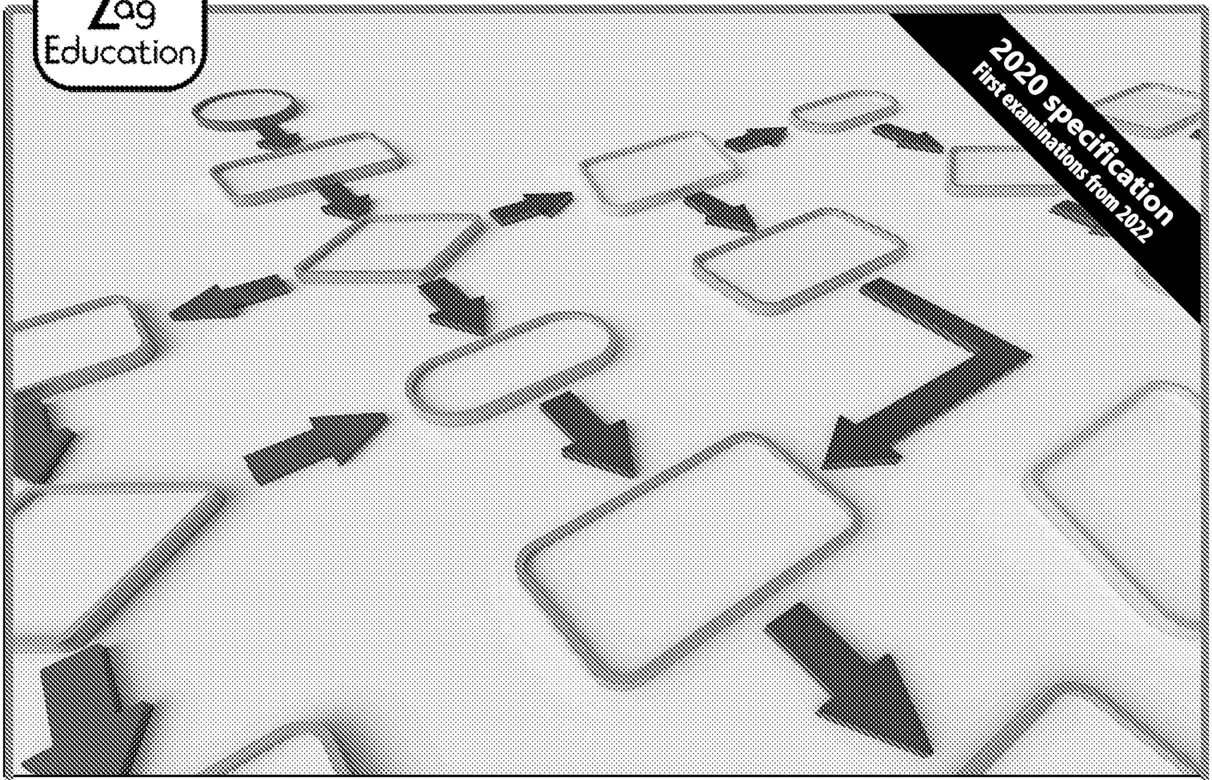
Computer Science GCSE (9–1) | AQA | 8525



Algorithms Resource Pack

for AQA GCSE Computer Science (8525)

Sue Wright

Part 1 – Theory

zigzageducation.co.uk

POD 10602a

Contents

Teacher's Introduction

All examples and vocabulary used follow the AQA Subject Specific Vocabulary definition of key terms used in the AQA GCSE Computer Science (8525) specification.

This resource explores specification section ***3.1 Fundamentals of Algorithms*** and looks at each section in depth identifying key terms and their definitions to help your students understand some of the more difficult concepts.

There are detailed pseudocode examples, explanations and diagrams which explain the stages of the searching and sorting algorithms and how the pseudocode relates to the process of each algorithm.

Students are shown how to plan algorithms using both flowcharts and AQA standard version pseudocode, starting with explanations and examples of how to analyse an algorithm in terms of its inputs, processes and outputs before attempting the algorithm design itself. The resource also includes a range of exercises, as well as crosswords for each section to check students' understanding of the key terms. Solutions to all exercises are included.

The resource is presented in 2 parts:

> **Part 1:** Seven chapters of theory, interspersed with task prompts. Give to students in its entirety or as separate handouts as and when you need them.
>
> **Part 2:** Worksheets (for completion of the tasks referred to in Part 1), plus solutions

This booklet could be used as a stand-alone resource to deliver this important part of the syllabus, as well as to support the delivery of syllabus section ***3.2 Programming***, where much of the content (such as variables, arrays, subroutines and operators) is covered naturally while looking at algorithms.

This resource will be invaluable in giving students a detailed introduction to the use of written pseudocode and code segments that will form part of their written exams.

About the author

Sue Wright has been teaching for over 25 years and has a B.A., B.Ed. and Undergraduate Diploma in Computing from the Dept. for Continuing Education at Oxford University. She has taught A Level Computing, A Level Computer Science and GCSE Computer Science. In her spare time she enjoys writing, playing in her local brass band, reading crime novels and visiting new places.

Sue Wright, September 2020

ALGORITHMS: WHAT ARE T

An algorithm is a **series of instructions that solves a problem in a finite number**

What does that mean? Below is an example of a recipe for making cherry tomat can use to explain an everyday algorithm in terms of its **inputs**, **process** and **outp**

INPUTS

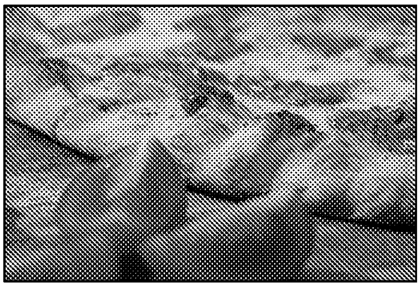
- 280 g packet white bread or pizza base mix
- 1–2 tsp olive oil
- 150 g cherry tomatoes, halved
- 125 g ball mozzarella, drained
- 75 g cubed pancetta or bacon
- bunch of basil

PROCESS

1. Empty the pizza doug the basil and stir half with 200 ml warm wa minutes, then roll out
2. Transfer to a floured with olive oil and leav Push the cherry toma dough so the pizza is
3. Dry the mozzarella ve paper, and then cut i pizza along with the p bake for 20 minutes u golden. Tear over the

OUTPUT

The finished pizza:



Another example uses the 'shortest path algorithm' invented by Dutch computer s

INPUTS

Your location and destination.

PROCESS

Google maps will use data from GPS, its own mapping system and data from other road users to calculate a suitable route.

OUTPUT

The route and the steps your destination are disp on your smartphone.

In both of these examples:

- The steps or instructions must be clear so that they cannot be misunders
- The steps or instructions must follow the correct order, e.g. Step 1 is foll
- They must produce the outputs you want at the end, e.g. the pizza or the location to your destination.
- Each time the instructions are used, the same results must be produced, your destination.

Every **successful algorithm** can be judged using three criteria:

- Accuracy – does it lead to the expected results?
- Consistency – does it produce the same result each time it is run?
- Efficiency – does it solve the problem in the shortest possible time?

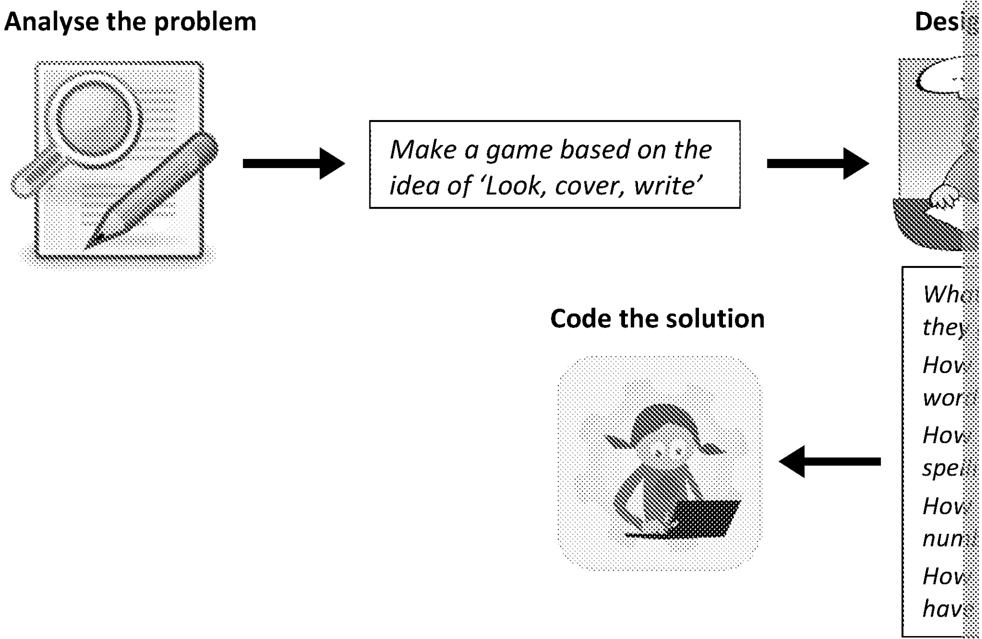
Key Terms

Algorithm	A series of instructions that solves a problem in a finite number
Sequence	An ordered set of steps or instructions.
Unambiguous	Written in a way that makes it completely clear what is meant.

ALGORITHMS VS PROGRAMS

Algorithms and programs are very closely related BUT the important distinction program to solve a problem, you have to work out the solution (the algorithm) fi

Example: You have two young cousins who struggle to learn their spellings each to help them.



Analyse the problem

Make a game based on the idea of 'Look, cover, write'

Desi

Wh
the
Hov
wor
Hov
spel
Hov
num
Hov
hav

Code the solution

There are two ways in which we can plan and design algorithms:

- VISUAL – using flow chart symbols
- TEXT – using a written sequence of instructions

In your exam you will need to be able to use and understand both methods.

Complete Exercise 1: Charity Fundraiser – Analyse the Problem

VARIABLES – WHAT ARE THEY AND WHY DO WE N

In order to process data, all computers need to be able to temporarily store that accessed and changed as the program runs.

For example, in a simple hangman game the computer needs to store and access

- the word to be guessed
- which letters in the word have been guessed correctly
- which letters are incorrect guesses
- which parts of the hangman image have been displayed

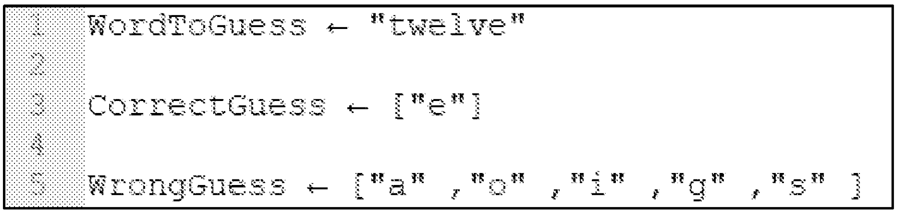
Variables are locations in memory where the data is stored; each of these locations has an address – a bit like your postal address – so the computer know where it has stored the data and where to find it again.

When we plan and write algorithms or create programs we name the variables u algorithms easier to understand. The name or identifier used should be easy to u algorithm or program.

Rules for variable names:

- The name must be written first before a value is **assigned** to it
- The name cannot start with a number; it must be a letter or an undersco
- Variable names must not have spaces – use CamelCase
- Names must be chosen that make sense in the algorithm

Example:



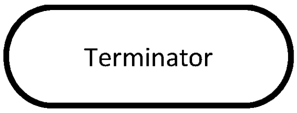
```
WordToGuess ← "twelve"

CorrectGuess ← ["e"]

WrongGuess ← ["a" ,"o" ,"i" ,"g" ,"s" ]
```

In AQA pseudocode the backwards arrow is used to show **assignment** of a value symbol (=) is used to show equality, e.g. 4 = 4 evaluates to True.

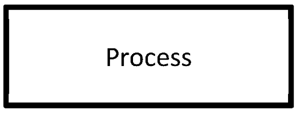
REPRESENTING ALGORITH

FLOW CHART SYMBOLS

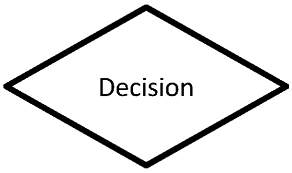
There are many different symbols used in flow charts; you need to be able to rec following symbols:



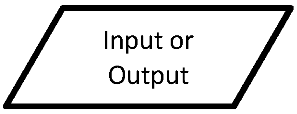
Start or end of your algorithm



A process in the algorithm, e.g. calculating the price of



A decision/selection symbol will always have one of tw true or false



This shows data into the algorithm or outputs from it

Arrows show the sequence of steps in the algorithm. Th or horizontal.

Key Terms

Flow chart A flow chart is a visual representation of the **sequence of step** in the sequence are shown as symbols or shapes which are linke arrows to show the order.

Variable A storage location used to store a value; this could be text or c variable may change as the program is run.

Complete Exercise 2: Charity Fundraiser – Put the Symbols in the Correct Order
Complete Crossword One

PSEUDOCODE

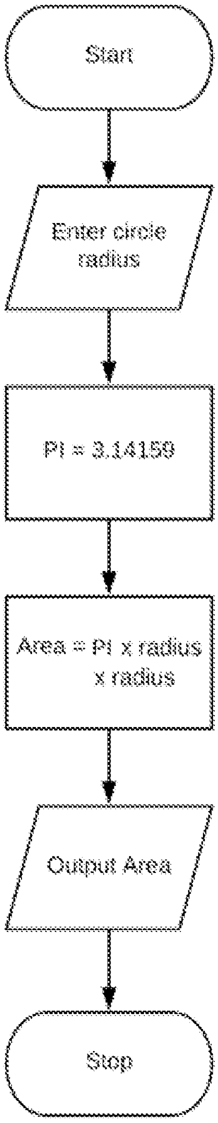
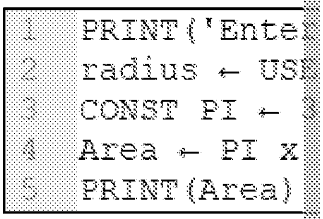
The second way we can plan and design an algorithm is using pseudocode.

PSEUDO means 'pretend' or 'unreal' as it is not a real programming language, it i an algorithm using text. Different textbooks will use different versions of pseudo your meaning clear and unambiguous, you can use any text to describe your ans

However, you will also be expected to be able to understand simple algorithms w version of pseudocode in your exam; this booklet will use the exam board versio

Look at the two examples of the same algorithm below.

Flow chart	Pseu

 Start
Enter circle radius
PI = 3.14159
Area = PI x radius x radius
Output Area
Stop
	
	PRINT('Ente
	radius ← US
	CONST PI ←
	Area ← PI x
	PRINT(Area)

This example uses both a **variable** and a **constant**; each of these has been given in this simple algorithm. Identifiers should make your algorithm easy to underst and they MUST be unique; you cannot use the same **identifier** or name for diffe algorithm.

Variables, Constants and Assignment

We have already discussed variables and briefly looked at how to **assign** a value

On Line 2 of the example, the pseudocode algorithm **assigns** the value entered b **variable** called **radius**. Each time the algorithm is used the value of 'radius' can c name 'radius' in our calculation we do not need to change anything when the va

The symbol used to show **assignment** of a value to a **variable** is a backwards arro and 4. It is important that the **identifier** for the variable is created BEFORE any va

When a variable in a program is NOT going to vary it is known as a **constant**. The the **identifier** in capital letters. Values are **assigned** to a **constant** in exactly the s you can see this on Line 3.

Key Terms

Constant A storage location used to store a value that never changes as

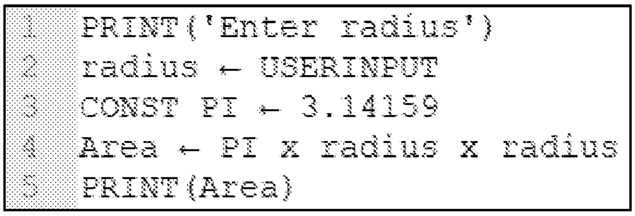
Assignment Giving a variable or constant a value by linking a value to the

Identifier A unique name given to a variable or constant in your algorith makes your algorithm easier to read and understand.

Pseudocode A structured, code-like language that can be used to describe

PRINT AND USERINPUT

In pseudocode you will be expected to understand and be able to use **keywords** capital letters.



```
1 PRINT('Enter radius')
2 radius ← USERINPUT
3 CONST PI ← 3.14159
4 Area ← PI x radius x radius
5 PRINT(Area)
```

The two keywords used here are:

- USERINPUT – Getting values into the algorithm from the user (via keybo
- PRINT – Messages or results displayed on the screen.

