraction

ⓘ **Data abstraction** is where the details of how a compound data object is constructed a used.

Therefore, the primitive data objects that make up a user-defined data structure are hidde user-defined data structures, such as Records in Pascal and Structs in Java.

Problem abstraction/reduction

ⓘ **Problem abstraction** is where the details of the problem are successively removed or r represented in a way that is straightforward to solve.

Once the unnecessary details in the problem are removed, then the problem can be more c possible that the problem has already been solved or a similar problem has been solved be simplification approach.

Decomposition

ⓘ **Procedural decomposition** is the process of breaking down a problem into a series of s where each sub-problem achieves an identifiable task. In some cases the sub-problem n further subdivided into smaller identifiable tasks.

Problems that are not decomposed are more difficult to solve since dealing with several st at the same time is more challenging than decomposing the problem and solving one sub-problem at a time.

Composition

ⓘ **Composition** is the opposite of decomposition; it is the process of creating a system by combining the tasks identified in the decomposition abstraction.

The process involves:

- Writing procedures for each of the tasks and sub-tasks identified in the decomposition
- Linking these procedures to create compound procedures
- Creating the necessary data structures to support the compound procedures

Automation

This automation process is based on the following:

- **Creation of algorithms** – which includes the breaking up of the problem into sub-problems and the listing of the steps needed to solve each sub-problem.
- **Implementing the algorithms in program code** – which includes conversion betw pseudocode algorithm and the instructions of the programming language chosen.
- **Implementing the models in data structures** – chosen data structures should be s for specific model; commonly used data structures include arrays, files and record/
- **Executing the code** – once the code has been created it should be executed to en runs and then tested/debugged to ensure it operates as expected.

It is essential that sufficient detail is included from the abstraction process to create a mo that can solve the problem to the required level of accuracy.

4.1 – Progress Check

7. Explain the difference between procedural abstraction and functional abst

4.2 REGULAR LANGUAGES

Finite state machines (FSMs) with and without output

A **finite state machine (FSM)** is an abstract machine that can be in any one of a finite
one state at a time and a transition to a new state is triggered by an event.

FSMs are useful in that they can recognise logical sequences and they are used to model a
lights, combination locks, electronic design automation and lifts.

A finite state machine with no output is called **finite state automata (FSA);** it does no
sequence for the final (or goal) state.

State transition diagrams are used to describe an FSM in a graphical format, where ea
transition is connected to a circle by an arrowed line with a description of the input th

In a finite state machine with no output, the final (or goal) state is indicated by a doub

State transition tables are a method used to record all the states and transitions poss

The state transition diagram for a table lamp push button switch is shown on the right.

Where:

- S_0 = state 0 = Lamp off
- S_1 = state 1 = Lamp illuminated

The user simply presses the switch to turn the light from on to off or vice versa.

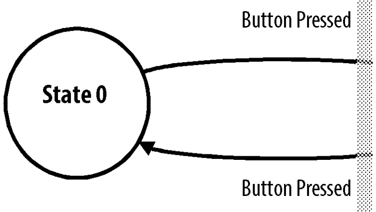


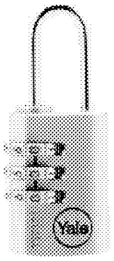
Table lamp push button state

State transition table for a table lamp push button switch is shown on the right. The current state to the next state toggles the lamp between illuminated and off.

Input	Current St
Table lamp switch pressed	Lamp illumi
Table lamp switch pressed	Lamp o

Combination lock – FSA (FSM with no output) example:

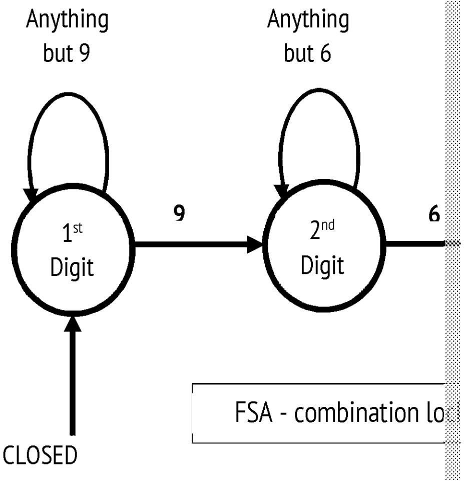
The code for this combination lock is the sequence 963.



The initial state is shown as closed (or locked) and indicated by the lined arrow on the left.

To open (or unlock) the combination, the sequence 9 must be entered into the first digit, then 6 into the second digit and, finally, 3 into the third digit.

The combination lock opens only when all three digits have been correctly entered.



FSA - combination lo

The final or goal state is indi
(OPEN) on the right s

Decision table for combination lock

ⓘ **Decision tables** can be used to model logic sequences in a compact way.

As shown in the state transition diagram above, the combination lock is only open when all three digits have been correctly entered for the combination code 963.

Condition	Conditions		
1st Digit = 9	Y	Y	Y
2nd Digit = 6	Y	Y	N
3rd Digit = 3	Y	N	Y
Final / Goal State	Lock Open		

ⓘ **Mealy machine** is a finite state machine with outputs.

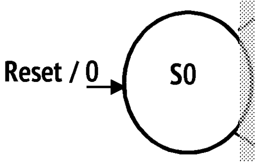
The output from a mealy machine is determined by its present state as well as its present input.

Mealy machines were orginally created to represent electronic circuits and bitwise operations; for example, Mealy machine can use the present state and the present input to obtain an exclusive or (XOR) function.

Mealy machines have a range of uses, such as traffic light control and vending machines.

The example shown on the right is used to detect the sequences (0,1) and (1,0).

The state transition table for the sequence detector is shown below.



Reset	Input	Current State	Next St
1	-	-	S0
0	0	S0	S1
0	1	S0	S2
0	0	S1	S1
0	1	S1	S2
0	0	S2	S1
0	1	S2	S2

4.2 – Progress Check

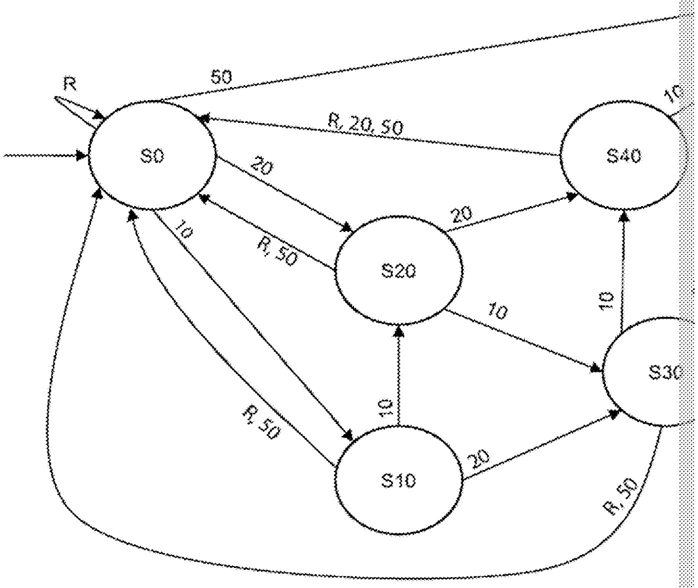
8. Explain the terms 'finite state machine' and 'state transition diagrams'. (

9. The state transition diagram of a finite state machine (FSM) used to con

   The vending machine dispenses a drink when a customer has inserted e

   A transaction is cancelled and coins are returned to the customer if mor button (R) is pressed.

