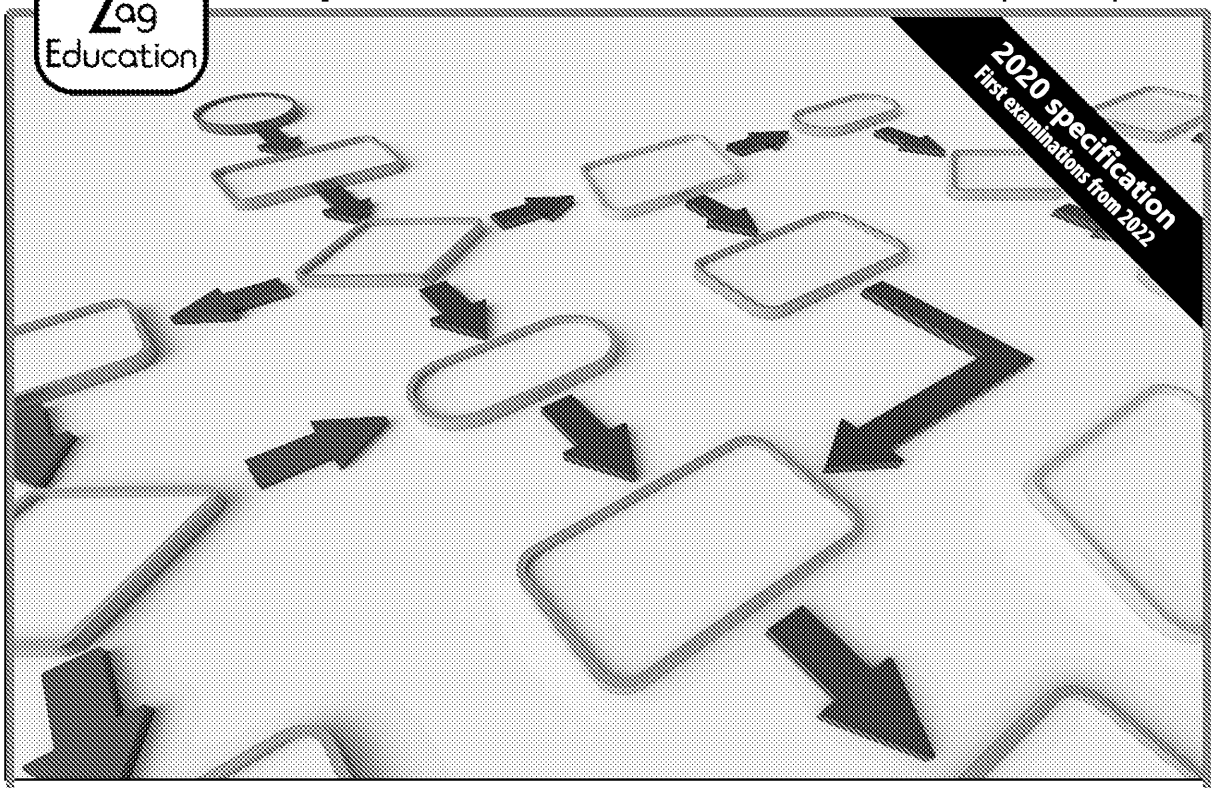
Zig Zag Education

Computer Science GCSE (9–1) | OCR | J277



Algorithms Resource Pack

for OCR GCSE Computer Science (J277)

Sue Wright

Part 1 – Theory

zigzageducation.co.uk **POD 10603a**

Contents

Product Support

Get Product Updates

Occasionally we make improvements to resources after you receive them. Download the updated content **free of charge**.

Download Support Files

Any support files for your purchased resources are available for download. This includes HTML links pages for resources that contain lots of hyperlinks.

Send us Your Feedback

For every completed review, **get a £10 voucher** to use on your next order! Tell us what you thought, and report any issues or ideas for improvement.

Get New Resource Notifications

Opt in to receive email alerts about new resources for your subject(s).

Register today via:

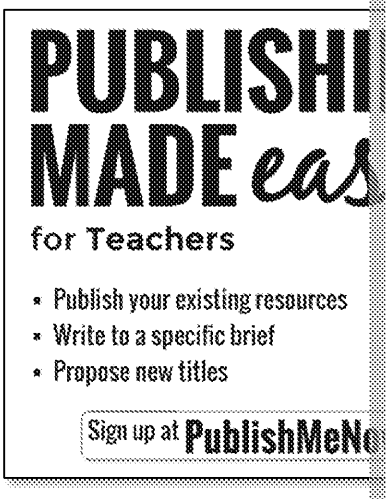
ZigZagEducation.co.uk → **Computer Science & IT** → **Pro**

Quick link: zzed.uk/PS

INSPECTION COPY

Ever considered publishing your work?

Join PublishMeNow, our teacher-author website, today!



Zig Zag Education

Terms and Conditions of Us

Terms and Conditions

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

> **"You acknowledge that you rely on your own skill and ju**
> **determining the suitability of the Goods for any particula**

"We do not warrant: that any of the Goods are suitable for any particular pur
qualification), or the results that may be obtained from the use of any publica
that we are affiliated with any educational institution, or that any publication
sponsored by or endorsed by any educational institution."

Copyright Information

Every effort is made to ensure that the information provided in this publication is ac
responsibility is accepted for any errors, omissions or misleading statements. It is Zi
permission for any copyright material in their publications. The publishers will be gl
with any copyright holders whom it has not been possible to contact.

Students and teachers may not use any material or content contained herein and in
without referencing/acknowledging the source of the material ("Plagiarism").

Disclaimers

This publication is designed to supplement teaching only. Practice questions may b
specification and may also attempt to prepare students for the type of questions th
will not attempt to predict future examination questions. ZigZag Education do not
that may be obtained from the use of this publication, or as to the accuracy, reliabil

Where the teacher uses any of the material from this resource to support examinat
ensure that they are happy with the level of information and support provided pert
and to the constraints of the specification and to others involved in the delivery of
that the teacher adapt, extend and/or censor any parts of the contained material
specification and the needs of the individual or group concerned. As such, the tea
the material, if any, to provide to the students and which parts to use as back
Likewise, the teacher must determine what additional material is required to cover
cover each specification point to the correct depth.

ZigZag Education is not affiliated with Pearson, Edexcel, OCR, AQA, WJEC, Edu
Baccalaureate Organization or DFE in any way nor is this publication authorised b
endorsed by these institutions unless explicitly stated on the front cover of this pub

Teacher's Introduction

This resource explores OCR GCSE Computer Science (J277) specification section ***2.1 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 examples, using OCR Exam Reference Language, explanations and diagrams which explain the stages of the searching and sorting algorithms and how the Exam Reference Language relates to the process of each algorithm.

Students are shown how to plan algorithms using both flowcharts and OCR specific Exam Reference Language, 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 ***2.2 Programming Techniques***, where much of the content (such as variables, arrays, subprograms and operators) is covered naturally while looking at algorithms.

This resource will be invaluable in giving students a detailed introduction to the use of the OCR Exam Reference Language and code segments that could 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 TH

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 tomato
can use to explain an everyday algorithm in terms of its **inputs**, **process** and **outp**

INPUTS

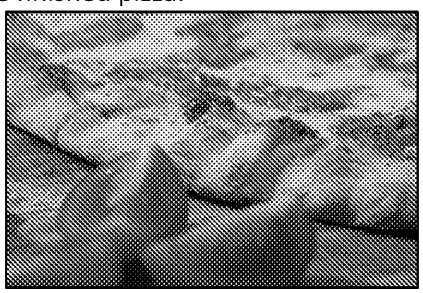
280 g packet white bread or pizza base mix
1–2 tsp olive oil
150 g cherry tomatoes, halved
125 g ball mozzarella, drained
75 g cubed pancetta or bacon
bunch of basil

PROCESS

1. Empty the pizza c
chop the basil an
along with 200 m
for five minutes,
shape.
2. Transfer to a flou
surface with olive
until risen. Push t
up, into the doug
them.
3. Dry the mozzarell
paper, and then c
pizza along with t
bake for 20 minu
golden. Tear over

OUTPUT

The finished pizza:



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

INPUTS

Your location and destination.

OUTPUT

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

PROCESS

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

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 th 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?

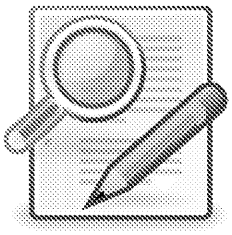
Key Terms

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.

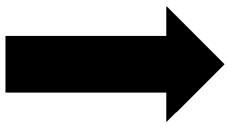
ALGORITHMS VS PROGRAMS

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

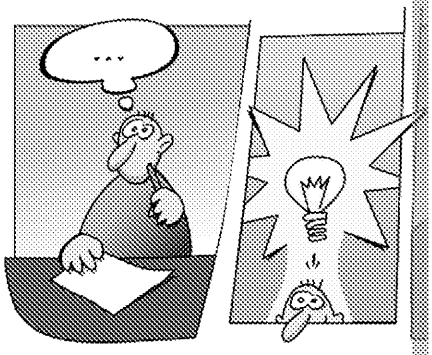
Example
You have two young cousins who struggle to learn their spellings each week; you
them.



Analyse the problem



Make a game based o
idea of 'Look, cover, w



Design and test the alg

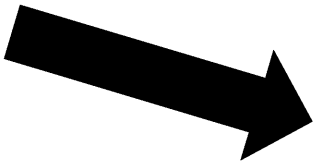
What are the inputs? How do they get into the game?

How will the game show the words to be spelt?