Complete Exercise 3: Constants or Variables?
Complete Exercise 4: Holiday Calculations

ARITHMETIC OPERATORS

You will be familiar with these from your Maths lessons, but there are some sligh able to recognise and use in pseudocode.

STANDARD ARITHMETIC OPERATORS	PSEUDOCODE VERSION	
Addition +	+	5 + 6 eval
Subtraction –	–	7 – 3 eval
Multiplication ×	*	4 * 2 eval
Division ÷	/	12/3 eval
Integer division (only evaluates the **quotient** from the division)	DIV	9 DIV 6 e *This evalu* *The **quoti*** *integer di*
Modulus operator (only evaluates the **remainder** from the division)	MOD	10 MOD *This evalu* *The **rema*** *modulus*

ORDER OF OPERATIONS: BIDMAS

Remember that you may have a question that involves understanding the order

For example:

6 × (7 + 3) = 6 × 10 = 60 ☑

6 × (7 + 3) = 6 × 7 = 42 + 3 = 45 ☒

4 + 5 × 6 = 4 + 30 = 34 (Multiply BEFORE addition or subtraction) ☑

1	**B**rackets
2	**I**ndices (powers, square roots)
3	**D**ivision
4	**M**ultiplication
5	**A**ddition
6	**S**ubtraction

Complete Exercise 5: Holiday Temperature Converter

Relational Operators

These are sometimes called equality or comparison operators as they are used t
expressions which use relational operators will evaluate to either True or False.

Operator	What it means	Example
<	Less than	5 < 7
>	Greater than	3 > 12
= or ==	Equality operator – checks whether both values are the same	5 = 5
<> or !=	Not equal to	7 <> 8
<=	Less than or equal to	9 <= 10 6.2 <= 6.2
>=	Greater than or equal to	12 >=21 5.7 >= 5.7

In a calculation, any arithmetic operators will be evaluated BEFORE relational op

Example:
(5 * 9) < 30 evaluates to False
(12 / 4) != (36/9) evaluates to True

Boolean or Logical Operators

Boolean or logical operators are very useful for combining with relational operat
expressions.

Operator	Explanation	
AND	Logical AND checks whether both conditions are true or false	A password must include a number
OR	Logical OR checks whether EITHER of the conditions is true	A password must symbol.
NOT	Logical NOT reverses a Boolean value. In the example x > y evaluates to False, using the logical NOT reverses the evaluation to True.	If a password is N NOT including a n

AND operator

We can start with two statements which could be true or false about your passw

1. The password has eight or more characters.
2. The password includes a number.

If you know the answer to both statements is true, they can be linked with AND

The password has eight or more characters	The password includes a number	T
False	False	
False	True	
True	False	
True	**True**	

In order to evaluate the overall value of a Boolean expression using the AND ope evaluate to true. For example:

```
(5 !=12)  AND    (12 > 8)

True      AND    True

True
```

OR OPERATOR

This is a very common logical operator that you will be familiar with when makin fries or chunky chips? The logical OR will evaluate to True if one of the choices ev

```
(5 != 12)  OR     (12 < 8)

True       OR     False

True
```

It does not matter if both choices evaluate to True as the overall expression will s

NOT OPERATOR

Unlike the AND and OR operators, which compare two Boolean expressions and simply reverses the result of the Boolean expression. For example:

```
NOT (12 > 8)

NOT True

False
```

COMBINING BOOLEAN OR LOGICAL OPERATORS

The three Boolean operators can be combined into more complex expressions b check whether the expression will give you the answer you want.

Example 1 (using variables)

```
PwdLen ← 8
NumCount ← 1
NOT (PwdLen < 8) AND (NumCount >= 2)
```

We can evaluate our expressions (PwdLen < 8) and (NumCount ≥ 2) to False; our AND False), which we can simplify to NOT False and, therefore, True.

Example 2

(5 != 12) OR (NOT (12 < 8))

The expressions (5 != 12) and (12 < 8) can be evaluated to True and True so we c this:

True OR (NOT (True))

This now evaluates to True OR False which evaluates to True.

Key Terms

Operator In maths, an operator is a symbol (such as + * / –) that shov something you want to do with the values.

Quotient When a number is divided by another, the result is known a 12 × 3 = 4, the quotient is 4.

Div Integer division gives only the quotient and ignores any rem

Mod The modulus operation finds the remainder only after divisio

Relational operator This is used in programming to compare two values, e.g. 4 relational operators will evaluate to either True or False.

Boolean operator A Boolean or logical operator is used to combine conditions tested to see whether they evaluate to True or False

Complete Crossword Two

PROGRAMMING CONSTRUCTS

When we are planning algorithms, there are three basic building blocks or 'const algorithms easy to read and easy to understand. These are used to control the o executed.

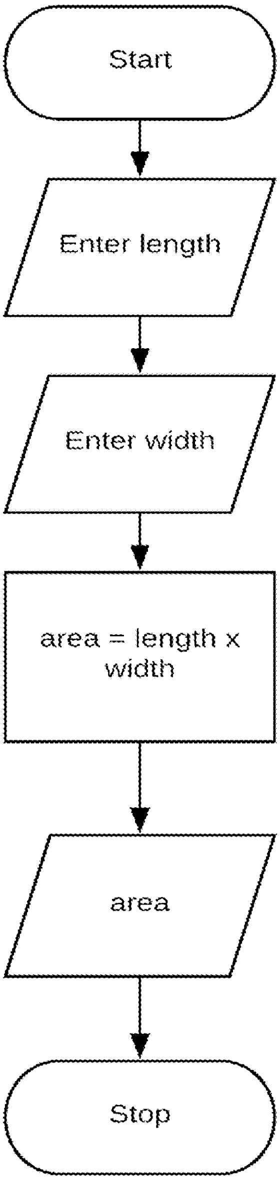
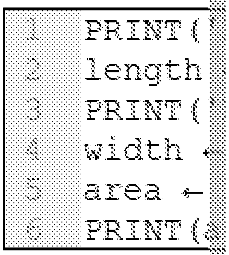
These building blocks also allow you to break a problem down into smaller block small blocks can then be joined together to solve a more complex problem.

The three constructs are:

- Sequence
- Selection
- Iteration

SEQUENCE

Sequence means doing things one after another. For example, when calculating t

FLOW CHART	PS
 Start Enter length Enter width area = length x width area Stop	PRINT(length PRINT(width ← area ← PRINT(a

Selection

Selection or conditional statements test whether a condition we have set is TRUE
choose or select what happens next based on whether the condition set evaluat

IF statement

```
IF < condition > THEN
        <statement when condition is true>
ENDIF
```

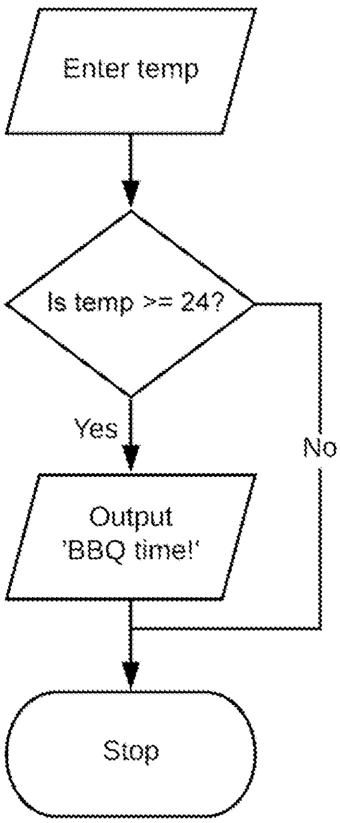
```
temp ← 25
IF temp >= 24 TH
    PRINT('BBQ
ENDIF
```

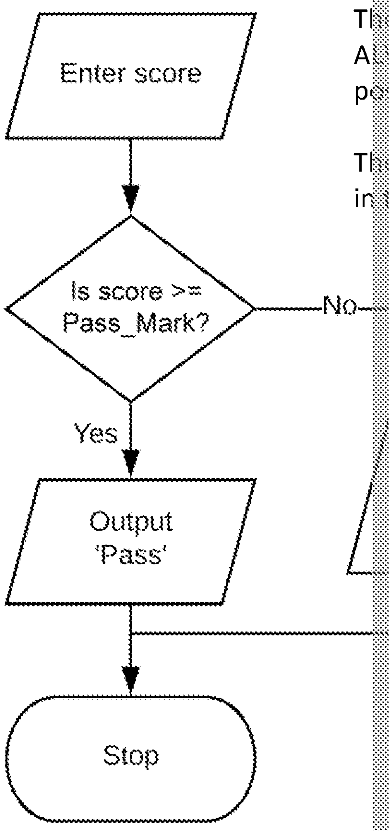
IF-ELSE statement

```
IF < condition > THEN
        <statement when condition is true>
ELSE
        <statement when condition is false>
ENDIF
```

```
Pass_Mark ← 60
score ← USERINPU

IF score >= Pass
    PRINT('Pass'
ELSE
    PRINT('Fail
ENDIF
```

Selection using flow charts





Th
Al
pe

Th
in

Complete Exercise 6: Odds or Evens

What happens if you want to check more than one condition?

In the example code below the algorithm will check whether one of the conditic execute the relevant code. If the first condition evaluates to True then none checked; if none of the conditions evaluates to True, then the default **else** code w

ELSE-IF STATEMENT

```
IF <condition> THEN
      <statement when
      condition is true>
ELSE IF < next condition>
THEN
      <statement when
      condition is true>
ELSE IF <next condition >
THEN
      <statement when
      condition is true>
ELSE
      <do this>
ENDIF
```

```
score ← USERINPUT

IF score >= 80 THEN
     PRINT('Levels 7 - 9
ELSE IF score <= 79 AND
     PRINT('Levels 4 - 6
ELSE IF score <= 59 AND
     PRINT('Levels 1 - 3
ELSE
     PRINT('Failed. Plea
ENDIF
```

Example 1:

Line	Score	score >= 80	score <= 79 AND score>= 60	score < scor
1	32			
2				
3		False		
4				
5			False	
6				
7				F
8				
9				
10				
11				

Example 2:

Line	Score	score>= 80	score <=79 AND score>= 60	score < scor
1	60			
2				
3		False		
4				
5			True	
6				
7				
8				
9				
10				
11				

As you can see in the trace table above, the ELSE-IF statement will not check any statement has been evaluated to True; Lines 6 to 11 would be ignored and the a line of code after Line 11.

Complete Exercise 7: Colour Range

Trace Tables

A trace table is a useful way of checking the logic of your algorithm BEFORE yo code. It involves using a range of data to check that the algorithm you have wri the example table shown on the pervious page.

This type of checking is always done on paper and is called performing a 'dry run

Complete Exercise 8: Trace Table 1
Complete Exercise 9: Trace Table 2

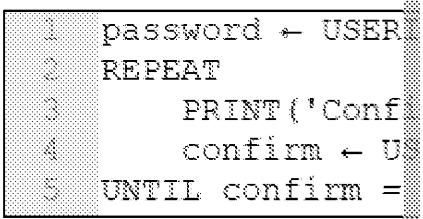
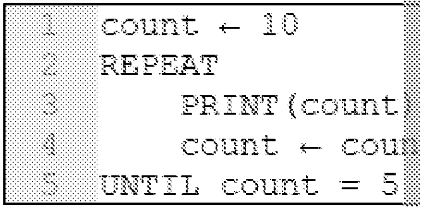
Iteration

In computer science, iteration means that instructions in your algorithm are repe of iteration or LOOPING that you need to know about:

- Condition-controlled Loop (indefinite iteration)
- Count–controlled Loop (definite iteration)

Condition-controlled Loop (indefinite iteration)

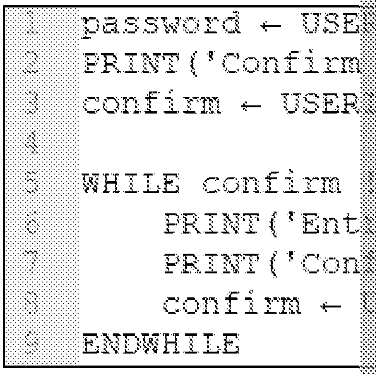
REPEAT... UNTIL...	

`REPEAT`
`      <statements>`
`UNTIL  <Boolean Expression>`
	Example 1
	
	`password ← USER`
	`REPEAT`
	`    PRINT('Conf`
	`    confirm ← U`
	`UNTIL confirm =`
	This will continue to ask for the us they match.
	Example 2
	
	`count ← 10`
	`REPEAT`
	`    PRINT(count`
	`    count ← cou`
	`UNTIL count = 5`
	This will print out 10, 9, 8, 7, 6 an

WHILE… ENDWHILE

WHILE <Boolean Expression>
    <statements>
ENDWHILE

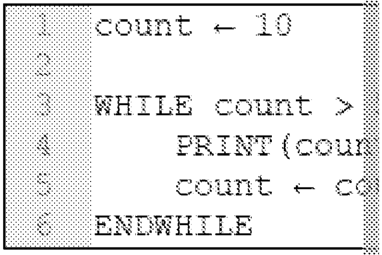
Example 1



```
password ← USE
PRINT('Confirm
confirm ← USER

WHILE confirm
    PRINT('Ent
    PRINT('Con
    confirm ←
ENDWHILE
```

In this example, the condition
If the password and confirm e

Example 2



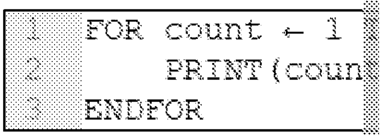
```
count ← 10

WHILE count >
    PRINT(coun
    count ← co
ENDWHILE
```

Again, the condition is checke
used will be different from the
want to output the numbers 1

Count-controlled Loop (definite iteration)

FOR… ENDFOR

FOR identifier ← IntExp TO IntExp
    <condition>
ENDFOR



```
FOR count ← 1
    PRINT(coun
ENDFOR
```

This will output 1,2,3,4,5,6,7,8

Key Terms

Construct The basic building blocks of an algorithm or progr instructions are executed.

Sequence When instructions are executed, in order, one after

Selection Also known as a conditional statement, this allows t instructions based on whether a condition is True or

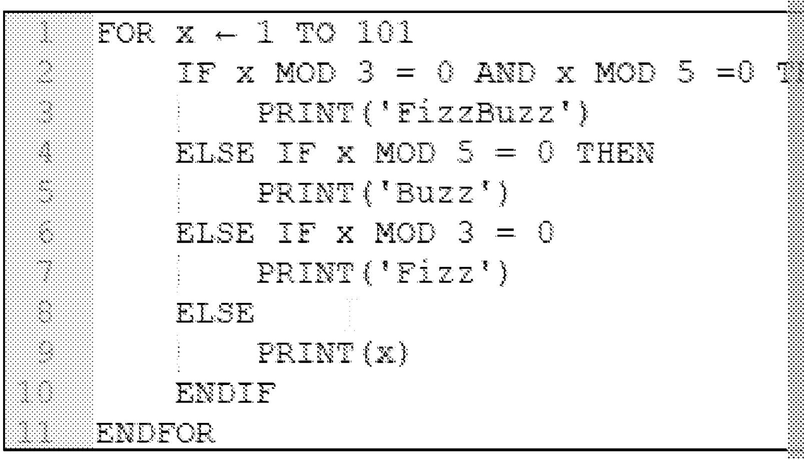
Iteration Instructions are repeated either until a condition is T times.

Trace table A manual method of testing an algorithm to ensure

Combining Sequence, Selection and Iterati

As you will probably be aware from your knowledge of programming so far, prob combination of these three building blocks or 'constructs'.