   The vending machine accepts 10, 20 and 50 pence coins. Only one type The only acceptable inputs for the FSM are 10, 20, 50 and R.



(a) Complete the state transition table for this finite state machine. (3

Original state	Input	New state
S0	10	
S0	20	
S0	50	
S0	R	

(b) There are different ways that a customer can provide exactly three machine dispensing a drink. Three possible permutations are '20, 1 four other possible permutations of exactly three inputs that will b (4 marks)

Original state	Input	New state
S0	10	S10
S0		
S0		
S0		

Maths for regular expressions

A **set** is a collection of elements such as objects or numbers; each element is unique w

Sets use the notation shown in the example for A and B:

Commonly used sets include: Natural Numbers (N)
and the set of Integers (Z).

Set comprehension can be used to define a set of values for a large or complex set.
In the set A example shown:

- the symbol **|** means 'such that'
- **x** is used to represent the values of the set listed after the **|** symbol
- N means 'is a member of all the natural numbers'
- ∧ x ⩾ 1 means 'and where x is greater or equal to 1'

An empty set is a set with no elements; it can be represented as:

Sets can be represented in a compact format.
For example, $\{0^n1^n \mid n \geqslant 1\}$ will contain all the strings with an equal number of 0s and

$\{0^n1^n \mid n \geqslant 1\}$ = {01, 0011, 000111, 00001111, 0000011

Finite and infinite sets

A **finite set** is a set with a finite number of elements; the number of elements of a fin

For example {10,20,30,40,50,60} is a finite set with six elements.

Cardinality is the number of elements in a set; the example set above has a cardinalit
an empty set has a cardinality of zero as it doesn't have any elements.

An **infinite set** is a set that is not finite; examples include the set of integers (ℤ), the s
set of real numbers (ℝ). Infinite sets may be countable or uncountable.

An **infinite countable set** contains an infinite number of elements, which **can** be mapp
numbers. Natural numbers and integers are infinite uncountable sets.

An **infinite uncountable set** contains an infinite number of elements, which **cannot** be
natural numbers. The set of real numbers (ℝ) is an infinite uncountable set.

Cartesian product of two sets

The Cartesian product of two sets X and Y can be written as 'X x Y' or 'X cross Y'; it is the co
two sets, in which the first element of each pair is chosen from one set and second elemen

For example, the cross product of set X = {a, b, c} and set Y = {1, 2, 3} is:

X x Y = {(a1), (a2), (a3), (b1), (b2), (b3), (c1), (c2), (c

Subsets

A **subset** is a set where all the elements of one set are elements of another set.

A **proper subset** is a set that does not include all the elements of the set to which it

Subset example
Set X = {1, 2, 4, 6}
and
Set Y = {4, 6, 1, 2}
	X is a subset of Y, because all of the elements of set X
	This can be written as: **X ⊆ Y**
	The term X ⊆ Y means subset X has fewer element

Proper subset example
Set X = {1, 2, 4, 6}
and
Set Y = {4, 6, 1, 2, 9, 13}
	X is a proper subset of Y as there is at least one addit in X.
	This can be written as: **X ⊂ Y**
	The term X ⊂ Y means a proper subset, as set X has

Set operations

The following set operations can be used to define the relationship between two or more

Membership is a set operation used to check whether an element is a member of a

Union of two sets is the set that contains all the elements of these sets.

Example: Find the union of sets X and Y.

Set X = {1, 2, 4, 6} and Set Y = {1, 4, 8, 9, 13}
X ∪ Y is equal to all the elements in sets X and Y
Therefore **{1, 2, 4, 6} ∪ {1, 4, 8, 9, 13} = {1, 2, 4, 6, 8, 9, 13}**

Intersection of two sets is the set that contains only the elements which are in both

Example: Find the intersection of sets X and Y.

Set X = {1, 2, 4, 6} and Set Y = {1, 4, 8, 9, 13}
X ∩ Y is only equal to the elements in both sets X and Y.
Therefore **{1, 2, 4, 6} ∩ {1, 4, 8, 9, 13} = {1, 4}**

Difference of two sets X - Y is the set that contains all the elements of X that are n

Example: Find the difference of sets X - Y.

Set X = {1, 2, 4, 6} and Set Y = {1, 4, 8, 9, 13}
X - Y is only equal to the elements that are in set X and not in set Y.
Therefore **{1, 2, 4, 6} - {1, 4, 8, 9, 13} = {2, 6}**

4.2 – Progress Check

10. Compare the terms 'finite set' and 'infinite set'. (4 marks)
11. Set X = {2, 4, 6, 7} and set Y = {5, 8, 9, 21}. Calculate the following:
    (a) Union of sets X and Y (1 mark)
    (b) Intersection of sets X and Y (1 mark)
    (c) Difference of sets X and Y (1 mark)

Regular expressions

A **regular expression** is a notation used to describe a pattern of characters in an objec

A key requirement in computing is to be able to use regular expressions to match characte replace text. In word processing the sets that are used are the alphabet = {a,b,c,d,e,.....} an

A typical example could be to check that an email address is valid when a user types it int can be validated to make sure it has the required characters and format. A valid example is address could be **dave!@xyz.com**.

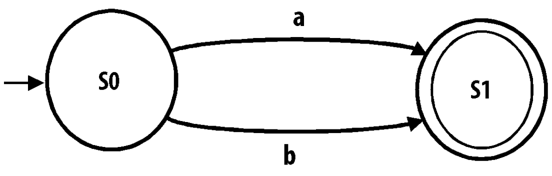
The regular expression **a(a|b)*** generates the set of strings **{a, aa, ab, aaa, aab, aba, ...}**. No regular expression can be listed, as there is an infinite number.

Regular expressions can be written in a convenient shorthand form; some of the notation p containing just two elements {a,b} is shown in the table below.