How will the game check the spelling that is entered?

How will the game store the number of right answers?

How will the player know if all have been spelt correctly?



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 NEE

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.

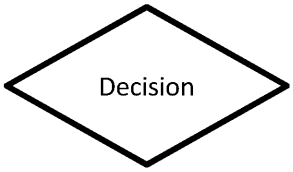
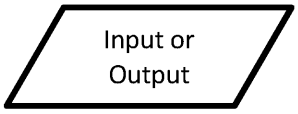
Example:

```
1  WordToGuess = "twelve"
2
3
4  array CorrectGuess [5]
5  CorrectGuess[0] = ["e"]
6
7  array WrongGuess [10]
8
9  WrongGuess [0]= "a"
10 WrongGuess [1]= "o"
11 WrongGuess [2]= "i"
12 WrongGuess [3]= "g"
13 WrongGuess [4]= "s"
```

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

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. Tl 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

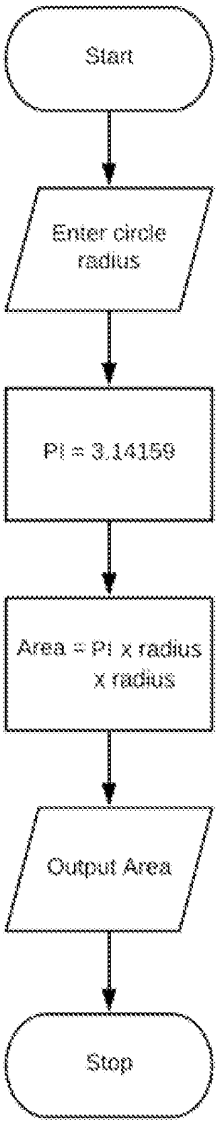
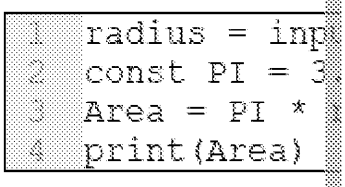
PSEUDOCODE

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 v
pseudocode in your exam; this booklet will use the exam board version in all exa

Look at the two examples of the same algorithm below.

Flow chart	Pse
	


```
1  radius = inp
2  const PI = 3
3  Area = PI *
4  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 understa
and they MUST be unique; you cannot use the same **identifier** or name for differ
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
by the **identifier** in capital letters. Values are **assigned** to a **constant** in exactly th
variable; 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

GETTING INPUTS AND OUTPUTS

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

```
1  radius = input("Enter Radius")
2  const PI = 3.14159
3  Area = PI * radius * radius
4  print(Area)
```

The two keywords used here are:

- INPUT
  - Getting values into the algorithm from the user (via keyboard).
- 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	PSEUDOC
Addition +	+	5 + 6 evaluates to 11
Subtraction –	–	7 – 3 evaluates to 4
Multiplication ×	*	4 * 2 evaluates to 8
Division ÷	/	12/3 evaluates to 4
Exponentiation	^	5^3 evaluates to (5 × 5 = 25, 5 *In this example 5 is the **base** a* 2^4 = 16 (2 × 2 = 4, 2 × 4 = 8,
Integer division (only evaluates the **quotient** from the division)	DIV	9 DIV 6 evaluates to 1 *This evaluates to 9 ÷ 6 = 1 rem* *The **quotient** is the only outpu*
Modulus operator (only evaluates the **remainder** from the division)	MOD	10 MOD 3 evaluates to 1 *This evaluates to 10 ÷ 3 = 3 re* *The **remainder** is the only outp*

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

Comparison / Relational Operators

These are called comparison operators as they are used to compare two values. A relational operators will evaluate to either True or False.

Operator	What it means	Example
<	Less than	5 < 7
>	Greater than	3 > 12
==	Equality operator checks whether both values are the same	5 == 5
!=	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	For example, passv be >= 8 characters
OR	Logical OR checks whether EITHER of the conditions is true	Passwords must in
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 NO NOT including a nu

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

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
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, ou
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 sho something you want to do with the values.
Quotient	When a number is divided by another, the result is known for 12 × 3 = 4, the quotient is 4.
Div	Integer division gives only the quotient and ignores any re
Mod	The modulus operation finds the remainder only after divis 15 MOD 6 = 3.
Exponentiation	This means that a base integer is raised to the power of th 3^3 evaluates to 3 × 3 × 3 = 27
Comparison operator	This is used in programming to compare two values, e.g. 4 using comparison operators will evaluate to either True or
Boolean operator	A Boolean or logical operator is used to combine condition 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.

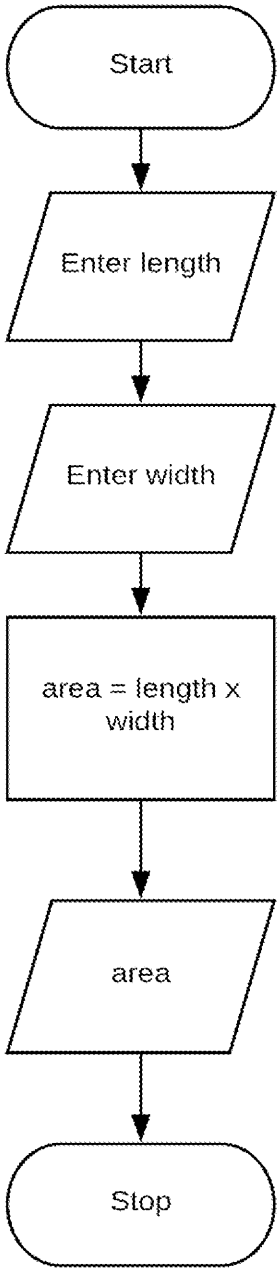
The three constructs are:

- Sequence
- Selection
- Iteration

SEQUENCE

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

FLOW CHART	PSEU

	`length = inpu`
	`width = input`
	`area = length`
	`print(area)`

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

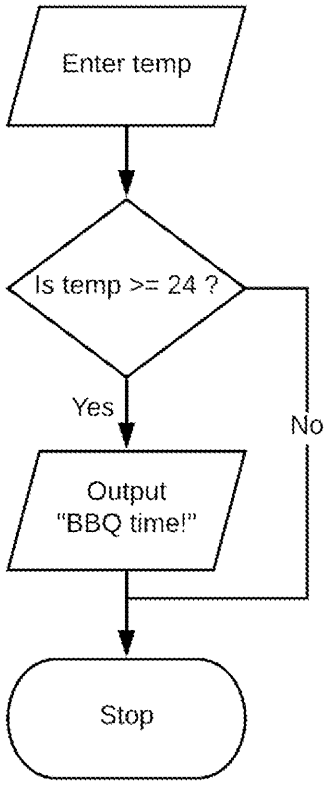
IF STATEMENT	

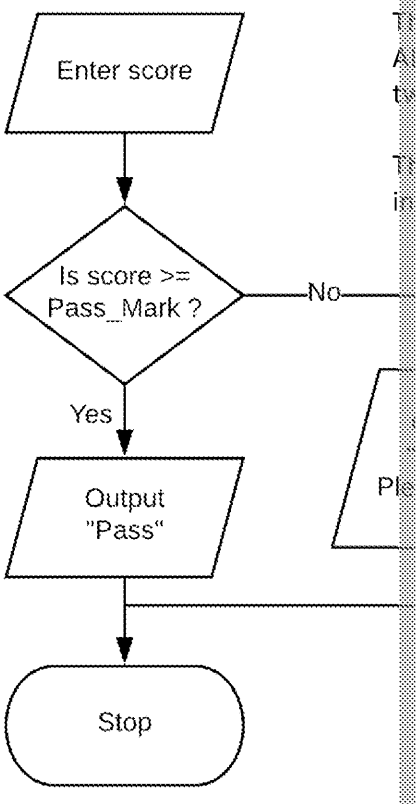
IF < condition > THEN
  <statement when condition is true>
ENDIF
	1 temp = 25
	2 if temp >= 24
	3 print ("bb
	4 endif

IF- ELSE STATEMENT	

IF < condition > THEN
  <statement when condition is true>
ELSE
  <statement when condition is false>
ENDIF
	1 pass_mark = 60
	2 score = input ("E
	3
	4 if score >= pass_
	5 print("Pass")
	6 else
	7 print("Failed
	8 endif

SELECTION USING FLOW CHARTS





Complete Exercise 6: Odds or Evens

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

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

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
```

```
1  score = input("Enter
2
3  if score >= 80 then
4      print("Levels 7
5  elseif score <= 79 A
6      print("Levels 4
7  elseif score <= 59 A
8      print("Levels 1
9  else
10     print("Failed. P
11 endif
```

Example 1:

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

Example 2:

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

As you can see in the example trace table above, the **ELSEIF** statement will not e
one statement has been evaluated to True; Lines 7 to 10 would be ignored and
next line of code on Line 11.

The same result can also be achieved using a switch/case statement. Again, if th
then none of the other conditions are tested.

SWITCH/CASE STATEMENT

```
SWITCH  <input>

CASE <condition>:
   <statement>
CASE <condition>:
   <statement>
DEFAULT:
   <statement>
ENDSWITCH
```

```
 1  score = input("enter scor
 2
 3  switch score:
 4      case >= 80:
 5          print("Levels 7 -
 6      case >= 60:
 7          print("Levels 4 -
 8      case <= 59:
 9          print("Levels 1 -
10      default:
11          print("Failed. Pl
12  endswitch
```

Complete Exercise 7: Colour Range

TRACE TABLES

A trace table is a useful way of checking the logic of your algorithm BEFORE you s
code. It involves using a range of data to check that the algorithm you have writt
per the example table shown on the previous page. This type of checking is alwa
performing a 'dry run' of the algorithm.