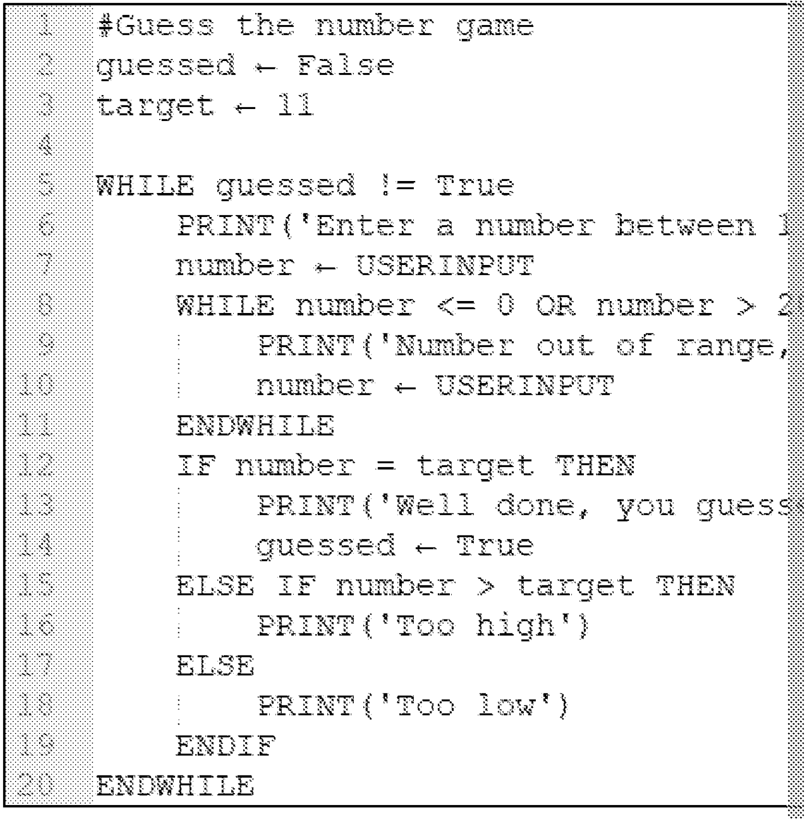
Example 1:



```
FOR x ← 1 TO 101
    IF x MOD 3 = 0 AND x MOD 5 =0 T
        PRINT('FizzBuzz')
    ELSE IF x MOD 5 = 0 THEN
        PRINT('Buzz')
    ELSE IF x MOD 3 = 0
        PRINT('Fizz')
    ELSE
        PRINT(x)
    ENDIF
ENDFOR
```

This is an example of a simple programming task often used in interviews to chec problems and code a solution that works!

Example 2:



```
1  #Guess the number game
2  guessed ← False
3  target ← 11
4
5  WHILE guessed != True
6      PRINT('Enter a number between 1
7      number ← USERINPUT
8      WHILE number <= 0 OR number > 2
9          PRINT('Number out of range,
10         number ← USERINPUT
11     ENDWHILE
12     IF number = target THEN
13         PRINT('Well done, you guess
14         guessed ← True
15     ELSE IF number > target THEN
16         PRINT('Too high')
17     ELSE
18         PRINT('Too low')
19     ENDIF
20 ENDWHILE
```

This example uses the variable **guessed** as a 'flag' on Line 2. Variables used as fla The flag variable will be set with an initial value (True or False), depending on wh

When the code on Line 12 evaluates to True then the '**flag**' on Line 14 (the varia with the result that the condition on Line 5 will now evaluate to False and the lo

Complete Exercise 10: Identify the Constructs
Complete Exercise 11: FizzBuzz
Complete Exercise 12: Dial a Pizza
Complete Crossword Three
Complete Exercise 13: Count until Zero

DATA STRUCTURES

ARRAYS

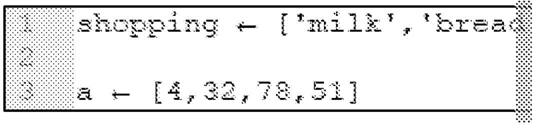
An array is a data structure that allows us to store multiple items using just one v array is usually referred to as an 'element'.

Most modern programming languages start numbering array indexes at 0; this w in your exam **unless the question tells you otherwise**.

ASSIGNMENT (OF AN ARRAY)

Identifier ← [Exp, Exp, Exp,…,Exp]

Note: Exp means any expression



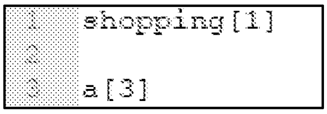
```
shopping ← ['milk','brea

a ← [4,32,78,51]
```

The shopping array has three elements star finishing at index position 2.

The a array has four elements, starting at in index position 3.

ACCESSING AN ELEMENT

Identifier [IntExp]

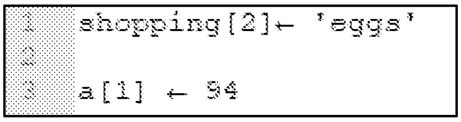


```
shopping[1]

a[3]
```

Note: shopping [1] will evaluate to 'bread' a these are the index positions of each item in

UPDATING AN ELEMENT

Identifier [IntExp] ← Exp

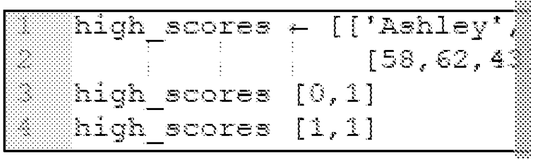


```
shopping[2]← 'eggs'

a[1] ← 94
```

Note: the element at index position shoppin 'butter' to 'eggs'. The array is now ['milk','b the element at index position a [1] has been array is now [4,94,78,51].

ACCESSING AN ELEMENT IN A 2D ARRAY

Identifier [IntExp] [IntExp]



```
1  high_scores ← [['Ashley',
2                    [58,62,43
3  high_scores [0,1]
4  high_scores [1,1]
```

Note: You can think of a 2D array as being l example, Line 3 would evaluate to the secon 2D array (['Ashley', 'Raheem', 'Jamie']), i.e. R the second item in the second element of th

You should think of a 2D array as looking like a table:

Ashley	Raheem	Jami
58	62	43

Row 1, column 0
high_scores [1, 0]

UPDATING AN ELEMENT IN A 2D ARRAY	
Identifier [IntExp] [IntExp] ← Exp	`high_scores [1,1] ← 67` *Note: This results in the 2D array now lookir has been increased from 62 to 67. [['Ashley', 'Raheem', 'Jamie'],[58,67,43]]*

ARRAY LENGTH	
LEN(Identifier)	`LEN (high_scores)` will evaluate to 2 player names in row 0 and the player score `LEN (shopping)` will evaluate to three

FOR LOOPS AND ARRAYS

When we use arrays to store multiple data items, a common process is to search whether it contains an item or to perform some other operation on each item.

We know how to use a FOR loop for counting:

```
FOR count ← 1
    PRINT(coun
ENDFOR
```

How do we loop through each item in an array using a FOR loop?

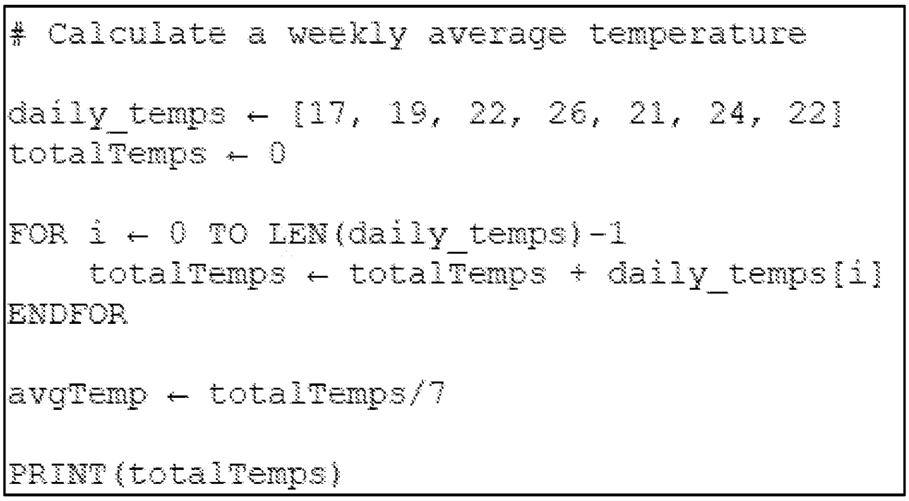
Example:

```
daily_temps ← [17, 19, 22, 26, 21, 24, 22]
totalTemps ← 0

FOR i ← 0 TO 6
    totalTemps ← totalTemps + daily_temps[i]
ENDFOR

avgTemp ← totalTemps/7

PRINT(totalTemps)
```

The array **daily_temps** has seven items but we are assuming (unless the exam qu array counting starts at index position 0.

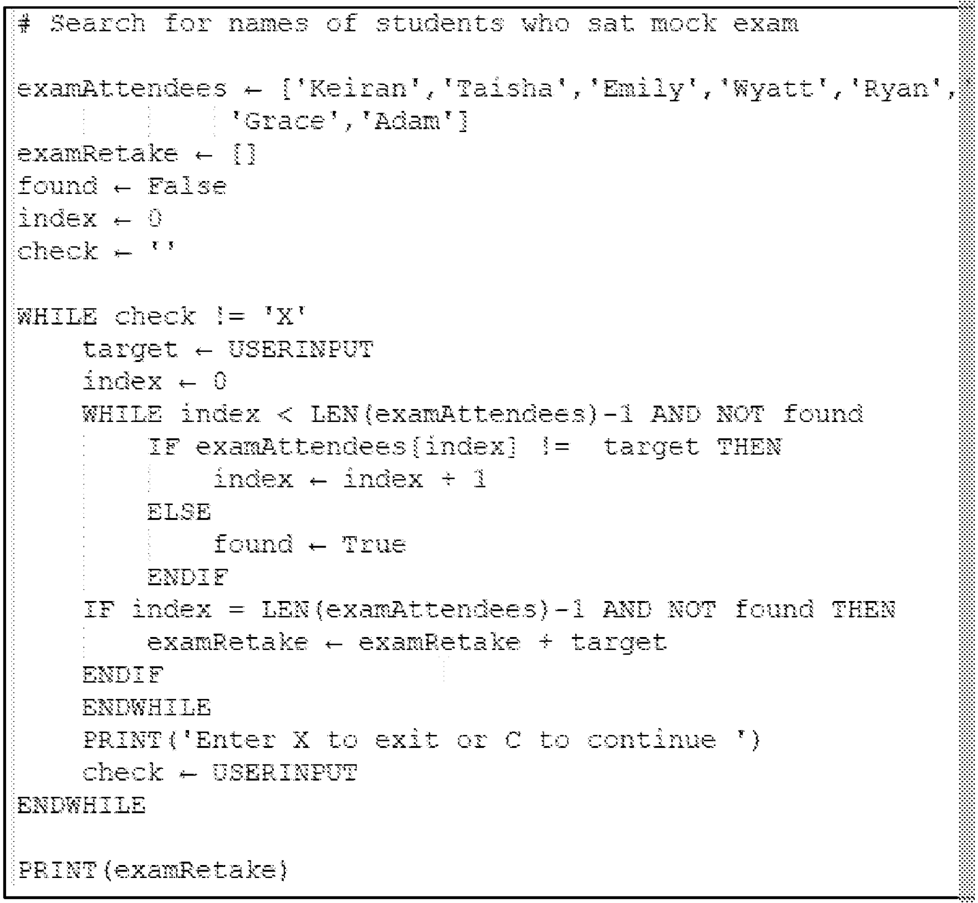
The FOR loop looks at the **index position** of each item in the array and then adds position to the variable **totalTemps**. The total is then divided by 7 to find the ave

What happens if we do not know how large the array will be for setting the cond
We can use the LEN () option to find the length of an array as well as a string. We c



```
# Calculate a weekly average temperature

daily_temps ← [17, 19, 22, 26, 21, 24, 22]
totalTemps ← 0

FOR i ← 0 TO LEN(daily_temps)-1
    totalTemps ← totalTemps + daily_temps[i]
ENDFOR

avgTemp ← totalTemps/7

PRINT(totalTemps)
```

Another process that can be achieved using arrays, WHILE loops and selection st
array includes a data item, e.g. a name:



```
# Search for names of students who sat mock exam

examAttendees ← ['Keiran','Taisha','Emily','Wyatt','Ryan',
                 'Grace','Adam']
examRetake ← []
found ← False
index ← 0
check ← ''

WHILE check != 'X'
    target ← USERINPUT
    index ← 0
    WHILE index < LEN(examAttendees)-1 AND NOT found
        IF examAttendees[index] !=  target THEN
            index ← index + 1
        ELSE
            found ← True
        ENDIF
    IF index = LEN(examAttendees)-1 AND NOT found THEN
        examRetake ← examRetake + target
    ENDIF
    ENDWHILE
    PRINT('Enter X to exit or C to continue ')
    check ← USERINPUT
ENDWHILE

PRINT(examRetake)
```

In this example, the algorithm is searching the array looking for a name entered

The WHILE loop looks at each item in the array; if an item does not match the ta
1. If the end of the array is reached and the target name is not found, then that s
exam and their name is added to the dynamic array called **resits**.

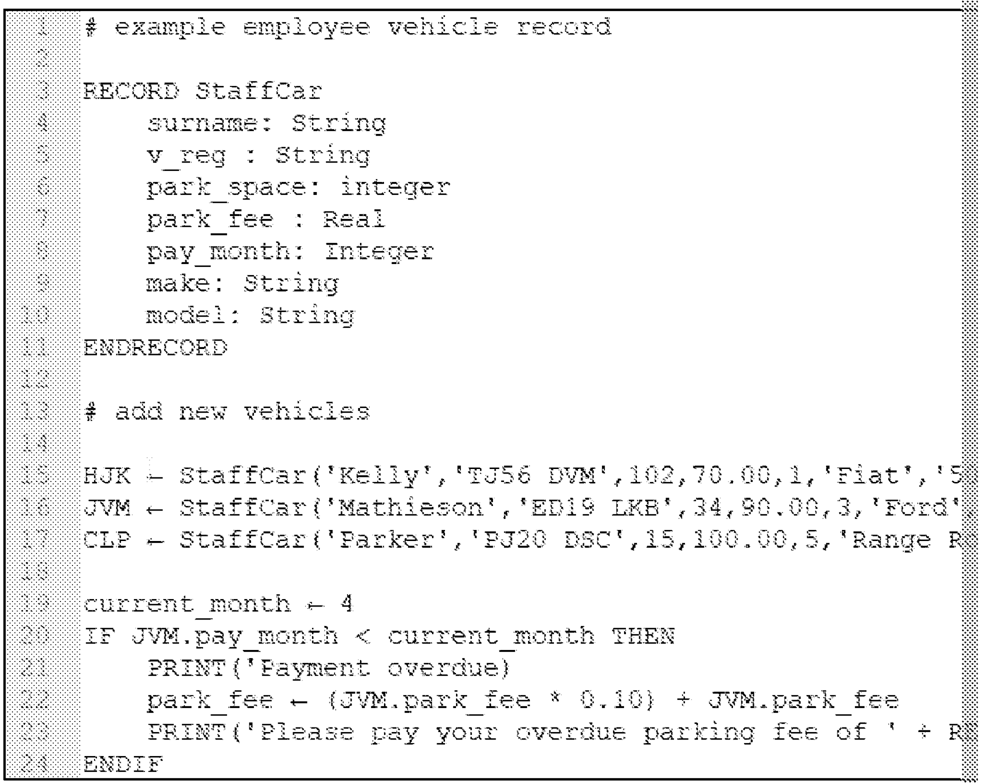
When the user has finished entering names and enters 'X', the main loop finishe

Complete Exercise 14: Calculate Fares

RECORDS

Records are data structures that allow multiple data types to be identified by
have used something similar like a dictionary in Python to store data.

In this example we are looking at the records of employee cars for a large firm.
space depending on the location in the car park and must park in their numbered



```
# example employee vehicle record

RECORD StaffCar
    surname: String
    v_reg : String
    park_space: integer
    park_fee : Real
    pay_month: Integer
    make: String
    model: String
ENDRECORD

# add new vehicles

HJK ← StaffCar('Kelly','TJ56 DVM',102,70.00,1,'Fiat','5
JVM ← StaffCar('Mathieson','ED19 LKB',34,90.00,3,'Ford'
CLP ← StaffCar('Parker','PJ20 DSC',15,100.00,5,'Range R

current_month ← 4
IF JVM.pay_month < current_month THEN
    PRINT('Payment overdue)
    park_fee ← (JVM.park_fee * 0.10) + JVM.park_fee
    PRINT('Please pay your overdue parking fee of ' + R
ENDIF
```

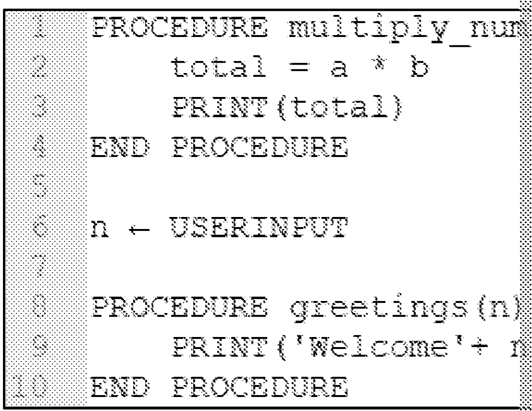
A record is a data type created by a programmer for a specific purpose, so
required for the task.

SUBROUTINES

Subroutines are clear, independent blocks of code within a computer program w by the main program. These are also known as functions or procedures, dependi value or not.