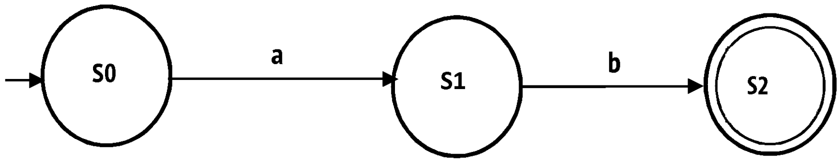
Notation	Regular Expression Explanation
a	Pattern matches the string consisting of only the symbol “a”
b	Pattern matches the string consisting of only the symbol “b”
ab	Concatenation where the pattern matches the string consisting of sym
a+	+ means pattern matches one or more “a”s, giving {a, aa, aaa, ...}
a*	* means pattern matches zero or more “a”s. giving {0, a, aa, aaa, ...}
ab?c	? means matches the preceding character one or zero times, giving {ac
a|b	| means pattern matches symbol “a” **or** the symbol “b”

Note that when interpreting a regular expression, there is a series of rules to follow:

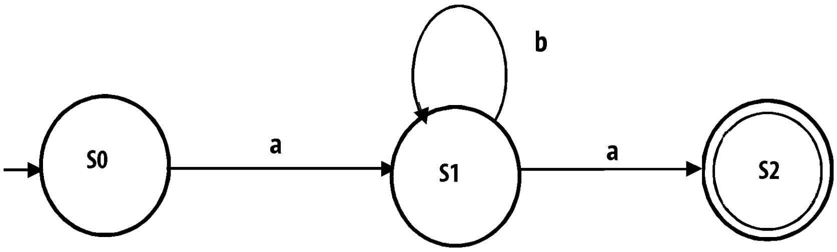
- Operations **+** and ***** are highest priority and always carried out first.
- Concatenation is carried out next.
- Operation **|** (or) is carried out last.



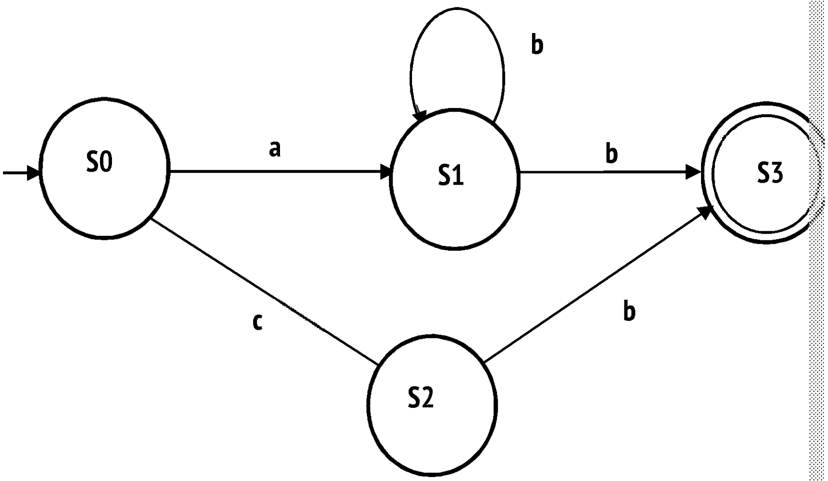
FSM representation regular expression “**a | b**”



FSM representation regular expression of concatenation “**ab**”



FSM representation regular expression of concatenation “**ab*a**”



FSM representation regular expression of concatenation “**ab+ | cb**”

Application of regular expressions

Regular expressions can be used in a wide range of string searching applications, includin

- Searching and locating files and folders
- Find or Search & Replace text strings in a block of text
- Parsing or analysing syntax in computer languages
- Searching for identifiers in a computer program
- Validation of data entry fields in a database or in an online form
- Pattern matching with commands in an operating system

The following terms are defined for the purpose of string searching:

- Literal – any character that is used in a search expression
- Metacharacter – these are special characters that have a unique meaning such as: **^**
- Escape character – using the **** (backslash) character in front of a metacharacter allo as a literal, for example: **\^**
- Search expression – the regular expression that is used to find the target string

Standard searching expressions

Example	Definition and explanation
h(a|o)t	This regular expression will find instances of hot and hat. The pattern will also be matched in hotel and hate.
in	This simple expression will find any sequence that contains the stri with Window and Finish.
\*	The "\" (backslash) character is used so that the * symbol can be us The pattern will be matched with a * b = c.
.ick	The "." dot acts as a wild card to match any literal character. In this case, the pattern will be matched with pick, tick, sick.
[1]	The square brackets are used to match a single character held withi
[0-9]	The square brackets contain the range of numbers {0,1,2,3,4,5,6,7,8 number in that range. If there is no match for 0, the search will continue to look for 1, the
[a-z]	The square brackets contain the range of lower-case letters, allowin in that range.
[A-Z]	The square brackets contain the range of upper-case letters, allowin in that range.
[^h]at	The circumflex symbol "^" means to match any character except th brackets. In this case. the pattern will be matched with cat, mat, sat, but NOT

Regular language

A regular language is a formal language that can be represented by regular expressions; it use by a finite state machine (FSM).

Note that a language is not regular if it is constructed from an infinite, rather than a finite, are useful in parsing, searching and programming language design.

4.2 – Progress Check

12. List the strings generated by the following regular expressions:
    (a) x+ (1 mark)
    (b) x* (1 mark)
    (c) xxy? (1 mark)

13. Draw the FSM representation of the regular expression a(bc)* (3 mar

14. Compare the terms 'metacharacter' and 'escape character' used for th (2 marks)

15. Define the term 'regular language'. (2 marks)

4.3 CONTEXT-FREE LANGUAGES

Backus–Naur (BNF) syntax diagrams

Context-free language is a formal system that is used to describe a language whe regular expressions, context-free languages have an infinite range of components; syntactic analysis where text or strings are broken down into their component part

Backus–Naur Form (BNF) is a notation that is used to describe the syntax or gram language. BNF functions use recursive techniques as there is no method available

BNF includes the following notation elements shown in the table below.

Notation	Definition and explanation
< >	Used to enclose a syntactic element
::=	Used to indicate the definition of a syntactic element
|	Symbol indicates an OR choice between two syntactic element
[ ]	Used to enclose an optional part of the rule
{ }	Used to enclose a content that can be repeated any number of

BNF examples

Integer is defined as a recursive function, meaning an integer can be defined as a digit OR

<integer> ::= <digit> | <integer> <digit>

Digit is defined as an integer between 0 and 9; there are no further syntactic elements ne

<digit> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

Conditional statement

if statement is defined as shown below.

```
    if a condition is satisfied a further statement is executed
        or
    if a condition is satisfied a further statement is executed
    else an alternative statement is executed

    <if statement> ::=
            if ( <condition> ) <statement>
       |    if ( <condition> ) <statement>
            else <statement>
```

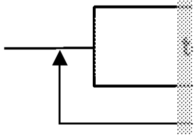
The **if** statement can also be defined in the compressed format below using the optional p

<if statement> ::=

if (<condition>) <statement> [else <statement>]