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 (INDEFINITE ITERATION)

DO... WHILE LOOP

DO
    <statements>
WHILE <Boolean expression>

Example 1

```
1  password = input("Enter passwor
2
3  do
4      confirm = input("confirm pa
5  until confirm == password
```

This will continue to ask for the user to confirm

Example 2

```
1  count = 10
2  do
3      print(count)
4      count = count -1
5  until count == 5
```

This will print out 10, 9, 8, 7, 6 and then stop

WHILE... LOOP

WHILE <Boolean expression>
    <statements>
ENDWHILE

Example 1

```
1  password = input("Enter
2
3  confirm = input("confir
4
5  while confirm != passwo
6      print("entry does n
7      confirm = input("co
8  endwhile
```

In this example, the condition is checked at the
If the password and confirm entries DO match

Example 2

```
1  count = 10
2
3  while count > 5
4      print(count)
5      count = count -1
6  endwhile
```

Again, the condition is checked at the start of
be different from the REPEAT... UNTIL... loop e
numbers 10, 9, 8, 7, 6.

COUNT-CONTROLLED (DEFINITE ITERATION)

	FOR... NEXT
FOR identifier = IntExp TO IntExp <condition> next identifier	1 for count = 1 to 10 2 print(count) 3 next count *This will output 1,2,3,4,5,6,7,8,9,10.*
FOR identifier = IntExp TO IntExp STEP <condition> next identifier	1 for count = 0 to 20 2 print(count) 3 next count *This will output 0,5,10,15,20.*

Key Terms

Construct	The basic building blocks of an algorithm or program that c instructions are executed.
Sequence	When instructions are executed, in order, one after another
Selection	Also known as a conditional statement, this allows the algori instructions based on whether a condition is True or False.
Iteration	Instructions are repeated either until a condition is True or F times.
Trace table	A manual method of testing an algorithm to ensure there ar

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:

```
1  for x = 1 to 101
2      if x MOD 3 == 0 AND x MOD 5 == 0 then
3          print("FizzBuzz")
4      elseif x MOD 5 == 0 then
5          print("Buzz")
6      elseif x MOD 3 == 0
7          print("Fizz")
8      else
9          print(x)
10     endif
11 next x
```

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
3  guessed = false
4  target = 11
5
6  while guessed != true
7      number = input("enter a number between
8
9      while number <= 0 OR number > 20
10         number = input("number out of range,
11     endwhile
12
13     if number == target then
14         print("well done, you guessed it!")
15         guessed = true
16     elseif number > target then
17         print("Too high")
18     else
19         print("Too low")
20     endif
21 endwhile
```

This example uses the variable **guessed** as a 'flag' on Line 3. Variables used as **fl**
The flag variable will be set with an initial value, either True or False, depending
to do.

When the code on Line 13 evaluates to True then the '**flag**' on Line 15 (the var
with the result that the condition on Line 6 will now evaluate to False and the loo

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 A STATIC ARRAY)

array Identifier [size]

identifier [0] = Exp
identifier [1] = Exp

Note: Exp means any expression

A static array size is set at the start and cannot be changed as the algorithm or program runs.

```
1  array shopping[3]
2
3  shopping[0] = "milk"
4  shopping[1] = "bread"
5  shopping[2] = "butte
6
7  array a[4]
8
9  a[0] = 4
10 a[1] = 32
11 a[2] = 78
12 a[3] = 51
```

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

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

ASSIGNMENT (OF A DYNAMIC ARRAY)

array Identifier [ ]

A dynamic array will change size as the algo
is left blank when the array is created.

```
1  array shopping_list []
2
3  done = false
4
5  while NOT done
6      item = input("What shopping do you need?
7      shopping_list = shopping_list + item
8      check = input (" Have you finished? Enter
9      if check == "Y" then
10         done = true
11         print(shopping_list)
12     endif
13 endwhile
```

The empty array is created on Line 1.
*The WHILE loop will continue to ask for items to be added to the shopping list u
adds each item to the array in the next index position. When the letter Y is e
controlling the loop and Line 11 prints out the contents of the array.*

ACCESSING AN ELEMENT	
Identifier [IntExp]	1 shopping[1] 2 3 a[3] *Note: shopping [1] will evaluate t* *these are the index positions of eac*

UPDATING AN ELEMENT	
Identifier [IntExp] = Exp	1 shopping[2]= "egg 2 3 a[1] = 94 *Note: the element at index posi* *'butter' to 'eggs'. The array is now* *In the integer array the element* *from 32 to 94. The array is now [4,*

ACCESSING AN ELEMENT IN A 2D AR	
Identifier [IntExp, IntExp]	1 array high_scor 2 3 high_scores[0,0 4 high_scores[0,1 5 high_scores[0,2 6 high_scores[1,0 7 high_scores[1,1 8 high_scores[1,2 9 10 print(high_scor 11 print(high_scor *Line 1 creates the array* ***high_scor*** *like the two rows containing three* *In this example Line 10 would evalu* *of the 2D array (['Ashley', 'Rahee* *evaluate to the second item in the* *43]), i.e. 62.*

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 looki has been increased from 62 to 67. [['Ashley', 'Raheem', 'Jamie'],[58,67,43]]*

ARRAY LENGTH	
Identifier.length	`high_scores.length` will evaluate t the player names in row 0 and the player sc `shopping.length` will evaluate to 3 as

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 t
    print(count
next count
```

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

Example:

```
array daily_temps [7]
daily_temps = [17, 19, 22, 26, 21, 24, 22]
totalTemps = 0

for i = 0 to 6
    totalTemps = totalTemps + daily_temps[i
next i

avgTemp = totalTemps/7
print (avgTemp)
```

This example shows our array of daily temperatures, which is declared to contain assuming (unless the exam question states otherwise) that array counting starts

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 identifier.length option to find the length of the array, which = 7
FOR loop at index position 0 we need to subtract 1 from the length to ensure the

This algorithm could then be adapted to calculate the averages over a fortnight o
to set the array size by entering a suitable value which would also be used to div
find the average for each day.

```
// Calculate a weekly average temperature

array daily_temps [7]
daily_temps = [17, 19, 22, 26, 21, 24, 22]
totalTemps = 0

for i = 0 to daily_temps.length -1
    totalTemps = totalTemps + daily_temps[i
next i

avgTemp = totalTemps/7

print (avgTemp)
```

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

```
1  // Search for a name of students who sat mock exam
2
3  array examAttendees [10]
4  examAttendees = {"Keiran","Taisha","Emily","Wyatt","Ryan","
5                   "Grace","Adam"}
6
7  index = 0
8  check = ""
9  resits = []
10 found = False
11
12 while check != "X"
13     target = input("Enter name ")
14     index = 0
15     while index <= examAttendees.length -1 AND NOT found
16         if examAttendees[index] !=  target then
17             index = index + 1
18         else
19             found = True
20         endif
21     if index == examAttendees.length-1 AND NOT found then
22         resits = resits + target
23         endif
24     endwhile
25     check = input("Enter X to exit or C to continue")
26 endwhile
27 print(resits)
```

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

SUBPROGRAMS

Subprograms are clear, independent blocks of code within a computer program
by the main program. These are divided into procedures and functions. Either ca
depending on their purpose.

PROCEDURE DEFINITION	
PROCEDURE Identifier (parameters) <statements> ENDPROCEDURE	1 procedure multiply_n 2 total = a * b 3 print(total) 4 endprocedure 5 6 procedure greetings(7 n = input("Enter 8 print("Welcome"+ 9 endprocedure *The first procedure has two parameters. V* *actual values or 'arguments' will replace the* *The second procedure has no parameters a* *the user when the procedure is called.*

FUNCTION DEFINITION	
FUNCTION Identifier (parameters) <statements> ENDFUNCTION	1 function CheckPwd() 2 pwd = input("Ente 3 if pwd == "Turing" 4 return true 5 else 6 return false 7 endif 8 endfunction *A function will ALWAYS return a value.*

CALLING A SUBPROGRAM	

Identifier (parameters)	`multiply_nums (5,12)`
	
	`greetings()`
	
	`pwd_result = CheckPwd()`
	
	When we need to use or start the subprogr
	we write the name of the subprogram and s
	variable name to store the return value fro

PARAMETERS AND ARGUMENTS

The example shows the use of parameters. These are 'placeholders' inside the brackets after the name of the subprogram. In the multiply_nums procedure the placeholders are also used in the calculation.

```
function m
    total
    retur
endfuncti

procedure
    print
endproced

result = m
greetings
```

Not all subprograms will need a parameter value. The CheckPwd () subprogram shown on the previous page has no parameters so the brackets are empty.

The parameters used in the first subprogram are **a**, **b**; in the second subprogram 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 e.g. `vw_cars = ["Up!", "Polo", "Golf", "T-Roc`
Subprogram	Subprograms are clear, independent blocks of code within be called and accessed by the main program.
Call	The term used to describe 'starting' the subprogram.
Return	Subprograms that return values (to be used elsewhere in the Subprograms that do not return any values (e.g. printing ou procedures.