PROCEDURE DEFINITION

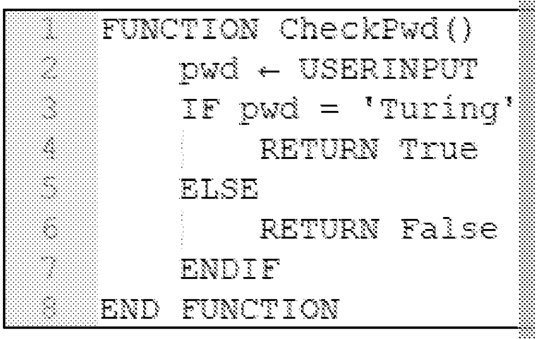
```
PROCEDURE
Identifier (parameters)
        <statements>
END PROCEDURE
```



```
PROCEDURE multiply_nu
    total = a * b
    PRINT(total)
END PROCEDURE

n ← USERINPUT

PROCEDURE greetings(n)
    PRINT('Welcome'+ n
END PROCEDURE
```

FUNCTION DEFINITION

```
FUNCTION
Identifier (parameters)
        <statements>
        RETURN Exp
END FUNCTION
```



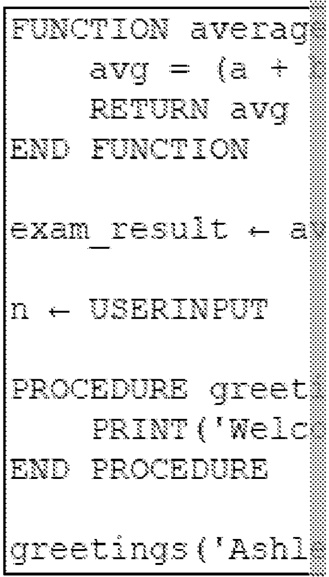
```
FUNCTION CheckPwd()
    pwd ← USERINPUT
    IF pwd = 'Turing'
        RETURN True
    ELSE
        RETURN False
    ENDIF
END FUNCTION
```

CALLING A SUBROUTINE (PROCEDURE/FUNCTION)

```
multiply_nums (5,12)

pwd_result ← CheckPwd()
```

In order to use the procedure or function th of 'calling' them and providing any inputs in The inputs to the procedure `multiply_n` *The expression returned from the function* `C` *variable name* `pwd_result`.

Parameters and Arguments



```
FUNCTION averag
    avg = (a +
    RETURN avg
END FUNCTION

exam_result ← a

n ← USERINPUT

PROCEDURE greet
    PRINT('Welc
END PROCEDURE

greetings('Ashl
```

The example shows the use of parameters. These are 'placeholders' inside the brackets after the name of the subroutine. Not all subroutines will need a parameter value.

The CheckPwd () subroutine shown on the previous page has no parameters so the brackets are empty.

The **parameters** used in the first subroutine are **a**, **b**; in the second subroutine the parameter is **n**.

Arguments are the actual values we use when we 'call' the function, as shown here.

Key Terms

Nesting This means combining code together; for example, putting a WHILE loop or an IF statement inside another IF statement.

Array An array is a data structure that allows us to store multiple name, e.g. vw_cars ← ["Up!", "Polo", "Golf", "T-Roc", "Tigu

Subroutine Subroutines are clear, independent blocks of code within a called and accessed by the main program.

Call The term used to describe 'starting' the subroutine.

Return Subroutines that return values (to be used elsewhere in the c Subroutines that do not return any values (e.g. printing out a procedures.

Complete Exercise 15: Guessing Game using Subroutines

STRING HANDLING

A string is a sequence of characters; these could be letters, numbers, punctuatio always surrounded by single or double quotation marks.

STRING LENGTH	
LEN (StringExp)	`LEN ('the quick brown fox')` will evaluate to 19, which includes three

POSITION OF A CHARACTER	
POSITION (StringExp, CharExp)	`POSITION('the quick brown f` will evaluate to 4 *REMEMBER: as with arrays, exam pape is specifically stated otherwise.*

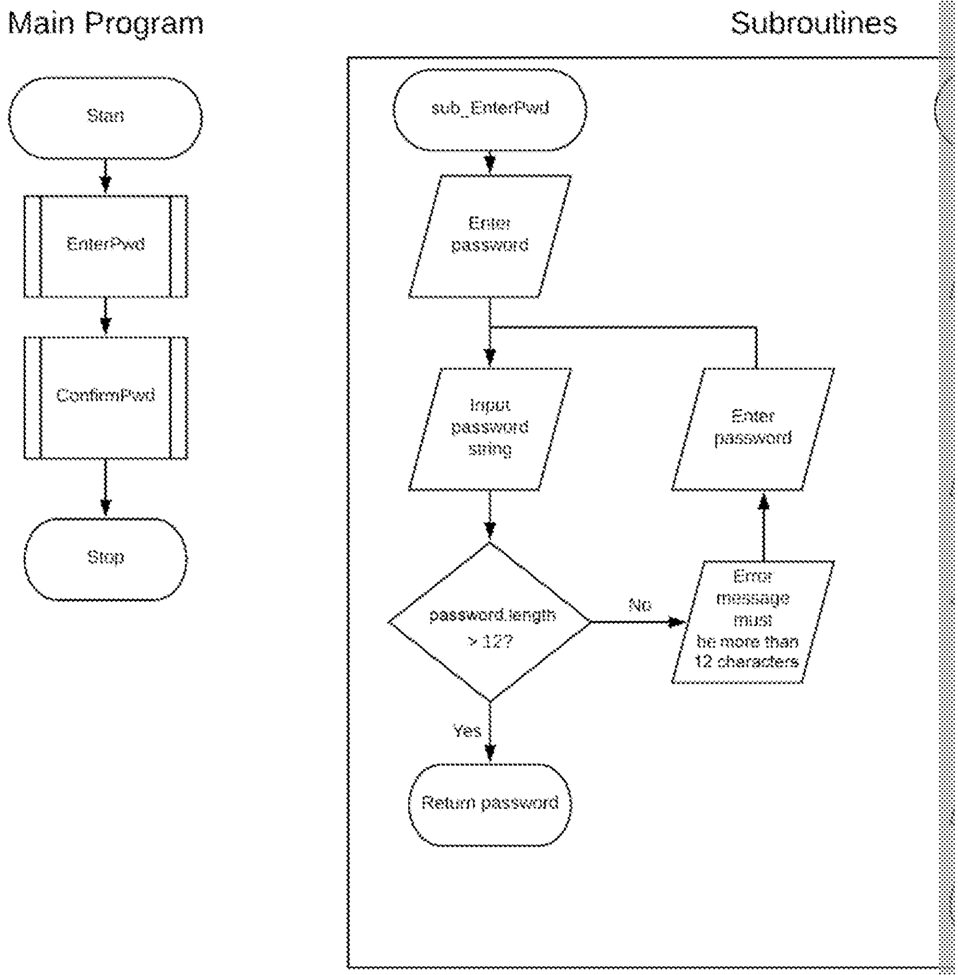
SUBSTRING	
SUBSTRING (IntExp, IntExp, StringExp)	`SUBSTRING(4,14,'the quick b` will evaluate to 'quick brow' *Note: the first parameter indicates the s the second parameter indicates the end*

CONCATENATION	
StringExp + String Exp	`'the quick brown fox' + ' jump` will evaluate to 'the quick brown fox jumpe

Complete Exercise 16: Strings and Substrings

Flow Charts and Subroutines

We have looked in detail at how to write a subroutine in pseudocode and how to show these structures using flow charts.



Explanation

The first subroutine, **sub_EnterPwd**, asks for the password and checks that it is l
not, an error message is displayed. This is then looped until a password of over 1

The password is returned from this subroutine and passed as an argument into s

The second subroutine, **sub_ConfirmPwd**, then asks for the password to be conf
string and the confirm string do not match, an error message is shown and the u
Again, this loops until the correct matching string is entered.

Complete Exercise 17: Area Tester
Complete Crossword Four

String and Character Conversion

String to Integer	
STRING_TO_INT (StringExp)	STRING_TO_INT (“24”) evaluates to the integer 24
String to Real	
STRING_TO_REAL (StringExp)	STRING_TO_REAL (“24.25”) evaluates to the real 24.25
Integer to String	
INT_TO_STRING (IntExp)	INT_TO_STRING (74) evaluates to the string “74”
Comments	
Single line comments	# code written to the right o
Multiline comments	# comment # additional comments

Real to String	
REAL_TO_STRING (RealExp)	REAL_TO_STRING (19.56) evaluates to the string “19.56”

Convert a string to upper/ lowercase	

TOUPPER (StringExp)
TOLOWER (StringExp)
	postcode ← TOUPPER(postcode)
	quiz_answer ← TOLOWER(quiz_an

Char to Code	
CHAR_TO_CODE (CharExp)	CHAR_TO_CODE('G') evaluates to 71 using ASCII/Unicode
CODE TO CHAR	
CODE_TO_CHAR(IntExp)	CODE_TO_CHAR (103) evaluates to ‘g’ using ASCII/Unicode

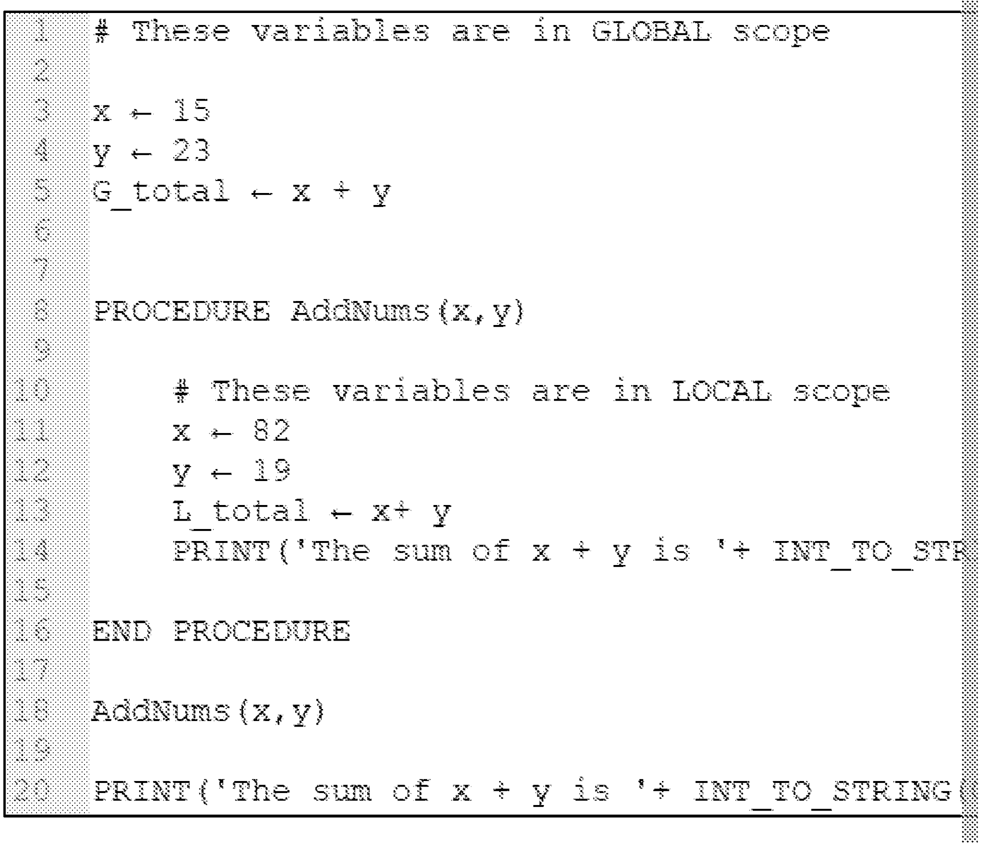
Scope of Variables, Constants and Subrouti

You may have heard of this term in your lessons on programming. The scope of a is about <u>where</u> that variable, constant or subroutine can be used. It is important t scope or you may get unexpected results from your code.

There are two types:

- Local
  - This means that the variable, constant or subroutine can only be where it is defined, e.g. inside a subroutine.
- Global
  - This means that the variable, constant or subroutine can be used

Example:



```
1  # These variables are in GLOBAL scope
2
3  x ← 15
4  y ← 23
5  G_total ← x + y
6
7
8  PROCEDURE AddNums(x,y)
9
10     # These variables are in LOCAL scope
11     x ← 82
12     y ← 19
13     L_total ← x+ y
14     PRINT('The sum of x + y is '+ INT_TO_STR
15
16 END PROCEDURE
17
18 AddNums(x,y)
19
20 PRINT('The sum of x + y is '+ INT_TO_STRING(
```

The output from the subroutine AddNums(x, y) would be:

```
'The sum of x + y is 101'
```

Although the subroutine takes in the two parameters, x and y, which are GLOBAL INSIDE the code block <u>with the same name</u> will take precedence, i.e. will be used removed, there are NO local variables. The subroutine will then use the GLOBAL v

The output from the code on Line 20 will be:

```
 'The sum of x + y is 38'
```

The code on Line 20 is not part of any code block and so will use the GLOBAL vari This example shows the use of the same variable name in both the GLOBAL and L should try to use <u>different</u> meaningful identifiers/names for your variables as this make your code clearer and easier to understand.

DEALING WITH ERRORS: VALIDATION TECHNIQUE

When you are planning and designing algorithms it is important to think about a errors could occur in the logic of your design and write solutions that will deal w without crashing the program. This is called 'validation'.

You have already seen examples and exercises where the code checked the leng continue until the data entered matched a specified minimum length, or in Exer in a certain range.

The example below prompts the user to enter the data in integers, but we also n enter 'fifteen' instead of 15 to make sure our algorithm will not fail.

```
PRINT(' Enter your age: ')
age ← USERINPUT
```

CATCHING ERRORS

A common way of dealing with incorrect data types being entered is using error h 'exceptions', which you may have already encountered in your programming less 'exceptional' happens we can 'catch' the error and output a message or write co

In the simple example above we want to:

1. Ask the user for data
2. If the data type entered is not an integer, output an error message
3. Loop back to No. 1

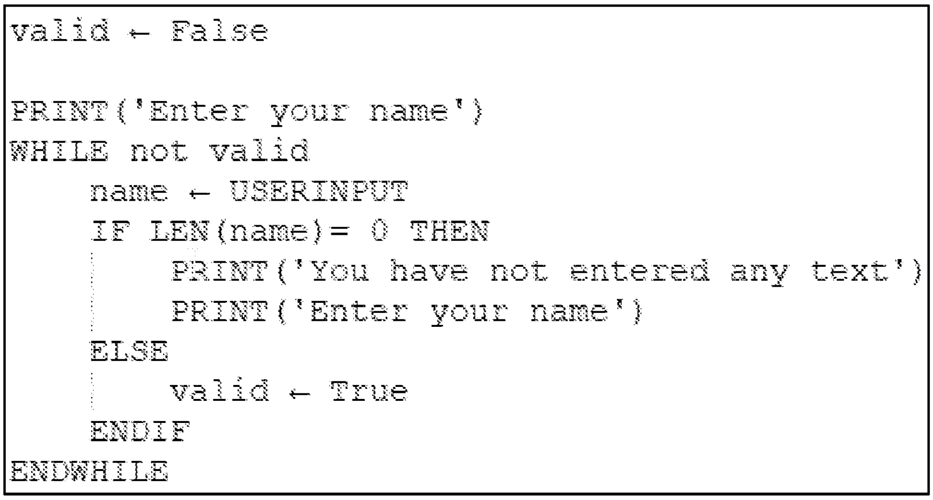
```
valid ← False

PRINT(' Enter your age: ')
REPEAT
    age ← USERINPUT
    TRY
        ageNumber ← STRING_TO_INT(age) # the typ
        valid ← True
        IF ageNumber < 16 THEN
            PRINT('You cannot drive anything yet
        ELSE IF ageNumber >= 16  AND ageNumber <
            PRINT('You can drive a moped at 16 a
        ELSE
            PRINT('You can now drive any vehicle
        ENDIF
    CATCH
        PRINT('Please enter age as a number in y
UNTIL valid
```

The code in the block is execut

If it fails (the age e then the code ju

The REPEAT… UNTIL loop continues until integers are entered as the age variabl value of the flag variable valid to True and the REPEAT loop ends.

Another simple example is a 'presence check'. This means checking that some da example, when asking for data such as a password or a name.



```
valid ← False

PRINT('Enter your name')
WHILE not valid
    name ← USERINPUT
    IF LEN(name)= 0 THEN
        PRINT('You have not entered any text')
        PRINT('Enter your name')
    ELSE
        valid ← True
    ENDIF
ENDWHILE
```

In this example, the length of the input string is checked before the algorithm co

Another option is to 'cast' the USERINPUT to the data type you want by wrappin around the USERINPUT, e.g.