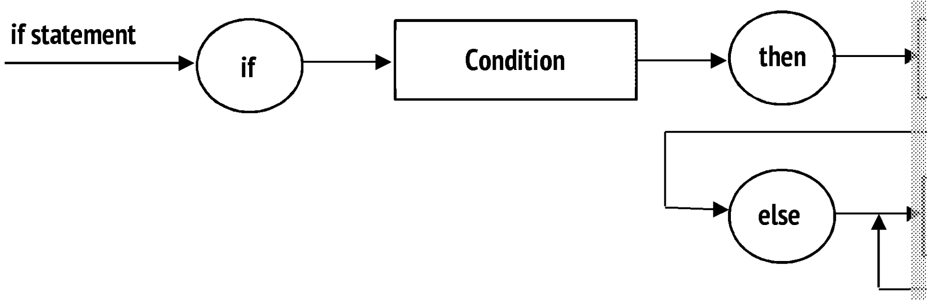
Syntax diagrams

Syntax diagrams are used to give a representation of context-free language; they can be u an alternative graphical method. The following symbols are used in syntax diagrams:

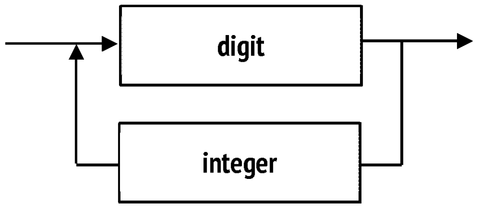
Definition and explanation	Sy
This symbol represents a terminal element, so no further syntactic elements are required.	
This symbol represents a non-terminal element, so will have a further syntactic element to add further detail.	
This symbol represents a non-terminal element that may be repeated more than once.	

Syntax diagram examples

If statement



Integer



4.3 – Progress Check

16. Define the term ‘context-free language’, giving an example of where it
17. Explain the use of Backus–Naur form notation and the use syntax diag

4.4 CLASSIFICATION OF ALGORITHMS

Comparing algorithms

The efficiency of two algorithms that solve the same problem can be compared against th which is the more efficient:

- Time complexity – the amount of time taken to process the task
- Space complexity – the amount of memory or space that is used to process the task

Maths for understanding Big-O notation

Big-O notation is used to determine the time complexity or run-time of the code as the vol worst case or maximum amount of time an algorithm would take to process.

The input size (N) of the algorithm is used in Big-O notation and it is the size of the data o

The specification requires an understanding of: constant, linear, polynomial, exponential a

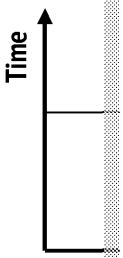
Constant complexity – O(1)

Constant complexity is where the algorithm will always be completed in the same time irrespective of the input size.

Popping an item from a stack or pushing an item onto a stack takes the same time whatever the size (N) of the stack.

In both cases, the time taken is a constant for all values of N.

N	10	100	1000	10,000	10,000
Constant	1	1	1	1	1



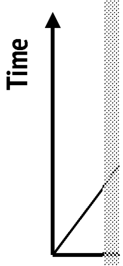
Linear complexity – O(N)

Linear complexity is where the time taken for the algorithm to process will be directly proportional to the input size.

A linear complexity example is where an algorithm sums the elements of an array; the time taken is directly proportional to the number of elements in the array.

An algorithm may have a linear relationship with the input, but for each element in the input, the algorithm has to run three operations. This means that run-time is proportional to three times the input size, as seen in the table below.

N	10	100	1000	10,000	100,000
O(3N)	30	300	3000	30,000	300,000

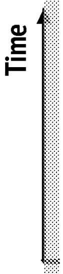


Polynomial complexity – O(N^k) where (k >= 0)

Polynomial complexity is where the time taken for the algorithm to process will be directly proportional to the square of the input size.

A selection sorting algorithms such as bubble sort and insertion are examples of polynomial time algorithms using O(N^2).

N	10	100	1000	10,000	100,000
O(N^2)	100	10,000	10^6	10^8	10^{10}



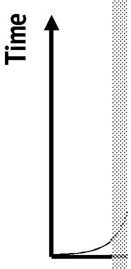
Exponential complexity – O(k^N) where (k > 1)

The graph of the exponential function O(2^N) indicates that the time taken for the algorithm to process will double for each unit increase in the input size.

Algorithms based on exponential growth are termed 'intractable' as they cannot be solved in reasonable amount of time.

The table below gives an indication of the rapid growth, so where an algorithm has a **100** steps as an input size, the time taken for it to process would be approximately **10^{30}** steps which is well in excess of many trillions of steps.

N	1	10	100	1000
O(2^N)	2	1024	≈10^{30}	≈10^{301}



Logarithmic complexity – O(Log(N))

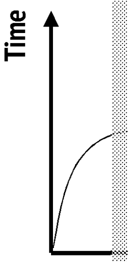
Logarithmic functions are based on the inverse of an exponential function; in computer science log base 2 is normally used.
The exponential function **2^3 = 8**

So the inverse of the exponential function is **$\log_2 8$ = 3**

Some values of **O($\log_2$N)** are shown in the table below.

N	8	128	1024	8192	131,072
O($\log_2$N)	3	7	10	13	17

Logarithmic complexity is useful since it scales up well as the input size increases. So, from the table it be seen that when the value of N increases, the execution time increases at a slower rate; therefore, algorithms with logarithmic complexity are time-efficient.



Order of complexity

Algorithm complexity

Big-O notation is used to determine the most efficient algorithm to solve a particular prob scalability of a solution, or the efficiency of the solution as the input size rises; see table b

Notation	Name	Derived complexity of algorithms
O(1)	Constant complexity	Most scalable algorithm as it always ta used to retrieve data from a look-up ta
O(LogN)	Logarithmic complexity	Considered the second most efficient m binary search based on a sorted array
O(N)	Linear complexity	The next most efficient method; can be where the data is unsorted
O(N LogN)	Linear/logarithmic complexity	This complexity seen in a merge sort c case of quicksort
O(N^k)	Polynomial complexity	An inefficient method where algorithm intractable; can be seen in a selection
O(k^N)	Exponential complexity	The least efficient method and conside however, can be used to solve the 'tra and integer factorisation

Limits of computation

In some cases, there are practical limits on what can be computed. The limiting factors are computer hardware used to solve the problem.

Exponential algorithms are inefficient methods; due to their high time complexity, these p acceptable time frame.

The computer hardware used to run the algorithm can also impact on the time frame need power can be limited by processor speed and lack of memory.

4.4 – Progress Check

18. Compare the Big-O notation terms 'constant complexity' and 'linear com
19. Rearrange the following list in order of complexity. (2 marks)

$O(N^k)$	Polynomial complexity
O(1)	Constant complexity
O(N)	Linear complexity
$O(k^N)$	Exponential complexity
O(LogN)	Logarithmic complexity

Classification of algorithmic problems

Algorithmic problems can be broadly classified as tractable or intractable.