Complete Exercise 15: Guessing Game using Subprograms

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	

StringVar.length	`Example:`
	`myText = "the quick brown fox"`
	
	`myText.length` will evaluate to 19, wh

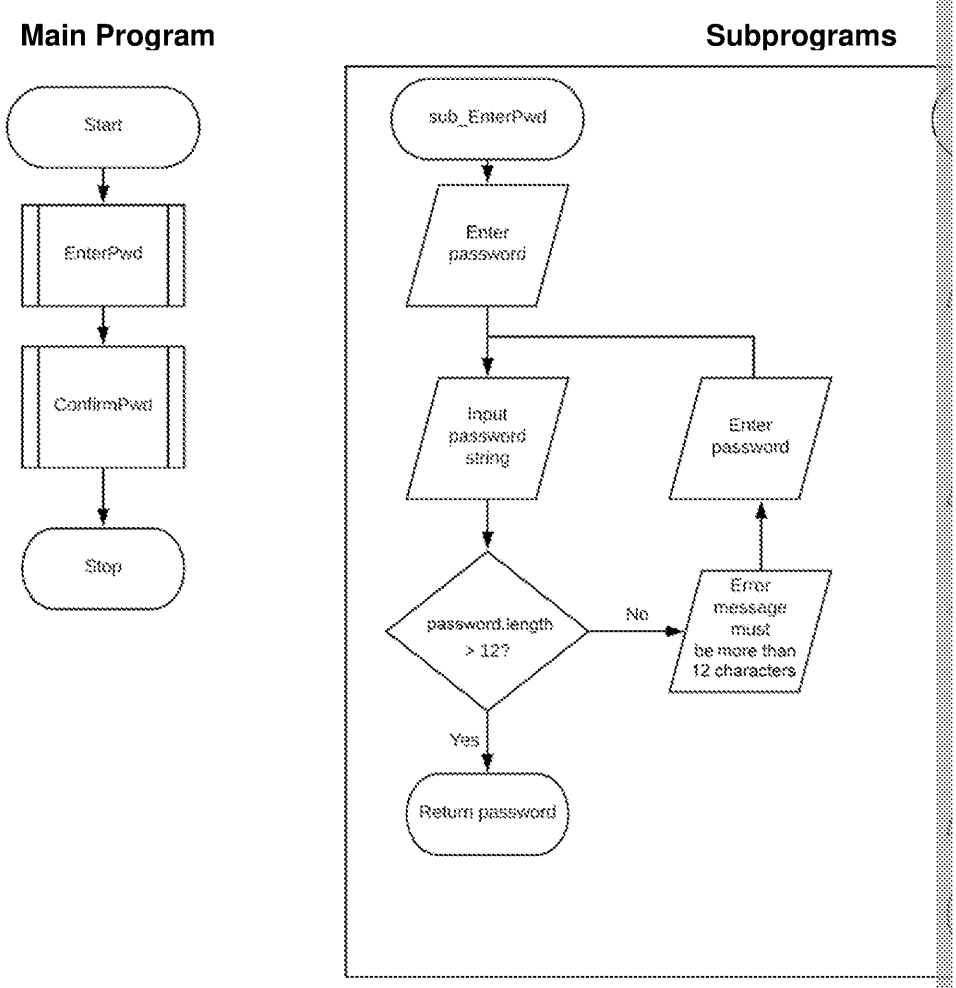
SUBSTRING	
StringVar.substring(IntExp, IntExp)	`myText.substring(4,10)` will evalu Note: the first parameter indicates the star the second parameter indicates the numbe
StringVar.left(IntExp)	`myText.left(9)` will evaluate to "the
StringVar.right(UntExp)	`myText.right(3)` will evaluate to "fox

CONCATENATION	
StringExp + StringExp	`"the quick brown fox" + " jum` will evaluate to "the quick brown fox jumpe

Complete Exercise 16: Strings and Substrings

Flow Charts and Subprograms

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



Explanation

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

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

The second subprogram, **sub_ConfirmPwd**, then asks for the password to be con
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	
INT (StringExp)	`int ("24")` `evaluates to the integer 24`
String to Real	
REAL (StringExp)	`real ("24.25")` `evaluates to the real 24.25`
Integer to String	
STR (IntExp)	`str (74)` `evaluates to the string "74"`
Comments	
Single line comments	`// anything written after th` `executed`

Real to String	
STR (RealExp)	`str (19.56)` `evaluates to the string "19.5`

Character to ASCII	
ASC (CharExp)	`ASC('G')` `evaluates to 71 using ASCII/U`
ASCII to Character	
CHR(IntExp)	`CHR (103)` `evaluates to 'g' using ASCII/`

Scope of Variables, Constants and Subprog

You may have heard of this term in your lessons on programming. The scope of a subprogram is about where that variable, constant or subprogram can be used. I correct type of scope or you may get unexpected results from your code.

There are two types:

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

Example:

```
1  // This variable is in GLOBAL scope
2
3   x = 15
4
5  function AddNums()
6
7      // This variable is in LOCAL scope
8      y = 19
9      global x
10     total = x + y
11     return total
12
13 endfunction
14
15 total = AddNums()
16
17 print(total)
```

In this example, the variable **x** is outside the procedure and therefore in GLOBAL
to the function **AddNums()**.

The variable **y** inside the function is in local scope and only available INSIDE the
practice to explicitly label global variables so that anyone reading the code at a l
code does. If the variable **y** had been created outside the function and assigned
running the function would still be 34 as the function will always look for local va

Using the same variable name in both the GLOBAL and LOCAL scope will lead to
try to use different meaningful identifiers/names for your variables as this will a
code clearer and easier to understand.

DEALING WITH ERRORS: VALIDATION TECHNIQUE

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

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

In this example, this algorithm expects the user to enter the data in integers, b
who may enter 'fifteen' instead of 15 to make sure our algorithm will not fail.

```
age = input ("Enter your age: ")
```

CATCHING ERRORS

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

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

do
    age = input ("Enter your age: ")
    try
        ageNumber = int(age) // the type check
        valid = True
        if ageNumber < 16 then
            print("You cannot drive anything yet")
        elseif ageNumber >= 16 AND ageNumber < 21
            print("You can drive a moped at 16 and
        else
            print("You can now drive any vehicle")
        endif
    catch
        print("Please enter age as a number in yea
until valid == True
```

Th
b

If it fai
an int

The DO… UNTIL loop continues until integers are entered as the age variable. This then changes the Boolean value of the flag variable **valid** to True and the 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

while NOT valid
    name = input ("Enter your name: ")
    if name.length == 0 then
        print ("You have not entered any text")
    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 input to the data type you want by wrapping the
the input command, e.g.

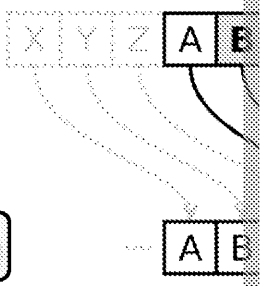
```
age = int (input ("Enter your age: "))
```

Complete Exercise 18: Password Checker Validation

USING CHARACTER TO ASCII AND ASCII TO CHA

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 ASC(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 CHR(76) will evaluate to the letter L.



Complete Exercise 19: Encryption Cipher

In order to solve some problems, it may be important to generate random num
roll of a dice in a board game.

RANDOM NUMBER GENERATION	
RANDOM(IntExp, IntExp)	`random(1, 6)` `# will generate 1, 2, 3, 4, 5`

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

READING FROM AND WRITING TO FILES

We have used arrays to store data and used iteration to loop through arrays that
algorithms. However, none of the data entered will be available to the algorithm
store data that can change each time the algorithm runs we need to write the da

OPEN A FILE	

Identifier = [open(Exp)]
 	
Note: Exp means any expression
	`1 myFile = open("myText.t`
	`2`
	`3 myFile.writeLine("The q`
	`4`
	`5 myFile.close()`
	
	This will open the specified text file and writ the file.

OPEN A FILE AND READ	

Identifier .readLine()
 	
Note: Exp means any expression
	`1 myFile = open("myText.`
	`2`
	`3 myLine = myFile.readLi`
	`4`
	`5 myFile.close()`
	
	This will open the specified text file and re myLine. The final line closes the file after re

CREATE A NEW FILE	
newFile(Exp) *Note: Exp means any expression*	1 newFile("myTextSample.txt 2 3 myFile = open("myTextSamp 4 5 myFile.writeLine("The qui 6 7 myFile.close() This will create the new file with the speci and write the text to the end of the file. The

ENDOFFILE()	
Identifier.endOfFile() *Note: Exp means any expression*	1 myFile = open("myTextSamp 2 3 while NOT myFile.endOfFil 4 print(myFile.readLine 5 endwhile 6 7 myFile.close() The statement endOfFile () is used in a wh each line of the file until the end is reached.

Example:

```
// Search for a name of students who sat mock exam

array examAttendees []
array mockResits []
index = 0
check = ""
found = False

myFile.openRead("examAttend.txt")
while NOT myFile.endOfFile()
    examAttendees = examAttendees + myFile.readLine()
endwhile

while check != "X"
    target = input("Enter name ")
    index = 0
    while index <= examAttendees.length-1 AND NOT fou
        if examAttendees[index] !=  target then
            index = index + 1
        elseif examAttendees[index] == target then
            found = True
        endif
    if index == examAttendees.length-1 AND NOT found
            mockResits = mockResits + target
    endif
    endwhile
    check = input("Enter X to exit or C to continue")
endwhile
print(mockResits)
```

This example was used back in the ARRAYS section. This time the data is algorithm. Each line in the text file is then read into the dynamic array e looping through the array can then occur. The rest of the algorithm wor results printed at the end of the process.

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 becomes
have already done this earlier in this booklet when we split problems up into inp
order to make them easier to solve.

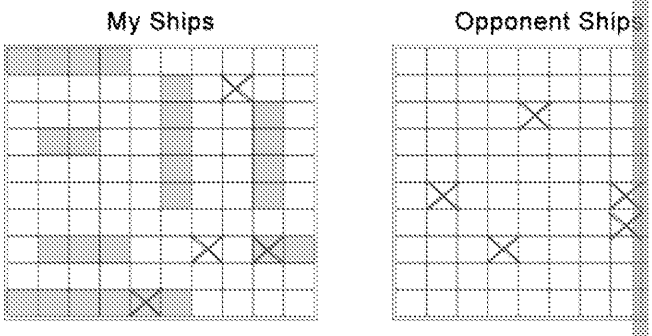
We can look at how we might plan our own battleships game starting with ident

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

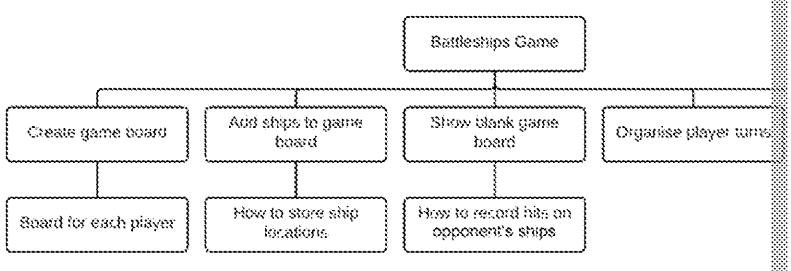
We can then look at each task, and consider: *Can it be solved in one go or does it*

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 or



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 i
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 n

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

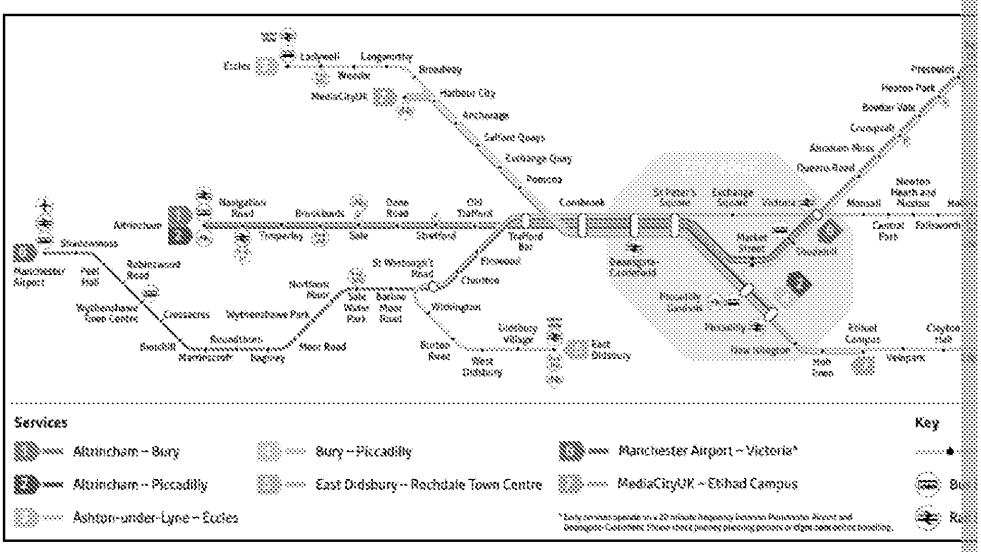
Step 1: Take the chicken across to bank B as the fox will not eat the grain but it w

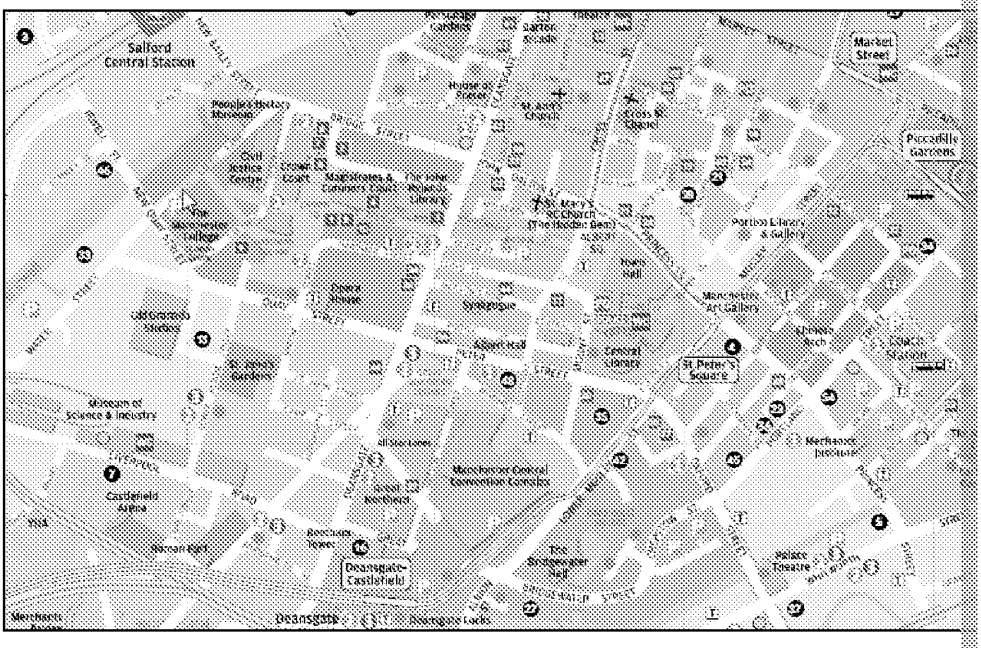
How many steps are left? What are they?

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 you
the city centre actually looks like, with some of the stations listed in the first map





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
there may be several different algorithms available which will all solve the same
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.

```
1  array nums [7]
2  nums = [4,7,12,13,17,19,23]
3
4  array odds []
5  array evens []
6
7  for i = 1 to nums.length - 1
8      if i MOD2 == 0 then
9          evens = evens + i
10     else
11         odds = odds + i
12     endif
13 next i
```

```
1  array nums [7]
2  nums = [4,7,12,13,17,19,23]
3
4  array odds []
5  odds = [next i for i = 1 to nums.length - 1
6  array evens []
7  evens = [next i for i = 1 to nums.length - 1
```

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

EFFICIENT CODE PROOF

If the Exam Reference Language is translated into Python 3 the code can be test
efficient. The examples shown below use a built-in function that enables the two

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 h searching for names in a set of data records to find a telephone number. We will 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:

```
1  procedure sea
2      found = f
3
4      for index
5          if li
6              f
7              p
8          else
9              i
10         endif
11     next inde
12     if found
13         print
14     endif
15 endprocedure
```

Step 1:

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

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.

```
for index = 0
    if list[i
        found
        print
    else
        index
    endif
```

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

```
4       for index
5           if li
6               f
7               p
8           else
9               i
10          endif
```

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

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

```
4       for index
5           if li
6               f
7               p
8           else
9               i
10          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 = 0
        if list[
            found
            print
        else
            index
        endif
    next index
    if found ==
        print("N
    endif
endprocedure
```

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 18, 19 and 20. This ensures that the search will only look at index positions from 0 to 9.

```
18      fo
19      fi
20      la
```

```
 1  array nameArray[10]
 2
 3  nameArray[0] = "Adam"
 4  nameArray[1] = "Bethany"
 5  nameArray[2] = "Darryl"
 6  nameArray[3] = "Emily"
 7  nameArray[4] = "Grace"
 8  nameArray[5] = "Keiran"
 9  nameArray[6] = "Ryan"
10  nameArray[7] = "Taisha"
11  nameArray[8] = "Wyatt"
12  nameArray[9] = "Zoe"
13
14  target = input("Enter search term")
15
16  procedure binary_Search(item, list)
17
18      found = false
19      first = 0
20      last = list.length -1
21      while NOT found AND first <= last
22          Midpoint = (first + last) DIV 2
23          if list[Midpoint] == item then
24              found = true
25              print("Name found at list index "+ st
26          else
27              if item < list[Midpoint] then
28                  last = Midpoint-1
29              else
30                  first = Midpoint + 1
31              endif
32          endif
33      endwhile
34      if found == false then
35          print( "Item not found")
36  endprocedure
37
38  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 22 adds 0 + 9 and then uses integer division to find the midpoint index posit

```
21      while NOT found
22          Midpoint = (
```

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

Midpoint

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

```
23          if list[Midpoint] == item then
24              found = true
25              print("Name found at list index
```

Step 4:
If the search item is not found here, then a check is made to see if the item we a
this starting midpoint value in the ordered list.

The name we are looking for, 'Zoe', is above so Line 30 is executed. The value of

```
if list[Midpoint] == item then
    found = true
    print("Name found at list index "+
else
    if item < list[Midpoint] then
        last = Midpoint-1
    else
        first = Midpoint + 1
    endif
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 s 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 n (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, 9
 4          104, 109, 120, 123, 125, 127, 128, 130, 131, 132, 138, 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 com
```

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 three sorting methods that you need to understand for your exam: bu
insertion 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 a
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   array nameArray[6]
2
3   nameArray[0] = "Keiran"
4   nameArray[1] = "Taisha"
5   nameArray[2] = "Emily"
6   nameArray[3] = "Wyatt"
7   nameArray[4] = "Ryan"
8   nameArray[5] = "Zoe"
9
10  swapped = true
11
12  while swapped == true
13      count = 0
14      for x = 0 to nameArray.length-1
15          if nameArray[x] > nameArray[x + 1]
16              temp = nameArray[x]
17              nameArray[x] = nameArray [x + 1
18              nameArray[x + 1]  = temp
19              count = count + 1
20          endif
21      next x
22      if count == 0 then
23          swapped = false
24      endif
25  endwhile
```

Step 1:
A **flag** variable called 'swapped' is set on Line 10. This is the condition controlling the WHILE loop.