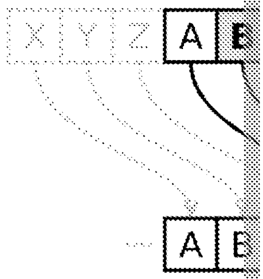
```
age ← STRING_TO_INT (USERINPUT)
```

Complete Exercise 18: Password Checker Validation

USING CHAR TO CODE AND CODE TO CHAR

A Caesar cipher is a simple way to encrypt messages using the numerical values b another a set number of places along in the alphabet.

If we know that CHAR_TO_CODE (A) evaluates to 65 (using the ASCII/Unicode tables) then substituting the letter A with another 11 places further on in the alphabet simply involves adding 65 + 11, and using CODE_TO_CHAR (76) will evaluate to the letter L.



Complete Exercise 19: Encryption Cipher

RANDOM NUMBER GENERATION	

RANDOM_INT(IntExp, IntExp)	`options ← RANDOM_INT(5, 8)`
	`# will generate 5, 6, 7 or 8)`

Complete Exercise 20: Simple Battleships
Complete Exercise 20A: Battleships Extension

APPROACHES TO PROBLEM-SO

DECOMPOSITION

One of the most important skills in computer science is problem-solving. Computer for themselves and, although it can compute at faster and faster speeds, a compute been designed and developed by humans.

The term 'problem-solving' means the ability to analyse problems, consider a ran the chosen solution clearly, perhaps in the form of an algorithm that can be tran

An important technique used by computer scientists to analyse a complex proble to break a problem down into smaller and smaller parts, until each part become have already done this earlier in this booklet when we split problems up into inp order to make them easier to solve.

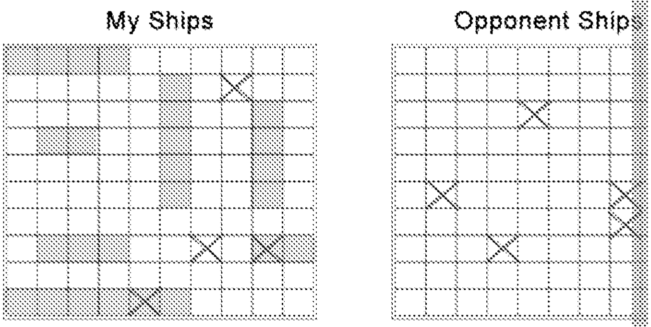
We can look at how we might plan our own battleships game starting with ident

1. Create a game board
2. Add ships to the board
3. Record hits on opponent board
4. Record hits on own board
5. Organise player turns
6. How to calculate when a play

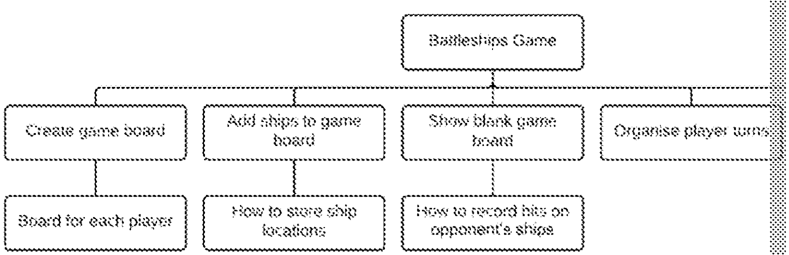
We can then look at each task, and consider: *Can it be solved in one go or does it*

1. Create game board

a. The game board must show hits on opponents and ship position, and hit
    i. Display must change the game boards with each player turn.
    ii. Display must show where hits have landed for opponents.
    iii. Display must show where opponent's hits have landed and whether s



Each problem must be broken down into smaller and smaller sub-problems until solved. Some people prefer to break up a problem by using charts like the one gaming software often have different teams of programmers working on differe thing each team needs to know about another part of the game is how to join on



The technical term for this process is **decomposition**.

Complete Exercise 21: RPG Game Inventory

ABSTRACTION

Abstraction is an important skill that is used when solving complex problems; it is unnecessary detail to focus on what is important in order to solve the problem.

There are many different examples of abstraction in everyday life; for example, w driving I do not need to know how the engine works, how the power from the en – I just need to know how to operate the car.

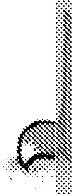
When you get in from school and need a quick snack, you do not need to know h order to heat up / cook your snack; just how to operate the microwave.

Here is a classic brain-teaser to demonstrate how details can be removed to mak

ABSTRACTION EXAMPLE 1

You have a fox, a chicken and a sack of grain.

You must cross a river, which is 20 metres wide, using a red rowing boat with only one of them at a time. If you leave the fox with the chicken he will eat it; if you leave the chicken with the grain he will eat it.



How can you get all three across safely?

We will call the side of the river you are on bank A, and the side you want to move to bank B.

There is a simple computational approach for solving this problem. We could try means trying every possibility, but logical thinking will help us find the solution m

We will first remove all the irrelevant detail from the problem:

- *Is the width of the river important?*
- *Is the colour of the boat important?*
- *Is it important that it is a rowing boat?*

We can now start with the following information:

1. River banks are A and B
2. Fox = F
3. Chicken = C
4. Grain = G

At the moment we have this:

A	B
FCG	

We want to end up with this:

A	B
	FCG

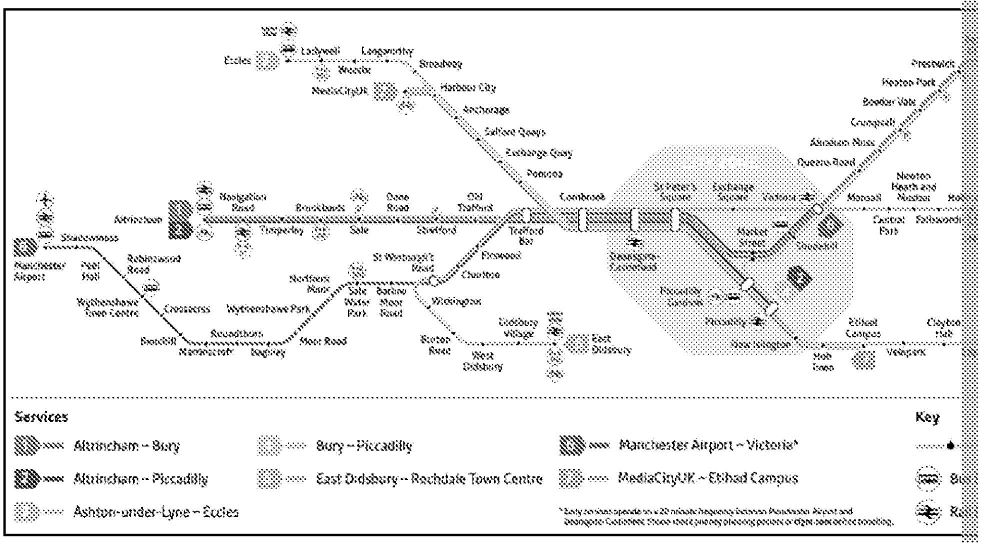
<u>Step 1</u>: Take the chicken across to bank B as the fox will not eat the grain but it w

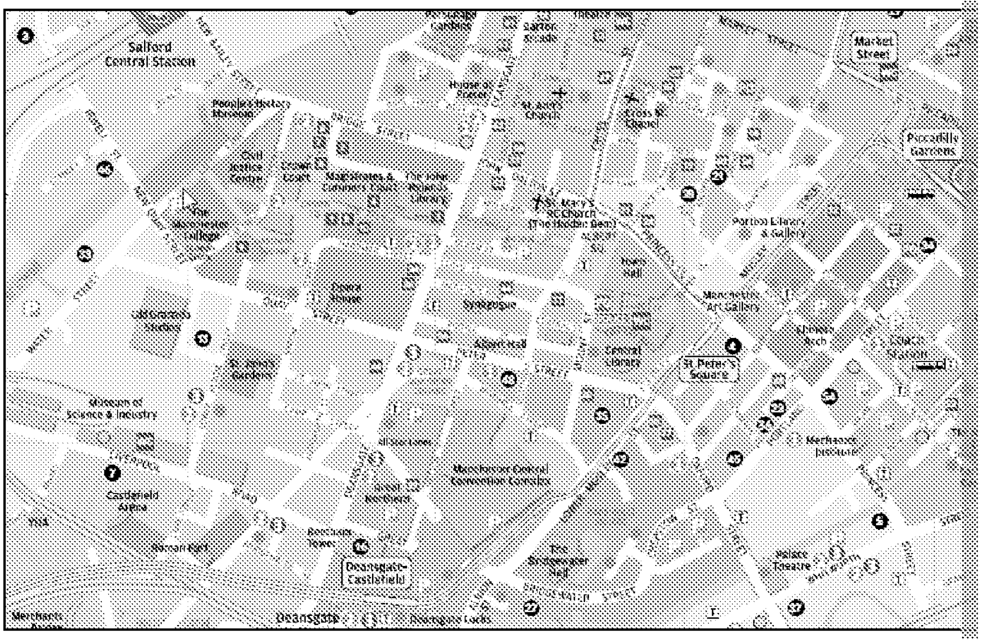
How many steps are left? What are they?

Abstraction example 2

If you have ever travelled on the London Underground, then you will be familiar below. The original was designed by Henry Beck, an electrical engineer. He creat London Underground in 1933 based on a circuit board layout; now many maps u

Here is a current version of the Metrolink tram system in Manchester. Below yo the city centre actually looks like, with some of the stations listed in the first ma





It is obvious that the first map is easier and clearer to read as all the irrelevant de

Complete Exercise 22: Music Gig

Key Terms

Decomposition This means breaking a problem down into smaller sub-prob tasks that can be solved.

String A sequence of characters, which could be letters, numbers, p surrounded by single or double quotation marks.

Concatenation This means merging or joining two strings together using the

Abstraction The process of removing unnecessary detail from a problem

EFFICIENCY OF ALGORITH

You should now be aware that algorithms are a fundamental part of problem-so
and there may be several different algorithms available which will all solve the sa

So how do we choose which is best?

There are two different measures that are used to measure the efficiency of an a

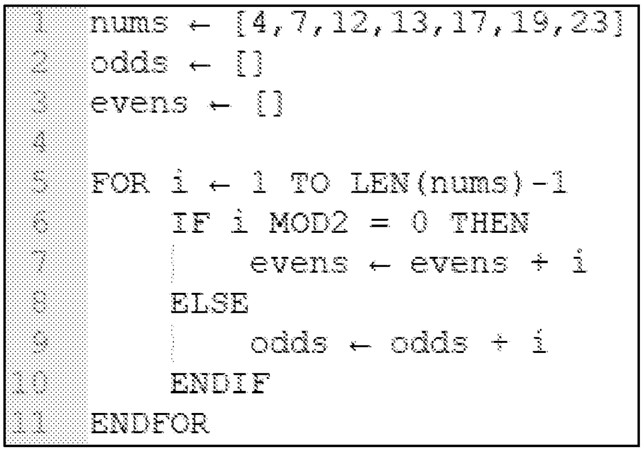
- Time – the amount of time the algorithm takes to complete.
- Space – the amount of memory that the computer needs to use to comp
  data items.

For example, if you have to solve a problem of finding a person's details from 50
one until you find the record you need, this would not take very long. However,
thousands and thousands of data records, the time needed to solve the problem

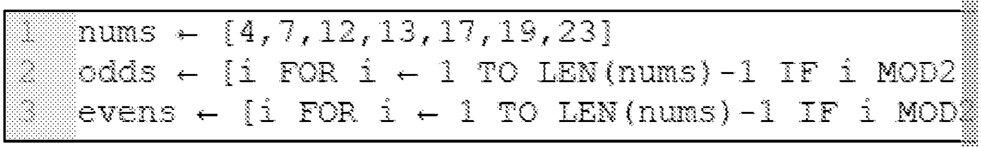
Looking at each record would eventually find the record you are searching for or
the data. This type of approach is known as a **brute force** approach as it solves th
numbers of records is not efficient; the time taken to complete the search grows
needs to be searched grows.

Some algorithms are suitable for small data sets but can then become very ineffi
This can be due to the way the code has actually been written.

Example: The code below will sort the data in the original array into two new arr
the condition set in the IF statement.



```
nums ← [4,7,12,13,17,19,23]
odds ← []
evens ← []

FOR i ← 1 TO LEN(nums)-1
    IF i MOD2 = 0 THEN
        evens ← evens + i
    ELSE
        odds ← odds + i
    ENDIF
ENDFOR
```



```
nums ← [4,7,12,13,17,19,23]
odds ← [i FOR i ← 1 TO LEN(nums)-1 IF i MOD2
evens ← [i FOR i ← 1 TO LEN(nums)-1 IF i MOD
```

The second example uses less code to achieve the same result, i.e. an array of od
numbers.

EFFICIENT CODE PROOF

If the pseudocode is translated into Python, the code can be tested to see w The examples below use a built-in function that enables the two approaches to b

The code has been amended into functions to make the time comparison easie sorted has been increased.

```
1       def sort_odds_evens_lc():
2           """sorts array using list comprehension"""
3
4           nums = [5, 6, 10, 16, 18, 24, 25, 30, 34, 36, 37
5                   64, 68, 75, 77, 80, 81, 83, 85, 88, 93,
6           evens = [i for i in nums if i % 2 == 0]
7           odds = [i for i in nums if i % 2 != 0]
8
9
10      def sort_odds_evens_loop():
11          """sorts array using a loop"""
12          nums = [5, 6, 10, 16, 18, 24, 25, 30, 34, 36, 37
13                  64, 68, 75, 77, 80, 81, 83, 85, 88, 93,
14          odds = []
15          evens = []
16          for i in range(0, len(nums)):
17              if nums[i] % 2 == 0:
18                  evens.append(nums[i])
19              else:
20                  odds.append(nums[i])
21
22
23      import timeit
24      print(timeit.timeit(sort_odds_evens_lc, number=10000
25      print(timeit.timeit(sort_odds_evens_loop, number=100
```

The code on Lines 23 to 25 runs each function 10,000 times to get the average speed of execution for each showing the function with less code is more efficient.

We will look at the relative efficiency of the linear search and the binary search i

SEARCHING ALGORITHM

There are two types of searches that you will need to understand for your exam: searches. You will also need to understand the differences between them.

We have already talked about searching for names in a set of data records to fin will now look at the mechanics of this type of search in more detail.

LINEAR SEARCH

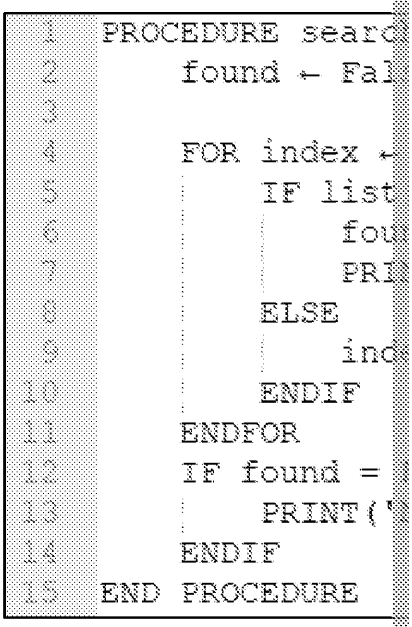
A linear search is the simplest type of search; it looks at each data item in your d item you are searching for OR reaches the end of the list.

Here is an example of a linear search using a small array of names.

0	1	2	3	4	5	6
Keiran	Taisha	Emily	Wyatt	Ryan	Zoe	Bethany

Note: The top row shows the INDEX value of each name in the array.

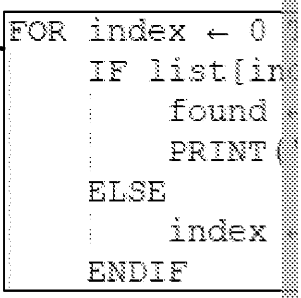
If we are searching to see if the name 'Zoe' is in our list, the algorithm will work like this:



```
PROCEDURE searc
    found ← Fal

    FOR index ←
        IF list
            fou
            PRI
        ELSE
            inc
        ENDIF
    ENDFOR
    IF found =
        PRINT('
    ENDIF
END PROCEDURE
```

Step 1:

0	1	2	3	4	5	6
Keiran	Taisha	Emily	Wyatt	Ryan	Zoe	Bethany



```
FOR index ← 0
    IF list[in
        found
        PRINT(
    ELSE
        index
    ENDIF
```

The algorithm starts looking at the data in **index** position 0 in the array.