Tractable problems can be solved in a reasonable time frame; they have a polynomi solved practically using a computer.

Typical tractable problems include: searching an ordered or unordered list and sorting a li

Intractable problems cannot be solved in a reasonable time frame; they do not have and so cannot be solved practically using a computer.

It is sometimes possible for intractable problems to have a working solution where the inp increases the time frame increases exponentially; therefore, a computer solution would no time frame.

Heuristics is a technique designed to solve a problem based on experience or an 'ed exact solution but heuristic methods are often used to tackle intractable problems.

Heuristic techniques are in direct contrast to the algorithmic approach as:

- the use of a well-designed algorithm will give a predictable answer for a particular ta intractable and so it is not possible to design an efficient algorithm capable of solving
- heuristics are useful as an alternative technique in solving difficult problems; using th optimal results; however, heuristic techniques can produce results that are satisfactor

Computable and non-computable problems

Computable problems are problems that have an effective procedure or algorithm to solve set of instructions is executed to determine the correct output for a given input.

Non-computable problems are problems that have no effective procedure or algorithm to unsolvable irrespective of the amount of computer power available or the time allocated t

Halting problem

The **halting problem** is used to determine whether it is possible to create a program (H) to given an associated input (I) will halt when executed with that input or will continue to ru

The Turing model of computation is used for this test which is based on there being no res limit on the execution time.

The significance of the halting problem is that Turing proved that it was not possible to cre problem; so, the halting problem was undecidable. Turing's proof demonstrated that there cannot be solved by a computer.

4.4 – Progress Check

20. Compare tractable and intractable problem classifications. (4 marks)
21. Compare computable and non-computable problems. (4 marks)
22. Explain the significance of the halting problem. (2 marks)

4.5 A MODEL OF COMPUTATION

Turing machine

A **Turing machine** (TM) is a machine that controls an infinitely long read/write (R/W) tape and an infinitely long storage device.

Turing machines can be viewed as computers that carry out a single fixed program based o

The tape is divided into a series of square or cells, where each cell holds one symbol from example below, where ■ indicates a blank symbol.

The tape shows a valid string between the two delimiters, where the symbol # is used.

#	0	1	1	■	0	0	1	1

The tape head reads one cell at time where the contents of that cell and the current state o determine the next move of the Turing machine. The TM device contains the following fea

- The tape has definite start point (after the delimiter #) at the left end and an infinite the right end.
- There is a finite set of symbols used in a Turing machine.
- The tape head can move left or right one cell at a time and can read or write the sele
- A transition table can be used to show the particular processes performed by a Turin

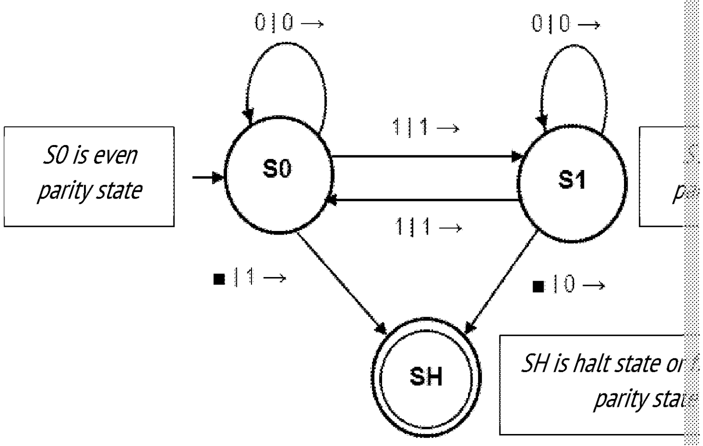
ⓘ The **start** state is the initial state of a Turing machine.

ⓘ A **halting** state is the point at which a Turing machine stops processing.

ⓘ A t
in
be
ta

State transition diagram symbol

Symbol	Alternative	Description
|	/	The vertical bar or 'slash' separates input from output
■	b	The square symbol or 'b' represents a blank
#		Delimiter symbol
→	R	Move the tape head one square to the right
←	L	Move the tape head one square to the left



State transition diagram for odd parity generator is sho

Turing machine odd parity generator

The Turing machine reads the 0s and 1s from the tape.

It is set up to ensure that number of 1s, including the parity bit, is an odd number.

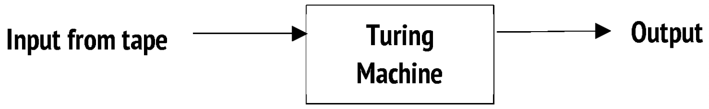
- When the number of 1s on the tape is odd, the parity bit is set to 0.
- When the number of 1s on the tape is even, the parity bit is set to 1.

The state transition diagram above performs the odd parity function generator; the rules o

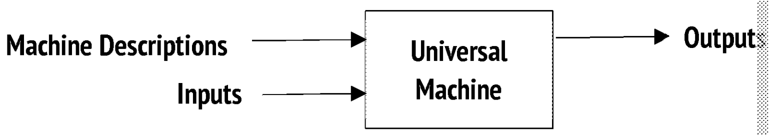
Transition rule	Explanation
(S0, ■) = SH, 1 ⟶)	If blank is read at S0, output 1, move to S
(S0, 0) = S0, 0 ⟶)	If 0 is read at S0, output 0 and remain at S
(S0, 1) = S1, 1 ⟶)	If 1 is read at S0, output 1 and move to S
(S1, ■) = SH, 0 ⟶)	If blank is read at S1, output 0, move to S
(S1, 0) = S1, 0 ⟶)	If 0 is read at S1, output 0 and remain at S
(S1, 1) = S0, 1⟶)	If 1 is read at S1, output 1 and move to S

A **universal machine** is a machine that simulates a Turing machine by reading a descriptio and the inputs associated with it.

A Turing machine simply performs one function and requires an individual tape with input limitation for complex programs with a combination of processes, as every process used re its own tape.



A universal machine combines a range of individual machines, allowing the option to perf



4.5 – Progress Check

23. Define the term 'Turing machine' (TM) and list the features a Turing ma
24. Explain how Turing machines make use of transition functions. (2 mark
25. Define the term 'universal machine'. Explain how it improves the efficie a basic Turing machine. (6 marks)

ANSWERS

Topic 1 – Programming

1.1 (a) Real/Float – contains a decimal point; for example 65.25 (1). The position of
hence the term 'float' (1).

(b) Character – a character represents a single alphanumeric item of data, for ex
number, letter or other character (1).

(c) Array – array data structures are made up of list or date elements that are th
example: Char Array[6] = {'C', 'O', 'D', 'I', 'N', 'G'} (1).

1.2 (a) A subroutine is a set of instructions (1) to perform a certain task (1). It can be
many times within a computer program (1).