```
10  swap
11
12  whil
13
```

The variable 'count' is used to check whether the array is sorted. The array will be sorted when Line 15:

`if nameArray [x] > nameArray [x + 1]` evaluates to False.

The algorithm will then move to Line 22, the flag variable 'swapped' is chang
finishes.

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 examp
the alphabet so no swap is needed and the code moves to Line 14 and the value
by 1.

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

```
12  while swapped == true
13      count = 0
14      for x = 0 to nameArray.length-1
15          if nameArray[x] > nameArray[x + 1]
16              temp = nameArray[x]
17              nameArray[x] = nameArray [x + 1
18              nameArray[x + 1]  = temp
19              count = count + 1
20          endif
21      next x
22      if count == 0 then
23          swapped = false
24      endif
25  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 16 stores the value at index p
temp. Line 17 puts the data at index position 2 into index position 1. Line 18 cop
temp into index position 2. The 'count' variable is also incremented as a swap ha

The array now looks like this:

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

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

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

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

Result of first **pass** of the bub

As you can see, the last two items are now in order in the correct position. The V
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
Emily	Keiran	Tais

0	1	2
Emily	Keiran	Tais

0	1	2
Emily	Keiran	Rya

0	1	2
Emily	Keiran	Rya

0	1	2
Emily	Keiran	Rya

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

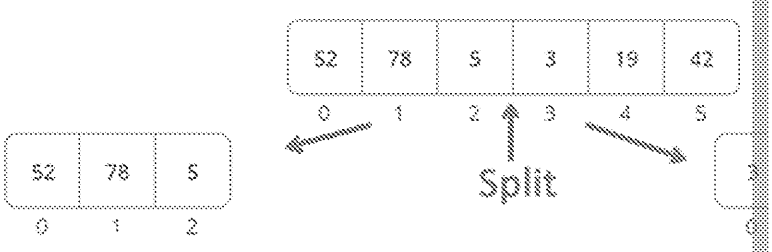
Complete Exercise 25: Bubble Sort Exercises
Complete Exercise 26: Put the Bubble Sort Flow Chart in Order

MERGE SORT: HOW IT WORKS

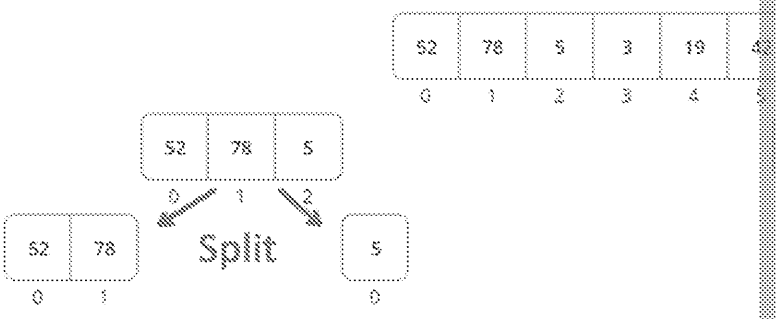
The merge sort is much more complex than the bubble sort and is known as a 'd
splits up the data array to be sorted into smaller sub-arrays until the sub-array h
arrays are then sorted and recombined into a sorted array.

Example:

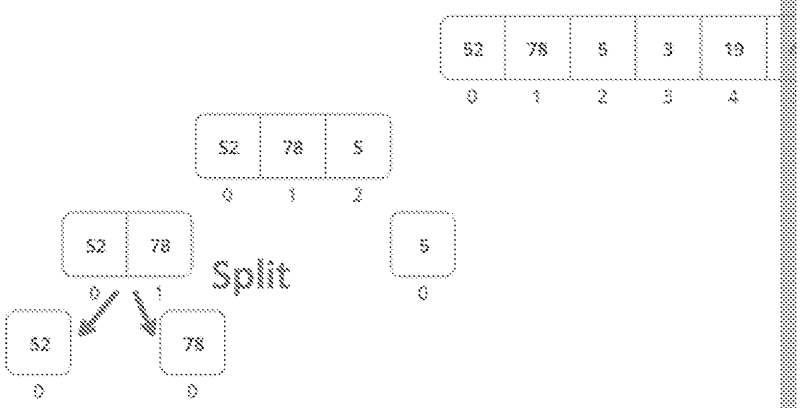
Step 1: Split the array in half at the midpoint.



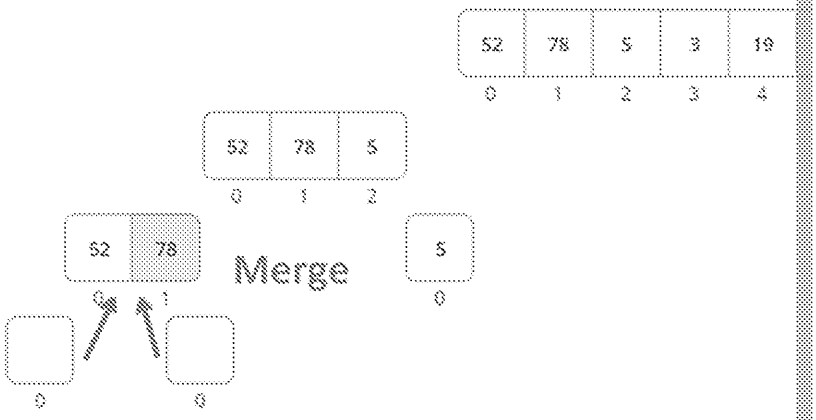
Step 2: Select the left sub-array and split again.



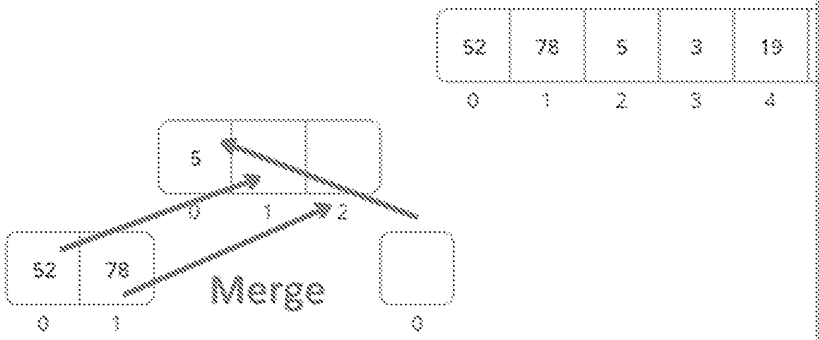
Step 3: Select the left sub-array and split again so that the sub-array has just one



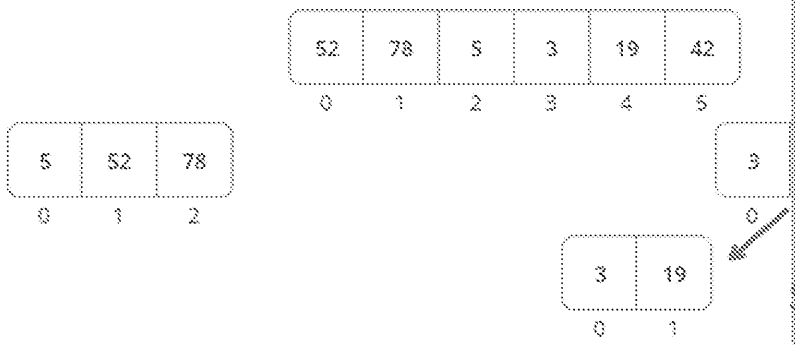
Step 4: Merge the sorted data back into an array.



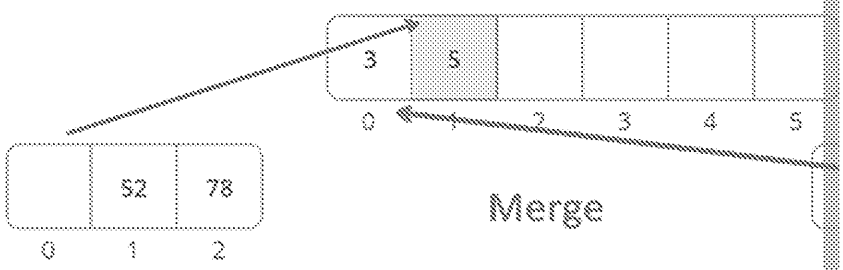
Step 5: Combine the sorted array and merge together with the smallest item firs



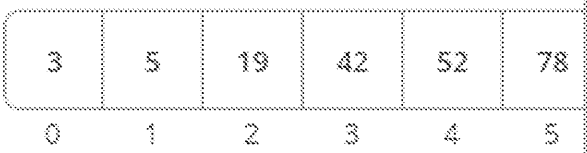
Step 6: Repeat the process with the right sub-array.