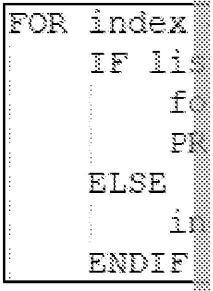
If the data item at that position, 'Keiran', matches the **name** 'Zoe', then the item has been found and the search will stop.

Step 2:
We can see that the data at index position 0 does not match the **name** 'Zoe' so t
ELSE part of the IF statement and executes Line 9.

The **index** value is now 1. The algorithm loops again and now checks **index** positi

0	1	2	3	4	5	6
Keiran	Taisha	Emily	Wyatt	Ryan	Zoe	Bethany

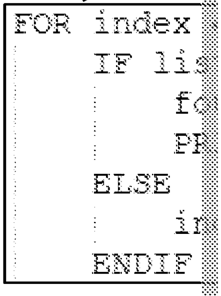
The FOR loop will continue to add 1 onto the **index** value each time the item in the list does not match the **name**.



```
FOR index
    IF lis
        fo
        PR
    ELSE
        in
    ENDIF
```

Step 3:
When the index value is equal to 5 the data item at that position in the array wil
found flag on Line 6 will be changed to True and the algorithm will output 'Found

0	1	2	3	4	5	6
Keiran	Taisha	Emily	Wyatt	Ryan	Zoe	Bethany

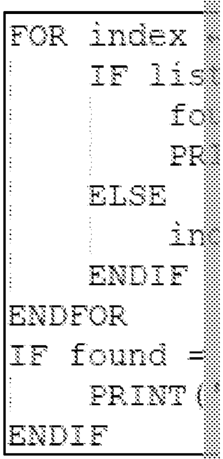


```
FOR index
    IF lis
        fo
        PR
    ELSE
        in
    ENDIF
```

Step 4:
The algorithm will continue to loop through the rest of the array to complete the FOR loop instructions.
The algorithm will then move to Line 12 and find that the value of **found** is True, and the algorithm will then finish.

Can you spot any inefficiency in this algorithm?

It should be clear that our algorithm should stop searching when the search item has been found and not continue to Step 4.



```
FOR index
    IF lis
        fo
        PR
    ELSE
        in
    ENDIF
ENDFOR
IF found =
    PRINT(
ENDIF
```

Complete Exercise 23: Fill in the Blanks
Complete Exercise 24: Linear Searches and Trace Tables

Binary Search

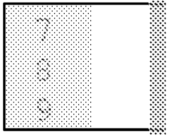
A binary search works by repeatedly reducing, splitting the data set into two halv
cannot contain the search item. This reduces the number of comparisons and th
efficiency of the algorithm.

Unlike a linear search algorithm, which will work whether the data is sorted into
will only work on an ordered list.

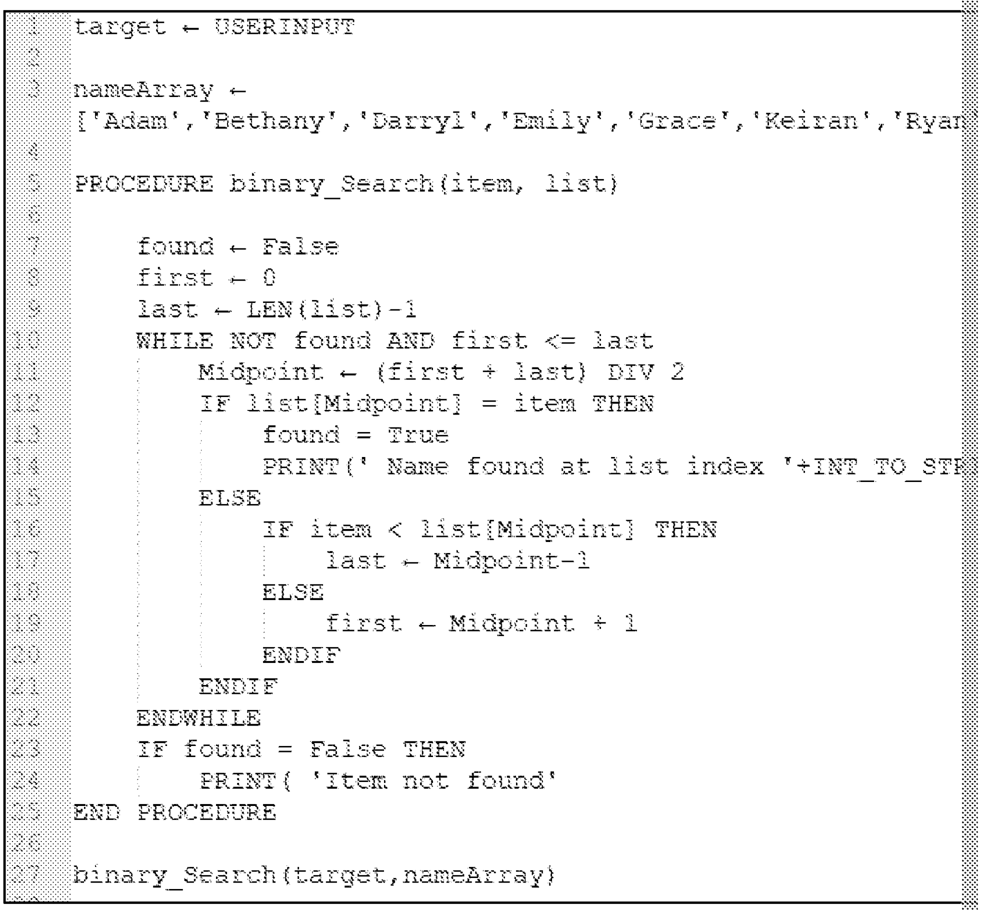
Here is our array of names; we are searching for 'Zoe' again.

0	1	2	3	4	5	6
Adam	Bethany	Darryl	Emily	Grace	Keiran	Ryan

Step 1:
The variable values for the search are set up in Lines 7, 8 and 9. This ensures that the search will only look at index positions from 0 to 9.

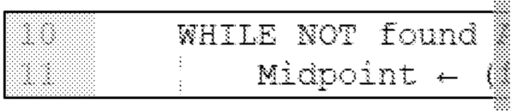


```
7
8
9
```



```
1   target ← USERINPUT
2
3   nameArray ←
    ['Adam','Bethany','Darryl','Emily','Grace','Keiran','Ryan
4
5   PROCEDURE binary_Search(item, list)
6
7       found ← False
8       first ← 0
9       last ← LEN(list)-1
10      WHILE NOT found AND first <= last
11          Midpoint ← (first + last) DIV 2
12          IF list[Midpoint] = item THEN
13              found = True
14              PRINT(' Name found at list index '+INT_TO_STR
15          ELSE
16              IF item < list[Midpoint] THEN
17                  last ← Midpoint-1
18              ELSE
19                  first ← Midpoint + 1
20              ENDIF
21          ENDIF
22      ENDWHILE
23      IF found = False THEN
24          PRINT( 'Item not found'
25  END PROCEDURE
26
27  binary_Search(target,nameArray)
```

Step 2:
The WHILE loop checks that the item has not been found AND that there are still
before setting the midpoint value. Line 11 adds 0 + 9 and then uses integer divisi
position 4.

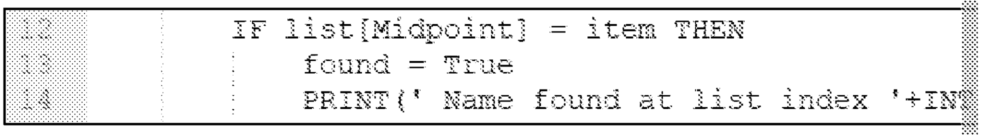


```
10      WHILE NOT found
11          Midpoint ← (
```

0	1	2	3	4	5	6
Adam	Bethany	Darryl	Emily	Grace	Keiran	Ryan

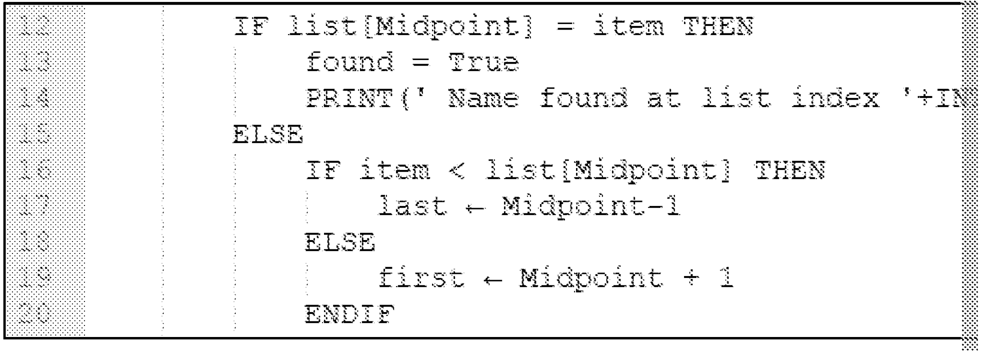
↑
Midpoint

Step 3:
The next step checks whether the item has been found and prints out a suitable
variable has changed to True, the conditions for the WHILE loop are no longer tr



```
IF list[Midpoint] = item THEN
    found = True
    PRINT(' Name found at list index '+IN
```

Step 4:
If the search item is not found here, then a check is made to see whether the ite
BELOW this starting midpoint value in the ordered list. The name we are looking
executed. The value of our variable **first** now becomes 5.



```
IF list[Midpoint] = item THEN
    found = True
    PRINT(' Name found at list index '+IN
ELSE
    IF item < list[Midpoint] THEN
        last ← Midpoint-1
    ELSE
        first ← Midpoint + 1
    ENDIF
```

0	1	2	3	4	5	6
Adam	Bethany	Darryl	Emily	Grace	Keiran	Ryan

These items are no longer part of the search

Step 5:
The item has not yet been found and there are still items to be searched in the li
The midpoint now evaluates to 7 ((5 + 9) DIV 2).

~~0~~	~~1~~	~~2~~	~~3~~	~~4~~	5	6
~~Adam~~	~~Bethany~~	~~Darryl~~	~~Emily~~	~~Grace~~	Keiran	Ryan

Step 6:
Step 3 is repeated again. As the item we are looking for is not in the midpoint, St
The midpoint now evaluates to 8 ((8 + 9) DIV 2).

~~0~~	~~1~~	~~2~~	~~3~~	~~4~~	~~5~~	~~6~~
~~Adam~~	~~Bethany~~	~~Darryl~~	~~Emily~~	~~Grace~~	~~Keiran~~	~~Ryan~~

Step 7:
Step 3 is repeated again. As the item we are looking for is not in the Midpoint, St
The midpoint now evaluates to 9 ((9 + 9) DIV 2).

~~0~~	~~1~~	~~2~~	~~3~~	~~4~~	~~5~~	~~6~~
~~Adam~~	~~Bethany~~	~~Darryl~~	~~Emily~~	~~Grace~~	~~Keiran~~	~~Ryan~~

Step 8:
Step 3 is repeated again. This time the data at index position 9 matches our sear
the WHILE loop as neither condition remains true.

LINEAR SEARCH VS BINARY SEARCH

COMPARISON CRITERIA	LINEAR SEARCH	
Advantages	(1) The data does not need to be sorted. (2) A linear search only needs access to the data to be sorted sequentially so less memory space is needed. (3) A linear search only needs to make equality comparisons.	For a will be
Disadvantages	A linear search is a sequential search. As the size of the array to be searched grows, the time taken to search will increase at the same rate.	(1) T (2) T a m (3) A c

Key Terms

Time efficiency The number of steps to complete the algorithm.

Space efficiency The amount of memory required to complete the algorithm.

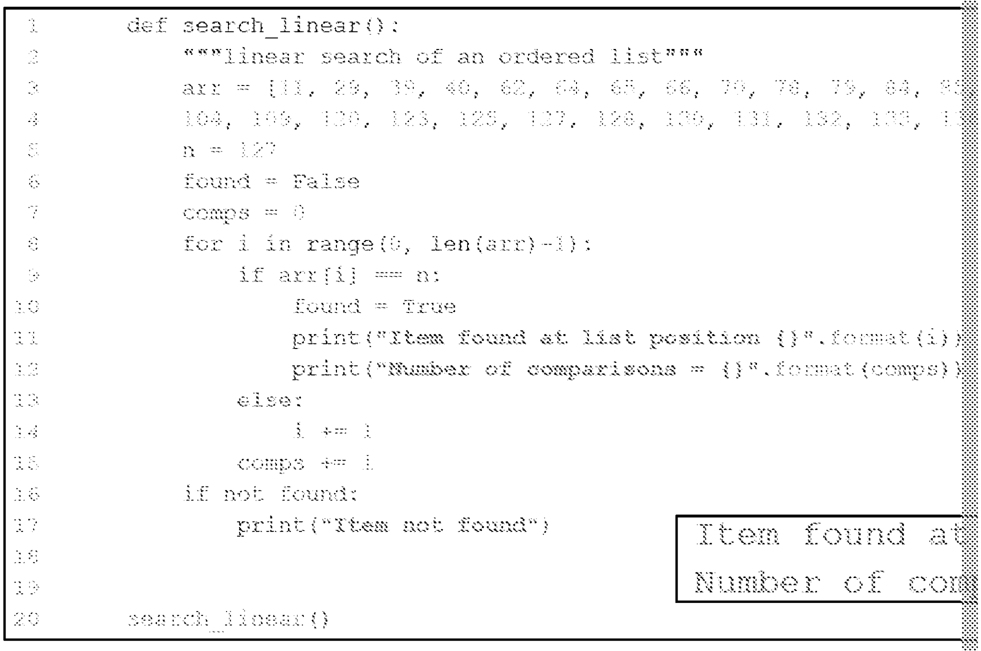
Brute force A process that tries all possible alternatives to find a solutio time to complete.

Linear search Used where data is unsorted. Each item in an array is comp item is found or the end of the array is reached.

Binary search Can only be used with a sorted array. Divides the array in search term with the 'midpoint' each time. The half which ca discarded. This continues until the item is found or the array

EFFICIENT SEARCHING PROOF

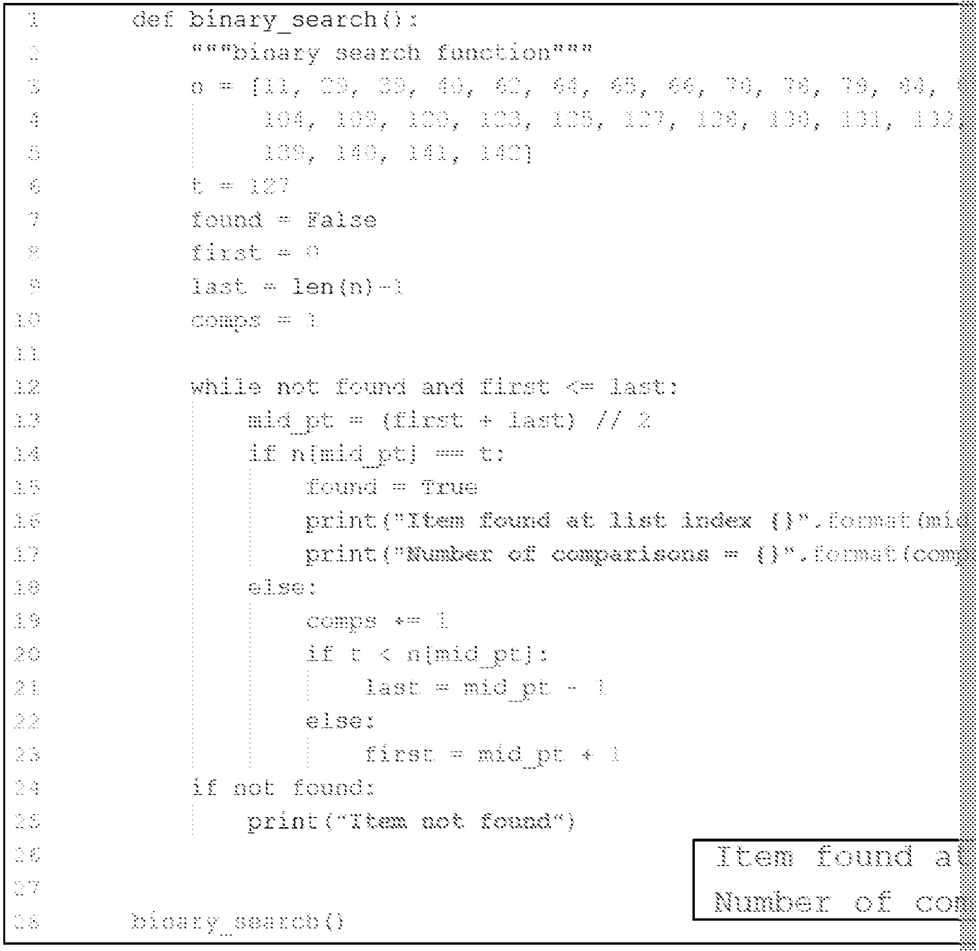
Again we can convert our search methods into Python to prove which is more ef
two searches perform by running some simple tests and checking the speed of e



```
1       def search_linear():
2           """linear search of an ordered list"""
3           arr = [11, 29, 39, 40, 62, 64, 65, 66, 70, 78, 79, 84, 8
4           104, 109, 120, 123, 125, 127, 128, 130, 131, 132, 133, 1
5           n = 127
6           found = False
7           comps = 0
8           for i in range(0, len(arr)-1):
9               if arr[i] == n:
10                  found = True
11                  print("Item found at list position {}".format(i))
12                  print("Number of comparisons = {}".format(comps))
13              else:
14                  i += 1
15              comps += 1
16          if not found:
17              print("Item not found")
18
19
20      search_linear()
```

```
Item found at
Number of com
```

The function has been modified to count the comparisons made in the search. E
the variable on Line 15 is incremented and the number is displayed when the ite



```
1       def binary_search():
2           """binary search function"""
3           n = [11, 29, 39, 40, 62, 64, 65, 66, 70, 78, 79, 84,
4               104, 109, 120, 123, 125, 127, 128, 130, 131, 132,
5               139, 140, 141, 142]
6           t = 127
7           found = False
8           first = 0
9           last = len(n)-1
10          comps = 1
11
12          while not found and first <= last:
13              mid_pt = (first + last) // 2
14              if n[mid_pt] == t:
15                  found = True
16                  print("Item found at list index {}".format(mi
17                  print("Number of comparisons = {}".format(com
18              else:
19                  comps += 1
20                  if t < n[mid_pt]:
21                      last = mid_pt - 1
22                  else:
23                      first = mid_pt + 1
24          if not found:
25              print("Item not found")
26
27
28      binary_search()
```

```
Item found a
Number of co
```

The number of comparisons needed to find the same item in the same array is m
than the linear search. This means that as the size of the array increases the amo
grow making the linear search slower than a binary search. If we use the **timeit(**
clear (Linear = 0.073… vs Binary = 0.024…)

SORTING ALGORITHMS

There are two sorting methods that you need to understand for your exam: the b sort.