(b) A procedure is a subroutine (1) that is called to perform a task (1). It may or n

(c) A function is a series of instructions (1) to perform a task (1). When called, it
returns a value (1).

(d) Iteration or repetition is where a program executes a statement or statemen
some logical condition is satisfied (1).

(e) A selection structure is where the program executes different actions (1) or s
result of a comparison (1).

1.3. Definite iteration is where the number of iterations is known before the start of th
definite iteration example is where a loop is set up to input n values and prints th
where the number of iterations is not known at the start of the loop (1), but is det
the loop (1).

1.4 (a) Rounding replaces a number with an approximate value using fewer digits (1
which is rounded to two decimal places (1). Truncation limits the number of d
point (1); 34.5674 can be truncated to 34.56 or two decimal places (1).

(b) 11 MOD 3 = 2 (1 mark)

(c) 3**2 = 9 (1 mark)

1.5 Truth table for the exclusive or function using variables A and B (1 mark):

A	B	A XOR B
0	0	0
1	0	1
0	1	1
1	1	0

1.6 Constants are used where data that is used in a computer program is preset (1), w
computer programs to store data that may change when the program is executed

1.7 The concatenation operation is used to join two strings together where the comb
"Computer" + "Science"; returns 'Computer Science' (1).

1.8 Where local variables are declared and used in a subroutine they are only in exist
executed and are only accessible or in scope within that subroutine (1). It is good p
than global variables in subroutines and functions as it is easier to trace the conte
program and the subroutine retains modularity where only parameters and not gl
execution (1).

1.9 Recursion is where a subroutine or function is defined in terms of itself (1). The fu
once (1) as it calls itself during execution (1); the function is terminated when a log

1.10 The recursive solution uses fewer variables than the iterative approach (1), but ca
which makes it more difficult to maintain (1) and can take more time to execute th

1.11 The advantages of using a structured approach to programming are:

- Breaking down large programming tasks into manageable subtasks means th several programmers, which saves development time (1).
- Program test and debug time is reduced where modularity helps to reduce t correcting the errors that may occur (1).
- Programs are easier to understand and, therefore, easier to maintain; also, a introduction of an additional module (1).

1.12 Four advantages are:

- New functionality can be added to a program (1) by simply creating a new cla
- Class is only concerned with the data defined within it (1), so it won't access an
- Data can be hidden within the class that accesses it (1), providing greater sys
- Objects can be set up to inherit attributes and methods to create reusable co were developed for as well as in other object-oriented programs (1).

1.13 In object-oriented programming, polymorphism allows objects in a class or subcla making use of the same method (1).

An example of polymorphism could be: where shape is the original class, a metho subclass objects such as rectangle and circle (1); but the programmer will define th subclass, such as length × breadth for rectangle and $\pi \times \text{radius}^2$ for circle (1).

1.14 1 mark for each correct definition

(a) Class describes shared attributes and methods and is a template that can be

(b) Object is one instance of a class, such as a real-world entity.

(c) Inheritance is where a new class that is created (termed subclass) retains the original class it was based on (termed superclass).

(d) Attributes are the properties or variables within a class.

(e) Encapsulation is the combining of attributes, methods and data in one objec

Topic 2 – Data structures

2.1 A data structure is the format used to efficiently store (1) and organise a collectio

2.2 Array data structures are made up of a list of date elements that are the same dat

2.3 A one-dimensional array is a list of data elements (1), whereas a two-dimensional a elements (1).

2.4 Text files store ACSII data and so they are humanly readable using a text editor (1) records data that contains unprintable characters so cannot be read using a text e

2.5 Static data structure is where the data and memory allocated is fixed (1); an advan addresses will be allocated at compile time, and so will be fixed and contiguous, p data structure is where memory allocation is varied to exactly meet the need of th efficient use of memory (1).

2.6 (a) A linear queue is a FIFO data structure organised as list of data (1).

(b) A circular queue is a FIFO data structure organised in a ring where the start a the last element to the first element of the array (1).

(c) A priority queue is similar to a regular queue except that each element is ass priority elements are served first. Where elements have the same priority, th order (1).

2.7 Final content of stack: 7, 12, 11 (1).

2.8 A directed graph is a graph where the direction of travel between vertices (1) can arrow (1), whereas an undirected graph is a graph where the direction of travel be so for vertices A and B the travel direction can be either A to B or from B to A (1).

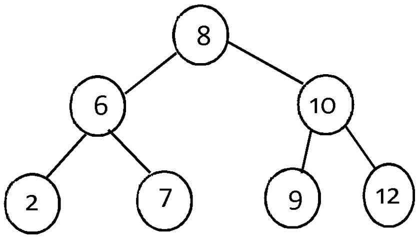
2.9 An adjacency list is used to represent a graph by listing each node in a graph (1) ar
(1), whereas an adjacency matrix is used to represent a graph by listing each node
edge between pairs of nodes (1).

2.10 A tree is an abstract data type based on an undirected graph that doesn't contain
node can only be reached through one path (1). The following terms apply to tre

- Root – this is the starting node that has no parents, and all other nodes bran
- Parent – this node in the tree has further nodes branching from it, termed ch
- Child – this node has nodes above it in the structure, termed parents (1).
- Leaf – this node does not have any further nodes beneath it (1).

2.11 A binary tree is a tree structure where each node has a maximum of two child no
since there is no relationship between a parent node and its child nodes (1), wher
ordered or sorted binary tree (1) with the recursive relationship that for each nod
and the right child is more than the node (1).

2.12 A binary search tree based on the following sequence of numbers: 8, 6, 10, 2, 7,



2.13 A hash table (or hash map) is a data structure that stores data in an associative wa
hash function which maps a key to an index in an array to find the value of an arra

A hash function takes a data input known as a key and outputs an integer known a
the key to a particular index in the hash table (1).

Collision is where two keys are applied to the hash function (1) and create an iden
chosen with the aim of minimising the chance of collisions (1).

2.14 Linear probing is where a key is assigned to the next available slot in the hash tab
problem with this technique is that it can create clustering, which is where more c
function index is not randomly distributed (1).

Separate chaining is where the hash table index is a pointer to a linked list data st
Where there are no collisions, the linked list contains one element and the linked
for a particular slot (1).

2.15 Dictionary is a data structure that contains a collection of 'key-value' data element
the associated key (1). One use of dictionaries is in information retrieval based on a

2.16 Vectors in programming data structures are similar to arrays; however, the size o
can automatically grow to store additional elements (1). The individual elements c
via an index (1).