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

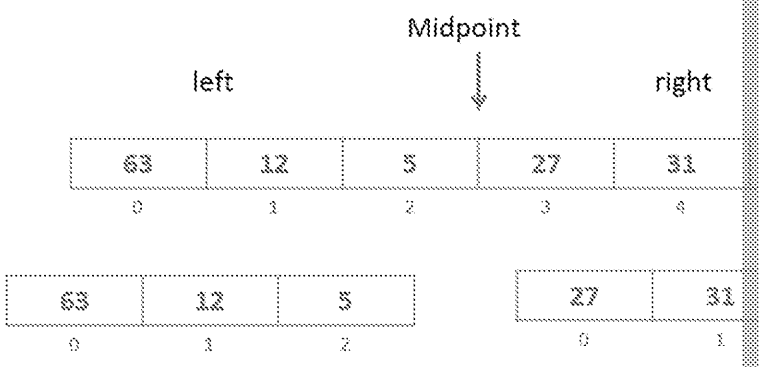
We will use a simple array of numbers: [63, 12, 5, 27, 31, 45]

```
1  function mergeSort(dataArray)
2
3      if  dataArray.length > 1 then
4          mid = dataArray.length DIV 2
5          leftHalf = dataArray[:mid]
6          rightHalf = dataArray[mid:]
7          mergeSort(leftHalf)
8          mergeSort(rightHalf)
```

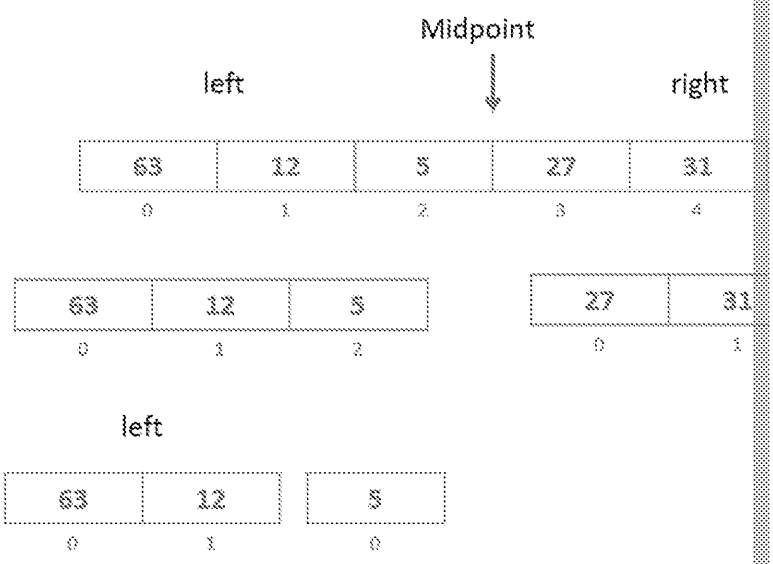
Step 1: Line 3 checks whether the array is larger than one; if it is not, the data is s
code in the IF statement will not be executed. Line 4 finds the midpoint of the a
the **dataArray** into a left half and a right half.

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 subprogram. In this case we are making the problem smaller by calling the
passing half the variable **dataArray** into the subprogram as the **parameter**.

Here is the current state of our array, dataArray:

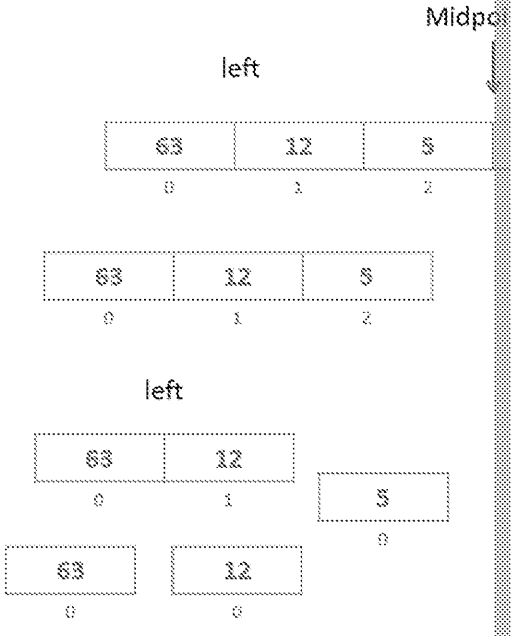


Step 3: The process is now repeated from Line 2 as the **mergeSort** subprogram is
using the **left** part of the original **dataArray**. The current state of the array now l
split in half again.



Step 4: The process is repeated again as the **mergeSort** subprogram is called again in Line 6 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 subprogram now sorts the data back into order, star
subprogram uses three index variables, i, j and k, on Lines 9 to 11 to set the start
sorting the data back into the original **dataArray**.

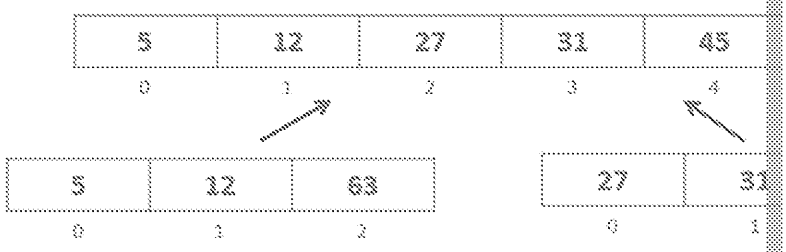
The comparison will start with 63 and 12. 63 is greater than 12 (see Line 17) so it
back into the **dataArray**. The values of variables i, j and k are also incremented (L

```
9           i = 0
10          j = 0
11          k = 0
12          while i < leftHalf.length AND j < rightH
13              if leftHalf[i] < rightHalf[j] th
14                  dataArray[k] = leftHalf[i]
15                  i = i+1
16              else
17                  dataArray[k] = rightHalf[j]
18                  j = j+1
19              k = k+1
20              endif
21          endwhile
22
23          while i < leftHalf.length
24              dataArray[k]=leftHalf[i]
25              i = i+1
26              k = k+1
27          endwhile
28
29          while j < rightHalf.length
30              dataArray[k]= rightHalf[i]
31              j = j+1
32              k = k+1
33          endwhile
34      endif
35      return dataArray
36
37  endfunction
38
39  array dataArray[6]
40
41  dataArray = [63, 12, 5, 27, 31, 45]
42
43  results = mergeSort(dataArray)
44  print(results)
```

Step 6: When the data on the left is sorted, the process of splitting and sorting w
the right.

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

Step 7: The left and right arrays are now finally sorted into order by running the
to the end by comparing each item in the two arrays before copying them into th



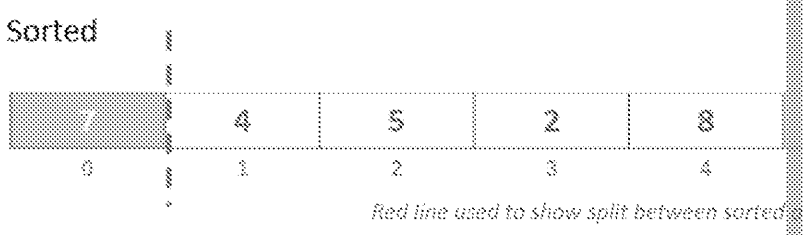
INSERTION SORT: HOW IT WORKS

The insertion sort is another simple sorting algorithm which uses two types of ite
set. It works by assuming the first item in the array is already sorted (index posit
to store the first element in the unsorted array (index position 1).

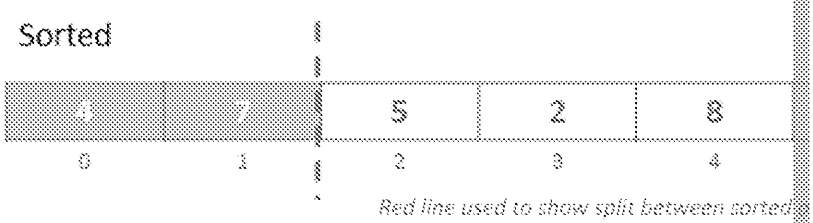
The first item in the unsorted array is then compared with the sorted array an
order. The next element in the unsorted array is then stored in the key, and the
is sorted.

Example:

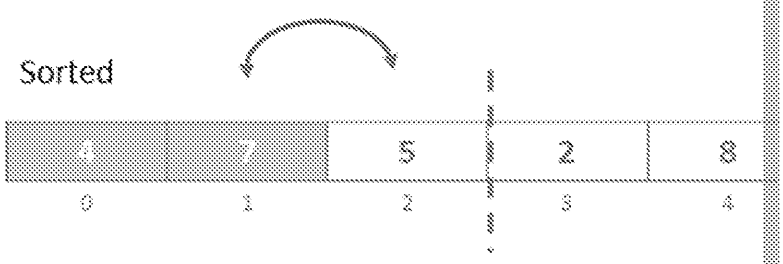
Step 1:
The first item in the array is assumed to be sorted and the first item in the unsor
variable.



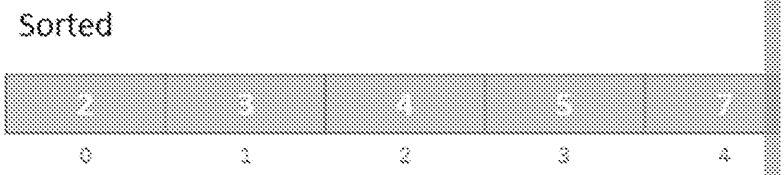
Step 2:
The item in the 'key' variable is now compared with the sorted array and mov
example the value of 'key' will now be 5.



Step 3:
The process is repeated for each element in the unsorted array; each time the n
data elements on the left and swapped until it is sorted.



Step 4:
This continues until the end of the array is reached and the data is sorted.