The simplest type of sorting algorithm is the bubble sort.

BUBBLE SORT: HOW IT WORKS

The bubble sort works on an array of data; these could be integers, real numbers

1. Starting at the beginning of the array (index position 0), the first element element.
2. If the first element is larger than the next element, the two are swapped.
3. Move one element to the right and compare the current element with th
4. Repeat Step 2 and Step 3 until the end of the array is reached.
5. If no swaps have been made in the comparisons of the elements in the ar
6. If not, repeat Steps 1 to 5 again.

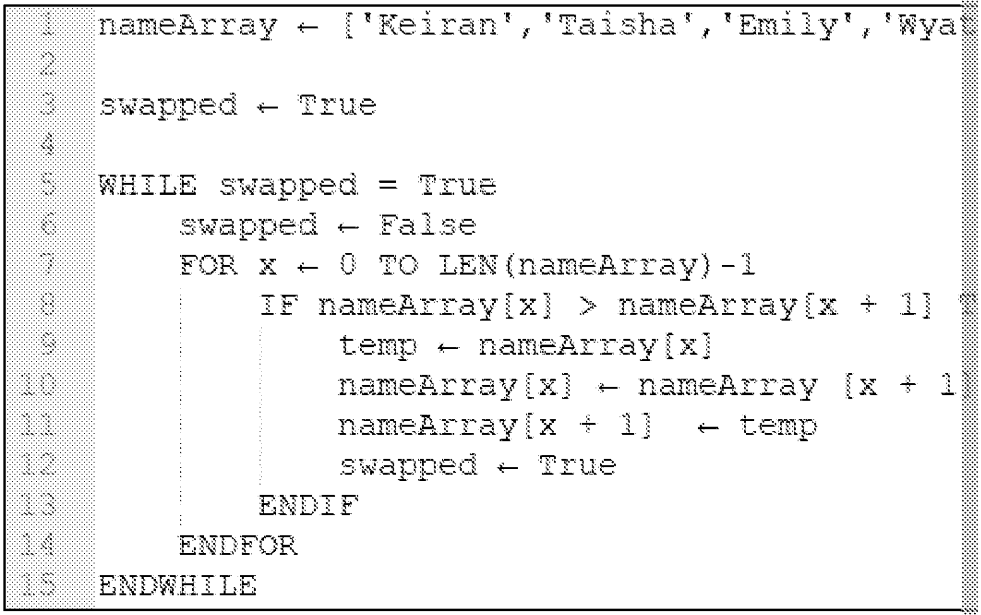
Simple example:

5	1	12	-5	16	UNSORTED
5	1	12	-5	16	5>1, SWAP 5 and 1
1	5	12	-5	16	5 < 12, OK
1	5	12	-5	16	12> -5, SWAP 12 and -5
1	5	-5	12	16	12 < 16, OK
					START AGAIN
1	5	-5	12	16	1<5, OK
1	5	-5	12	16	5 > -5, SWAP 5 and -5
1	-5	5	12	16	5 < 12, OK
					START AGAIN
1	-5	5	12	16	1 > -5, SWAP 1 and -5
-5	1	5	12	16	1 < 5, OK
					START AGAIN
-5	1	5	12	16	-5 <1, OK
-5	1	5	12	16	SORTED

PSEUDOCODE FOR THE BUBBLE SORT

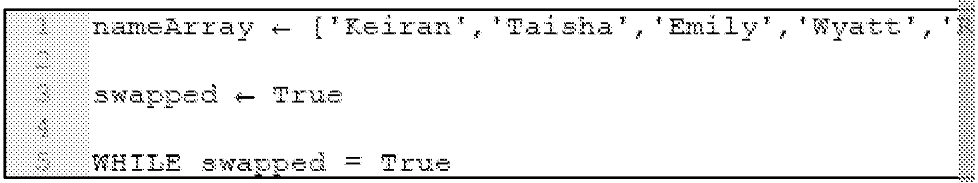
This example will use an array of names; we will look at how the algorithm works

0	1	2	3	4	5
Keiran	Taisha	Emily	Wyatt	Ryan	Zoe



```
1   nameArray ← ['Keiran','Taisha','Emily','Wya
2
3   swapped ← True
4
5   WHILE swapped = True
6       swapped ← False
7       FOR x ← 0 TO LEN(nameArray)-1
8           IF nameArray[x] > nameArray[x + 1]
9               temp ← nameArray[x]
10              nameArray[x] ← nameArray [x + 1
11              nameArray[x + 1]  ← temp
12              swapped ← True
13          ENDIF
14      ENDFOR
15  ENDWHILE
```

Step 1:
A flag variable called 'swapped' is used to control the WHILE loop and determine
i.e. when a pass has been made and there were no swaps needed. This is initially
control the WHILE loop.

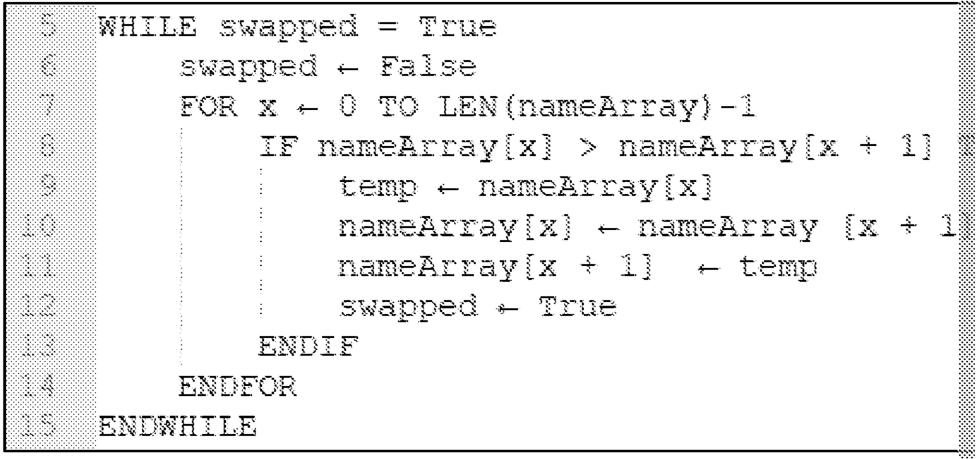


```
1   nameArray ← ['Keiran','Taisha','Emily','Wyatt','
2
3   swapped ← True
4
5   WHILE swapped = True
```

Step 2:
Line 6 sets the value of the 'flag' swapped to False. This means that if the array is
running the WHILE loop is no longer true and the sort will end.

Line 8 starts to compare the array items in index positions 0 and 1. In this example,
alphabet, so no swap is needed and the code moves to Line 14 and the value of x is

0	1	2	3	4	5
Keiran	Taisha	Emily	Wyatt	Ryan	Zoe



```
5   WHILE swapped = True
6       swapped ← False
7       FOR x ← 0 TO LEN(nameArray)-1
8           IF nameArray[x] > nameArray[x + 1]
9               temp ← nameArray[x]
10              nameArray[x] ← nameArray [x + 1
11              nameArray[x + 1]  ← temp
12              swapped ← True
13          ENDIF
14      ENDFOR
15  ENDWHILE
```

Step 3:
The WHILE loop iterates again with x being incremented (increased) to have the between index items 1 and 2.

As Emily comes before Taisha in the alphabet, Line 9 stores the value at index po Line 10 puts the data at index position 2 into index position 1. Line 11 copies the index position 2. The ‘flag’ variable **swapped** is also changed to True as a swap h

The array now looks like this:

0	1	2	3	4	5
Keiran	Emily	Taisha	Wyatt	Ryan	Zoe

The process continues until all the pairs have been compared. This is called the f algorithm. The array has changed as shown here.

0	1	2	3	4	5
Keiran	Taisha	Emily	Wyatt	Ryan	Zoe

0	1	2	3	4	5
Keiran	Emily	Taisha	Wyatt	Ryan	Zoe

0	1	2	3	4	5
Keiran	Emily	Taisha	Wyatt	Ryan	Zoe

Result of first **pass** of the bubb

0	1	2	3	4	5
Keiran	Emily	Taisha	Ryan	Wyatt	Zoe

0	1	2	3	4	5
Keiran	Emily	Taisha	Ryan	Wyatt	Zoe

As you can see, the last two items are now in order in the correct position. The W the value of our ‘flag’ **swapped** is still equal to True.

The bubble sort algorithm will need to make several passes or traversals of the a

SECOND PASS

0	1	2	3	4	5
Emily	Keiran	Taisha	Ryan	Wyatt	Zoe

0	1	2	3	4	5
Emily	Keiran	Taisha	Ryan	Wyatt	Zoe

0	1	2	3	4	5
Emily	Keiran	Ryan	Taisha	Wyatt	Zoe

0	1	2	3	4	5
Emily	Keiran	Ryan	Taisha	Wyatt	Zoe

0	1	2	3	4	5
Emily	Keiran	Ryan	Taisha	Wyatt	Zoe

FINAL PASS

0	1	2	3
Emily	Keiran	Taisha	Ryan

0	1	2	3
Emily	Keiran	Taisha	Ryan

0	1	2	3
Emily	Keiran	Ryan	Taisha

0	1	2	3
Emily	Keiran	Ryan	Taisha

0	1	2	3
Emily	Keiran	Ryan	Taisha

Why is the final pass needed when the data is all sorted after the second pass?

The second pass involved a swap between Ryan and Taisha which left the ‘flag’ v algorithm must run one last time to prove that no more swaps are needed befor

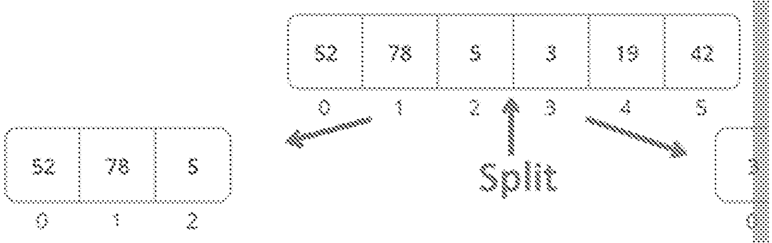
Complete Exercise 25: Bubble Sort Exercises
Complete Exercise 26: Put the Bubble Sort Flow Chart in Order

MERGE SORT: HOW IT WORKS

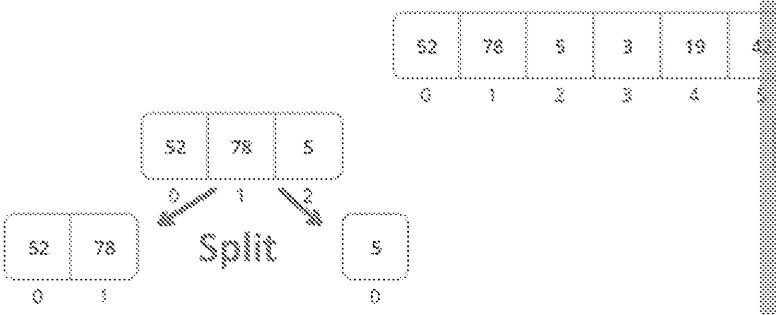
The merge sort is much more complex than the bubble sort and is known as a 'di
splits up the data array to be sorted into smaller sub-arrays until the sub-array h
arrays are then sorted and recombined into a sorted array.

Example:

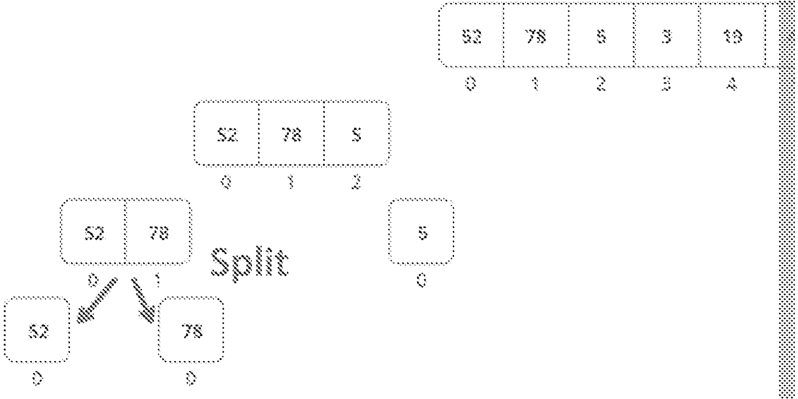
Step 1: Split the array in half at the midpoint.



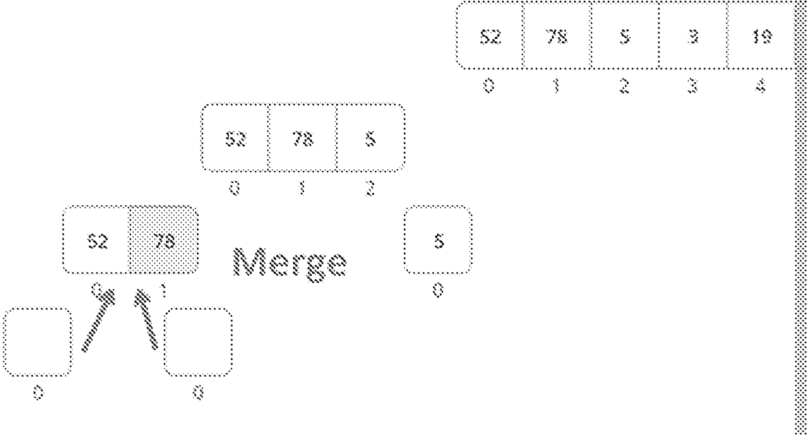
Step 2: Select the left sub-array and split again.