2.17 Vector addition:

(a) A = (4, 13) and B = (5, 2)
A + B = (9, 15) (1)

(b) A = (19, 12, 14) and B = (13, 16, 5)
A + B = (32, 28, 19) (1)

2.18 The magnitude of vector B when vector A = (5, 12) is multiplied by 4, so B = (20,

2.19 The dot product of vectors A(7,4) and B(9,2):
A · B = ((7*9), (4*2)) = (63,8) (1 mark)

Topic 3 – Fundamentals of Algorithms

3.1 Depth-first graph traversal starts at the chosen node (1) and explores each branch before backtracking to the remaining unvisited nodes (1). Breadth-first graph trav first and then continues to visit the child nodes, and so on (1). A queue is used to visited (1).

3.2 Pre-order tree traversal sequence: 44,10, 34,18, 36, 97, 92, 51 (2).
Post-order tree traversal sequence: 18, 36, 34, 10, 51, 92, 97, 44 (2).

3.3 Expressions in infix

(a) 7 / 4 (1)

(b) (7 + 3) * 2 (1)

3.4 Expressions in postfix

(a) 7 3 - 4 / (1)

(b) 3 4 * 6 2 / + (1)

3.5 A search algorithm is used to find an item with specific properties among a group algorithm is used to put elements from a list into a specific order (1).

3.6 A linear search is the simplest searching technique in the code; the 'target' is the to compare the target with each element of the list until it has been found (1).

3.7 In a linear search, each element in the list is examined until the target value is fou for a large array (1). In a binary search, the number of elements being examined is program; for example, a maximum of only six comparisons is needed to find a tar

3.8 Trace the stages of a binary search for the number 18 from the list {1,4,6,7,9,11,16,

1	2	3	4	5	6	7	8	9	10	Index
										Comments
1	4	6	7	9	11	16	18	21	67	Data set number
1	4	6	7	9	11	16	18	21	1	INT(1+10)/2 = Inde 9 < 18 so choose ri
1	4	6	7	9	11	16	18	21	1	INT(1+5)/2 = Index Number 18 found

3.9 An advantage of the binary search is that it is more efficient than the linear search, comparisons (1), and a disadvantage is that the data that has to be searched needs t

3.10 A binary search tree is an ordered or sorted binary tree (1) with the recursive rela

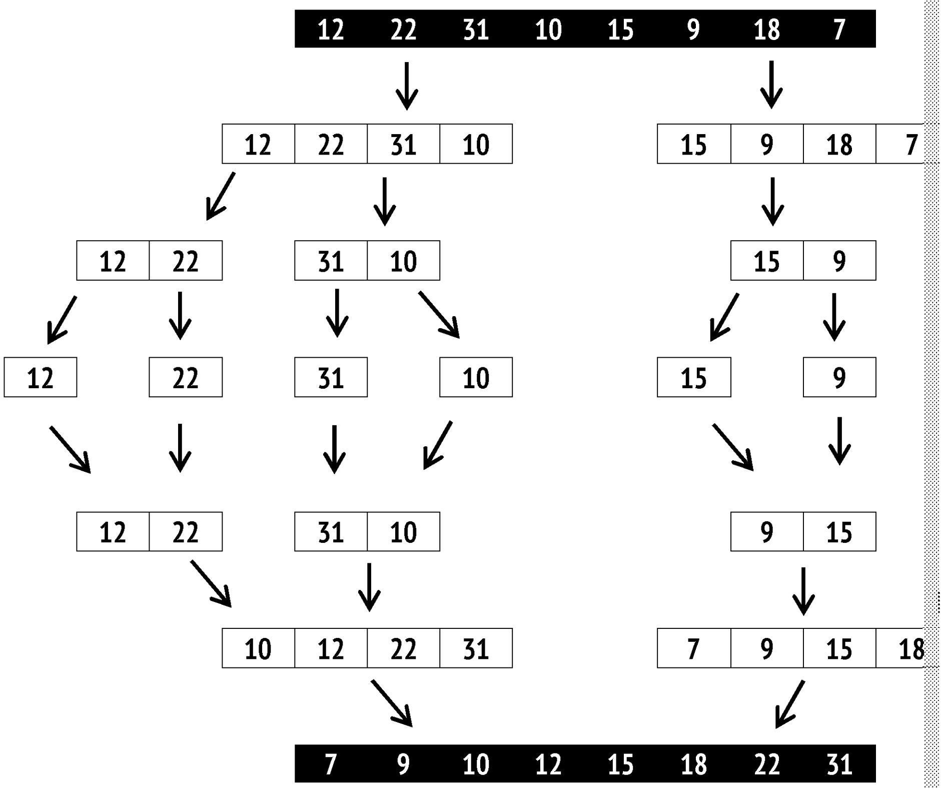
- the left child is less than the node (1)
- and the right child is more than the node. (1)

3.11 Bubble sort

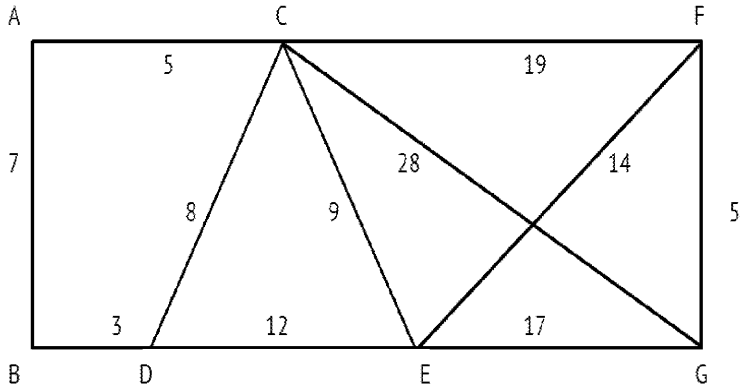
(a) A bubble sort is a sort where adjacent items in the array or list are scanned necessary, until one full scan performs no swaps (1).

(b) The main disadvantage of the bubble sort is that it can take a maximum of (size of the list that needs to be sorted (1); this is because an out-of-position i position per scan (1).

3.12 Merge sort with eight elements (12, 22, 31, 10, 15, 9, 18, 7) (4 marks):



3.13 Use Dijkstra's algorithm to work out the shortest distance between B and F:



- B starts as current node; update A, D and C
- D becomes current node; update C and E
- C becomes current node; update G and F (don't update E, as it is a longer ro
- E becomes current node; update G (don't update F, as it is a longer route)
- F becomes current node; (don't update G, as it is a longer route)

Shortest route is 29 from **B > D > E > F** (1 mark for each of the nodes identified).

Node	Shortest distance from B	Via
A	~~∞, 7~~	B
B	~~0~~	START
C	~~∞, 12, 11~~	~~A~~ D
D	~~∞,~~ 3	B
E	~~∞,~~15	D
F	~~∞, 30,~~ 29	C E END
G	~~∞, 39~~	C E

Topic 4 – Theory of computation

4.1 Problem-solving involves reaching a desired outcome (1) from an initial situation (

4.2 Top-down design or stepwise refinement is used to plan a solution based on a top a complex problem can be solved by breaking it down into a series of small steps (down further into even smaller steps (1). The smaller the steps, the easier it is to b the sub-problems (1) and eventually the whole problem.