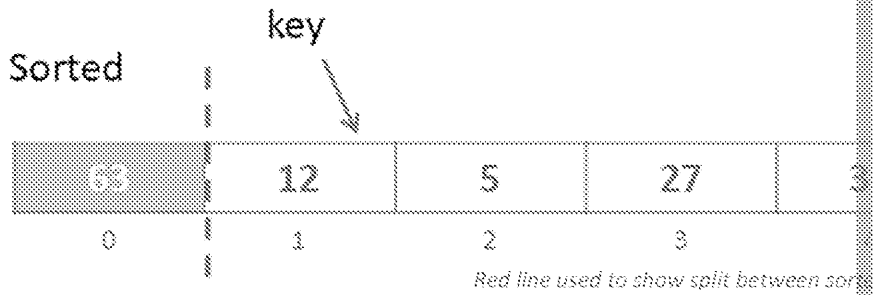


PSEUDOCODE FOR THE INSERTION SORT

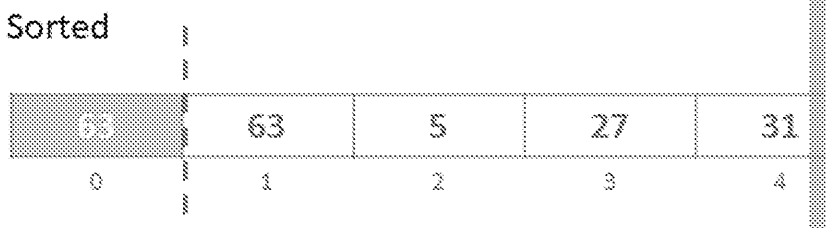
We will use a simple array of numbers: [63, 12, 5, 27, 31, 45]

```
1  array myArray[6]
2  myArray = [63, 12, 5, 27, 31, 45]
3
4  function insertionSort(arr)
5
6      for index = 1 to arr.length - 1
7          key = arr[index]
8          currentPos = index - 1
```

Step 1:
Line 6 sets the length of the unsorted array from index position 1 to the end of th
index position 1 in the unsorted array into the variable **key** so that it can be mov
array. The variable **currentPos** is used to control the position of the variable sto



Step 2:
The WHILE loop runs between Lines 9 and 13. At this point **currentPos** = 0 and **k**
is less than arr [0], which contains the integer value 63. Line 10 now puts whatev
1 so the array now looks like this temporarily:

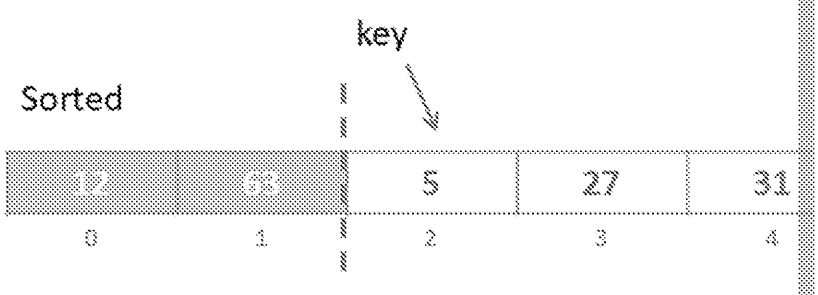


On Line 11 the value of **currentPos** is changed. Now **currentPos = currentPos -1**,
running the WHILE loop false.

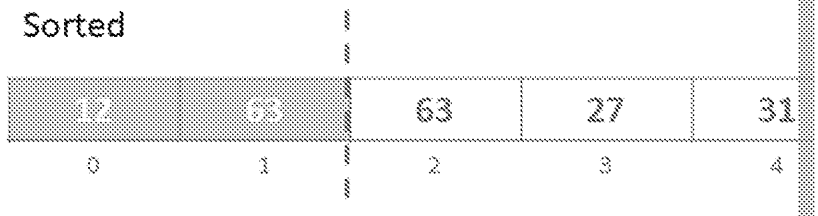
Line 14 now puts the value of **key** into the first index position in the array (-1 + 1

```
9           while currentPos >= 0 AND key <
10              arr[currentPos + 1] = arr[
11              currentPos = currentPos -
12
13          endwhile
14          arr[currentPos + 1] = key
15      next index
```

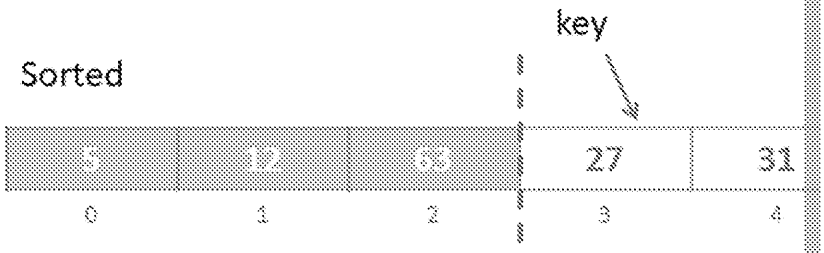
The array now looks like this:



Step 3:
Line 15 increments the value of **index = 2**. The value of **key** is now 5 and **current**
entered as both conditions are true and the array now looks like this:



Line 11 decrements the value of **currentPos = currentPos – 1** and Line 14 puts th
position 0 and the while loop finishes again. The array now looks like this:

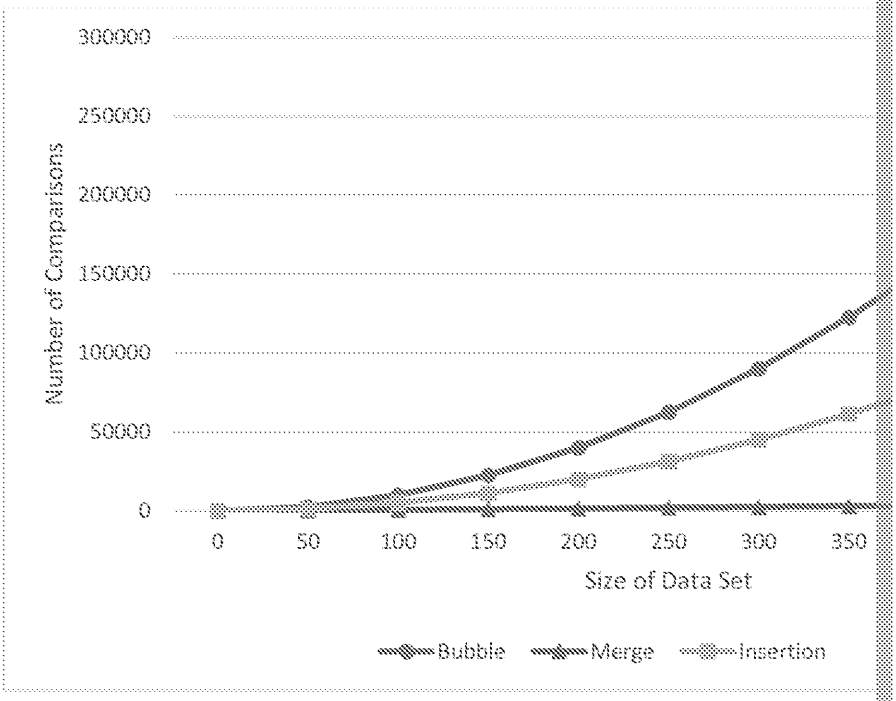


The **FOR** loop continues until the end of the array has been reached and the arra

```
array myArray[6]
myArray = [63, 12, 5, 27, 31, 45]

function insertionSort(arr)

    for index = 1 to arr.length - 1
        key = arr[index]
        currentPos = index - 1
        while currentPos >= 0 AND key <
            arr[currentPos + 1] = arr[
            currentPos = currentPos -


        endwhile
        arr[currentPos + 1] = key
    next index
    return arr
endfunction

sorted = insertionSort(myArray)
print(sorted)
```

CHART: COMPARE BUBBLE, MERGE AND INSERTI

This chart compares the bubble sort, the merge sort and the insertion sort for th
the merge sort is very efficient, regardless of the size of the data being sorted
quickly becomes very inefficient as the size of the data to be sorted grows. Th
bubble sort, particularly with small data sets, but still much less efficient than th



BUBBLE SORT VS MERGE SORT VS INSERTION SORT

COMPARISON CRITERIA	BUBBLE SORT	MERGE SORT
Advantages	(1) Very simple algorithm, easy to code. (2) Uses much less memory than a merge sort.	Much faster than the bubble sort, regardless of the amount of data.
Disadvantages	Slowest algorithm of the three.	(1) The merge sort is much more complex to code. (2) Uses more memory as copies of the arrays are made when the original array is split up.

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.

Insertion sort This sort assumes the first element in the array is already so
sorted elements from the unsorted elements. Each element in
compared one at a time with the data in the sorted array a

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
subprogram inside itself as part of the subprogram.

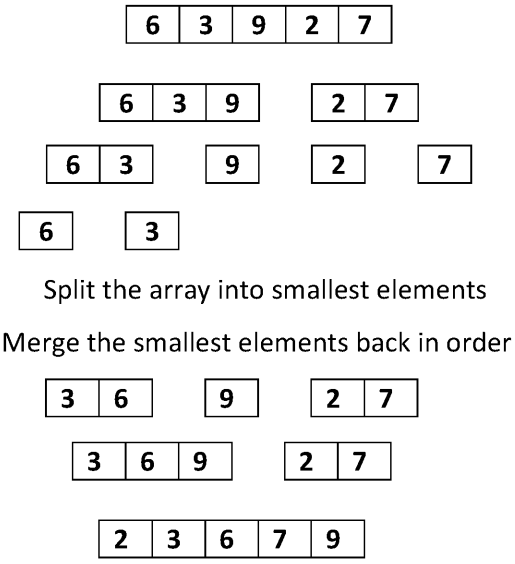
EXAMPLE EXAM QUESTION & SOLUTION:

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



Split the array into smallest elements

Merge the smallest elements back in order

Complete Exercise 27: Sorting and Searching
Complete Crossword Five