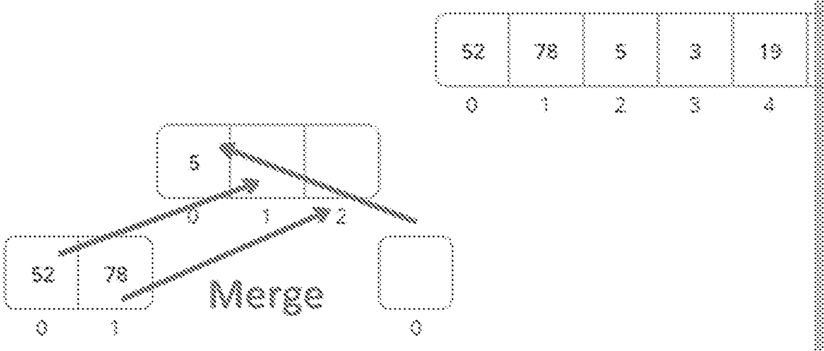
Step 3: Select the left sub-array and split again so that the sub-array has just one



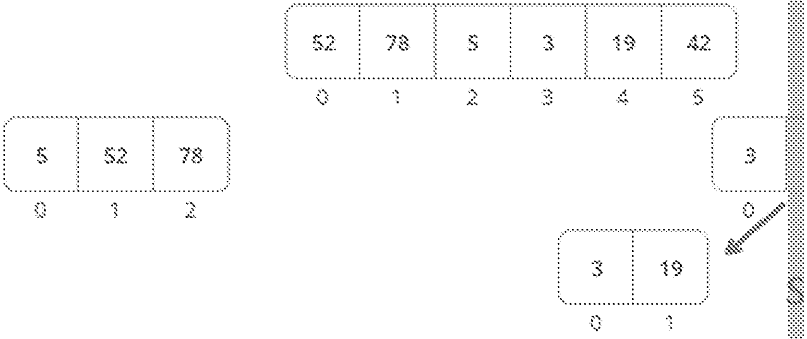
Step 4: Merge the sorted data back into an array.



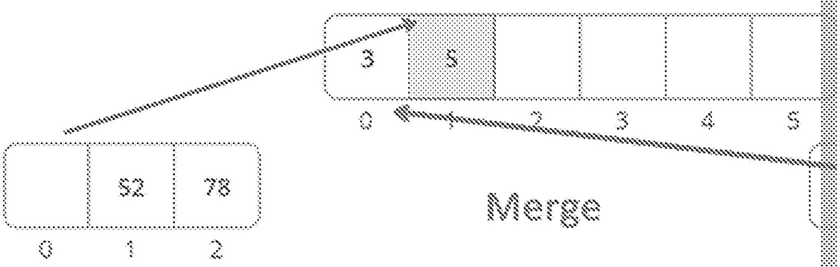
Step 5: Combine the sorted array and merge together with the smallest item firs



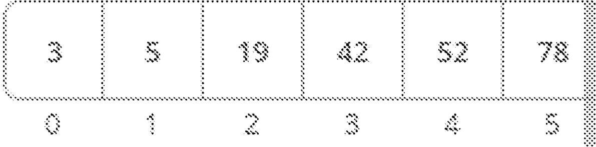
Step 6: Repeat the process with the right sub-array.



Step 7: When each sub-array has been sorted, the sub-arrays are then merged b by comparing the values in each sub-array and choosing the smallest.



Step 8: When all data has been merged back into the original array, the data is s

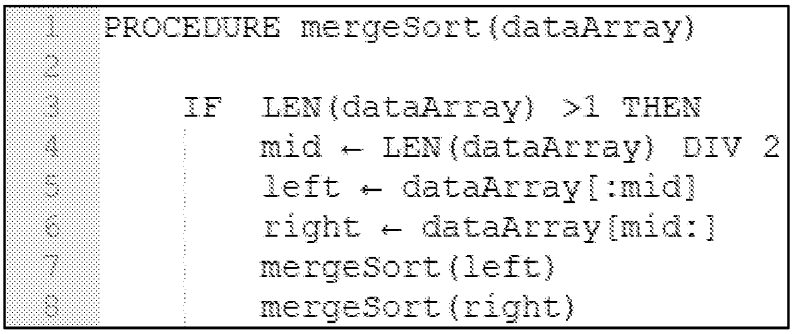


Merge sort summary

1. Divide the original array into two sub-arrays
2. Continue dividing all sub-arrays until they have just one element
3. Compare the element in the left sub-array with the element in the right
4. Add the smallest to the new array
5. Move to the next element in the sub-array you just used
6. If the sub-array is empty, add all elements from the other sub-array in th
7. Otherwise, repeat from 3 until one list is empty

PSEUDOCODE FOR THE MERGE SORT

We will use a simple array of numbers: [63, 12, 5, 27, 31, 45].



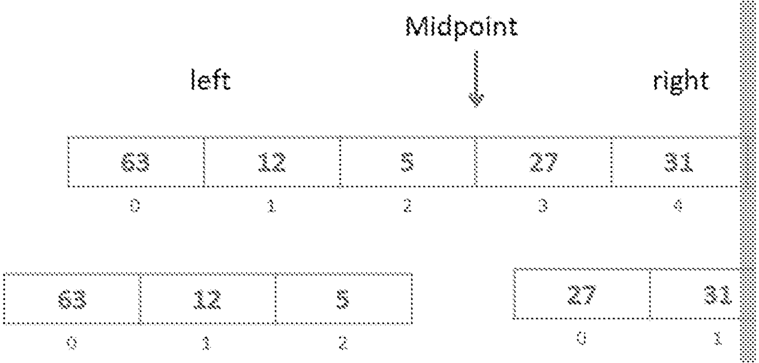
```
1  PROCEDURE mergeSort(dataArray)
2
3      IF  LEN(dataArray) >1 THEN
4          mid ← LEN(dataArray) DIV 2
5          left ← dataArray[:mid]
6          right ← dataArray[mid:]
7          mergeSort(left)
8          mergeSort(right)
```

Step 1: Line 3 checks if the array is larger than one, if not the data is already sort the array and uses that value to split the dataArray into a left half and a right hal to two variables.

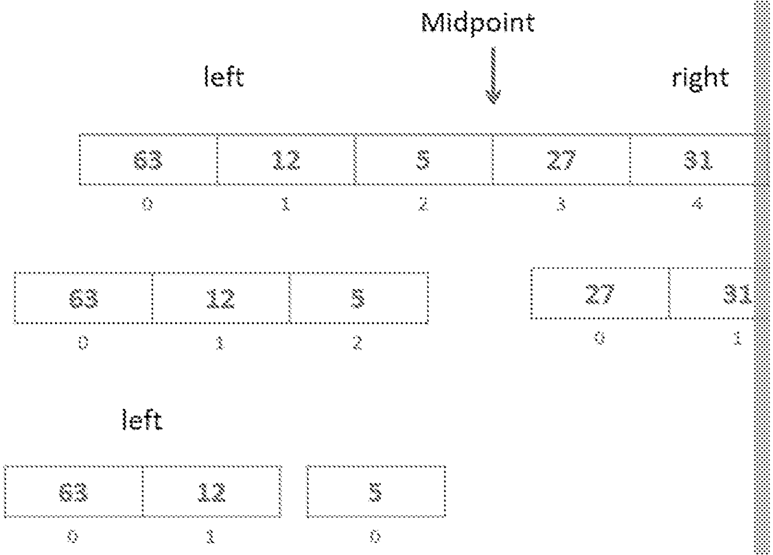
Step 2: This example uses a programming construct we have not yet seen called splitting a problem down into smaller versions of the same problem by calling th same subroutine on Lines 7 and 8.

In this case we are making the problem smaller by calling the **mergeSort** subrout dataArray into the subroutine as the **parameter**.

Here is the current state of our array, **dataArray**:

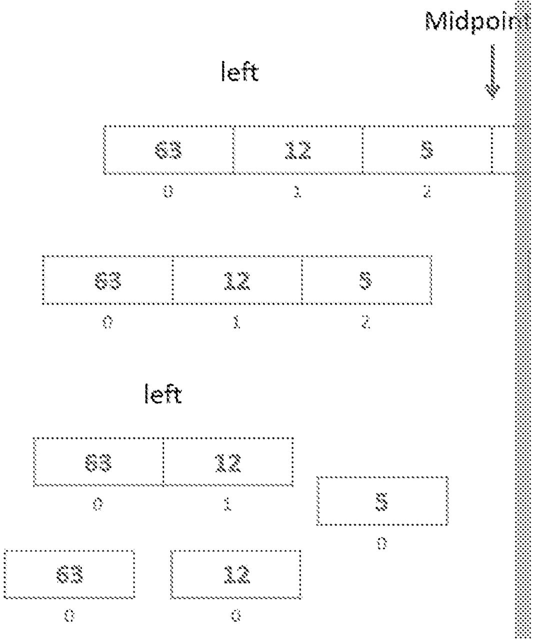


Step 3: The process is now repeated from Line 3 as the **mergeSort subroutine** is using the **left** part of the original **dataArray**. The current state of the array now split in half again.



Step 4: The process is repeated again as the **mergeSort** subroutine is called again in Line 7 now using the new **left** array, which has just two numbers in it.

The data on the **left** is now ready for merging back into order.



Step 5: The next part of the subroutine now sorts the data back into order, starti

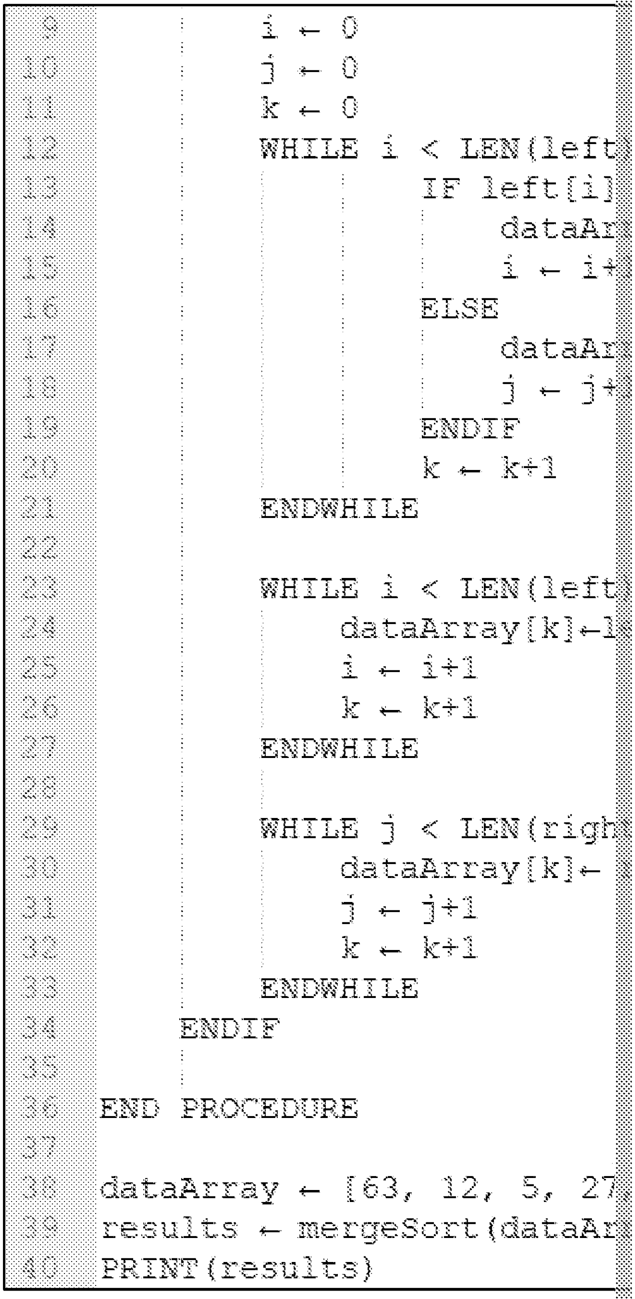
The subroutine uses three index variables, i, j and k, on Lines 9 to 11 to set the starting points for comparing and sorting the data back into the original **dataArray**.

The comparison will start with 63 and 12. 63 is greater than 12 (see Line 17) so it becomes the first item copied back into the **dataArray**.

The values of variables i, j and k are also incremented (Lines 15, 18 and 20).

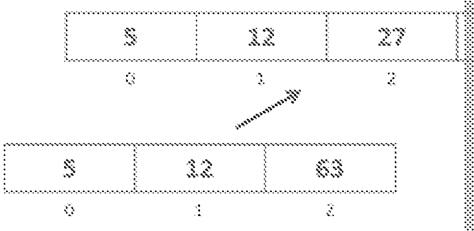
Step 6: When the data on the left is sorted, the process of splitting and sorting will be repeated for the data on the right.

***Note**: Even though it looks as if the data array on the right is already sorted (the numbers in the right side of the array just happened to be in order), the whole process must be repeated.*



```
9          i ← 0
10         j ← 0
11         k ← 0
12         WHILE i < LEN(left
13                 IF left[i]
14                     dataAr
15                     i ← i+
16                 ELSE
17                     dataAr
18                     j ← j+
19                 ENDIF
20                 k ← k+1
21         ENDWHILE
22
23         WHILE i < LEN(left
24             dataArray[k]←l
25             i ← i+1
26             k ← k+1
27         ENDWHILE
28
29         WHILE j < LEN(righ
30             dataArray[k]←
31             j ← j+1
32             k ← k+1
33         ENDWHILE
34     ENDIF
35
36 END PROCEDURE
37
38 dataArray ← [63, 12, 5, 27
39 results ← mergeSort(dataAr
40 PRINT(results)
```

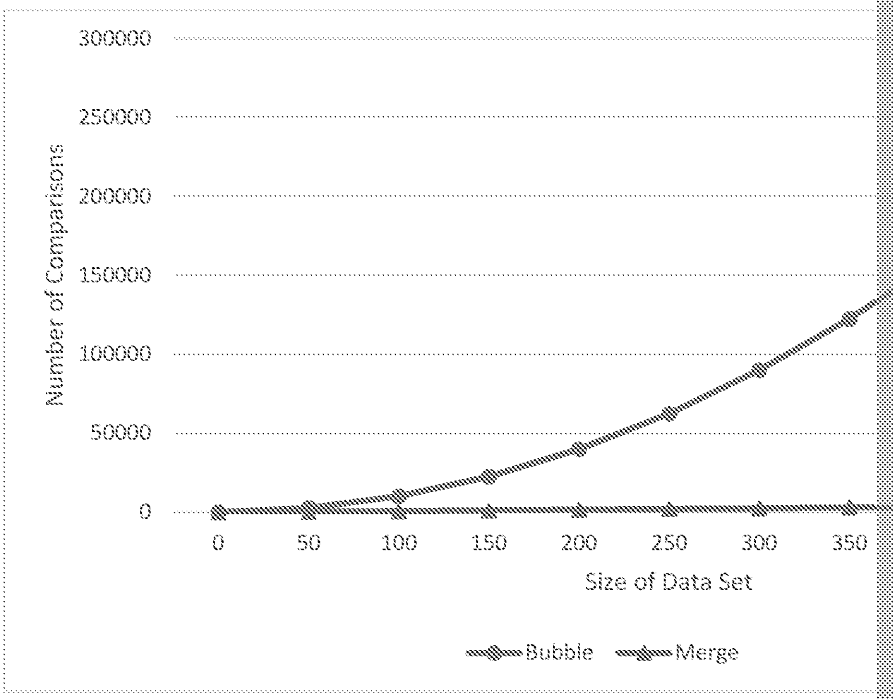
Step 7: The left and right arrays are now finally sorted into order by running the algorithm from Line 13 through to the end by comparing each item in the two arrays before copying them into the sorted **dataArray**.



Bubble Sort vs Merge Sort

Comparison criteria	Bubble sort	
Advantages	(1) Very simple algorithm, easy to code. (2) Uses much less memory than a merge sort.	Muc rega
Disadvantages	Slower algorithm than the merge sort.	(1) (2)

The chart below compares the bubble sort and the merge sort for the same data sort is very efficient, regardless of the size of the data being sorted. The bubble s inefficient as the size of the data to be sorted grows.



Key Terms

Bubble sort The sort works by comparing and swapping each pair of ite are in order. This may takes several passes through the arra

Pass Each process of working through an array is known as a 'pa

Merge sort This sort divides an array into smaller and smaller sub-arra sub-arrays are then merged back in the correct order.

Divide and conquer This is the term given to algorithms (searches and sorts) whic sub-problems which are easier to solve by using recursion. T subroutine inside itself as part of the subroutine.

Example exam question & Solution:

Using the data array [6, 3, 9, 2, 7], demonstrate how the data would be sorted us showing each stage in the sorting process.

Bubble Sort: Solution

6	**3**	**9**	**2**	**7**
3	6	9	2	7
3	6	2	9	7
3	6	2	7	9
3	2	6	7	9
2	3	6	7	9

Merge Sort: Solution

| 6 | 3 | 9 | 2 | 7 |

| 6 | 3 | 9 | | 2 | 7 |

| 6 | 3 | | 9 | | 2 | | 7 |

| 6 | | 3 |

Split the array into smallest elements

Merge the smallest elements back in order

| 3 | 6 | | 9 | | 2 | 7 |

| 3 | 6 | 9 | | 2 | 7 |

| 2 | 3 | 6 | 7 | 9 |

Complete Exercise 27: Sorting and Searching
Complete Crossword Five