4.3 The advantages of using top-down design are:
(a) Breaking the problem into parts helps to clarify exactly what needs to be ach
(b) Each stage of the refinement process creates smaller sub-problems that are e
(c) Some functions or parts of the solution might be reusable (1).
(d) Breaking design into parts allows more than one person to work on the solut

4.4 An algorithm is a step-by-step approach to solving a problem (1). It is normally wri independent of any particular programming language (1).

4.5 (a) Algorithm to calculate the area of a square from a value entered by the user (

```
INPUT Length
Area ← Length * Length
OUTPUT "Square"; Area
```

(b) Calculate the area of all the squares with values 2, 3, 4 and 5 (5 marks)

```
FOR Length = 2 TO 5
    Area ← Length * Length
    OUTPUT "Square"; Area
ENDFOR
```

4.6 Hand-tracing (4 marks)

Element	Data	X	Total	Average
1	9	1	9	9
2	7	2	16	8
3	2	3	18	6
4	10	4	28	7
5	2	5	30	6
6	2			
7	6			
8	1			
9	1			
10	4			

4.7 Procedural abstraction is based on programming where large programs are writte subprograms (1). The procedure is a named block of code, where the actual data a method are abstracted (1). The use of local variables declared within a procedure h code can be treated as a 'black box' that can be used by the operator without an u However, the actual computational method used is not hidden with procedural ab 'black box' to aid understanding (1).

In the case of functional abstraction the exact computational method used is hidd The use of built-in or library functions is an example of functional abstraction; the value is returned with no knowledge of the internal code within the function (1).

4.8 A finite state machine (FSM) is an abstract machine that can be in any one of a fin only be in one state at a time and a transition to a new state is triggered by an eve

State transition diagrams are used to describe an FSM in a graphical format (1), whe each transition is connected to a circle by an arrowed line with a description of the i

4.9 (a) Complete the state transition table for this finite state machine (3 marks):

Original state	Input	New state
S0	10	S10
S0	20	S20
S0	50	S50
S0	R	S0

(b) Any four from the following permutations (4 marks)
20,20,10; R,R,50; 10,20,20; 20,50,50; 20,R,50

4.10 A finite set is a set with a finite number of elements; the number of elements of a example {10, 20, 30, 40, 50, 60} is a finite set with six elements (1). An infinite set include the set of integers (Z), the set of natural numbers (N) and the set of real n countable or uncountable (1).

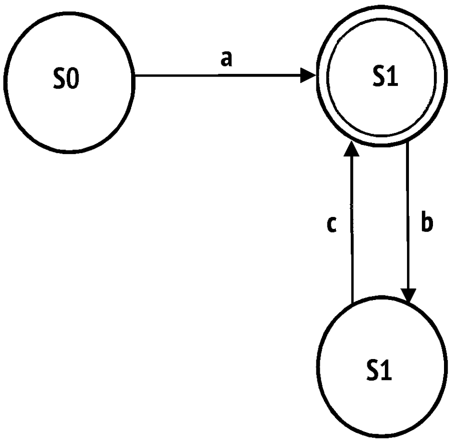
4.11 (a) **Union** X U Y is equal to all the elements in sets X and Y.
Therefore, {2, 4, 6, 7} U {5, 6, 9, 21} = {2, 4, 5, 6, 7, 9, 21} (1 mark).

(b) **Intersection** X ∩ Y is only equal to the elements in both sets X and Y.
Therefore, {2, 4, 6, 7} ∩ {5, 6, 9, 21} = {6} (1 mark).

(c) **Difference** X - Y is only equal to the elements that are in set X and not in set
Therefore, {2, 4, 6, 7} - {5, 6, 9, 21} = {2, 4, 7} (1 mark).

4.12 The strings generated by the following regular expressions
(a) x+ generates {x, xx, xxx, …} (1)
(b) x* generates {0, x, xx, xxx, …} (1)
(c) xxy? generates {xx, xxy}(1)

4.13 FSM representation of the regular expression a(bc)* (1 mark for each correct nod



4.14 Metacharacters are special characters that have a unique meaning such as: ^ or * using the \ (backslash) character in front of a metacharacter, allowing it to be use

4.15 A regular language is a formal language that can be represented by regular expre acceptable for use by a finite state machine (FSM) (1).

4.16 Context-free language is a formal system that is used to describe a language whe regular expressions, context-free languages have an infinite range of components logical syntactic analysis (1) where text or strings are broken down into their com any errors (1).

4.17 Backus-Naur Form (BNF) is a notation that is used to describe the syntax or gram language (1). BNF functions use recursive techniques as there is no method availa (1). Syntax diagrams are used to give a representation of context-free language (1) syntax and rules by an alternative, graphical method (1).

4.18 Constant complexity algorithms execute in the same time irrespective of the inpu array takes a constant time as only one operation needs to be carried out, irrespe

The time taken to process a linear complexity algorithm is directly proportional to to sum the elements of an array where the execution time is directly proportional

4.19 List in order of complexity (2 marks for all correct, 1 mark if three correct):

O(1)	Constant complexity
O(LogN)	Logarithmic complexity
O(N)	Linear complexity
O(N^k)	Polynomial complexity
O(k^N)	Exponential complexity

4.20 Tractable problems can be solved in a reasonable time frame (1); they have a poly can be solved practically using a computer (1).

Intractable problems cannot be solved in a reasonable time frame (1); they do not solution and so cannot be solved practically using a computer (1).

4.21 Computable problems are problems that have an effective procedure or algorithm containing a finite set of instructions is executed to find the correct output for a g

Non-computable problems are problems that have no effective procedure or algori are unsolvable irrespective of the amount of computer power available or the time

4.22 The significance of the halting problem is that Turing proved that it was not possib halting problem (1); so, the halting problem was undecidable. Turing's proof demo problems that simply cannot be solved by a computer (1).

4.23 A Turing machine (TM) is a finite state machine (FSM) that controls an infinitely lo be used as an input device (1) and an infinitely long storage device (1).

The TM device contains the following features:

- The tape has definite start point (after the delimiter #) at the left end and an space at the right end (1).
- There is a finite set of symbols used in a Turing machine (1).
- The tape head can move left or right one cell at a time and can read or write
- A transition table can be used to show the particular processes performed b

4.24 A transition function is a set of rules to indicate how a Turing machine moves betw tape data changes (1).

4.25 A universal machine is a machine that simulates a Turing machine (1) by reading a the machine and the inputs associated with it (1).

A Turing machine simply performs one function and requires an individual tape w This is a limitation for complex programs with a combination of processes, as every individual machine and its own tape (1).

A universal machine combines a range of individual machines (1), allowing the opt complex calculations like a digital computer (1).