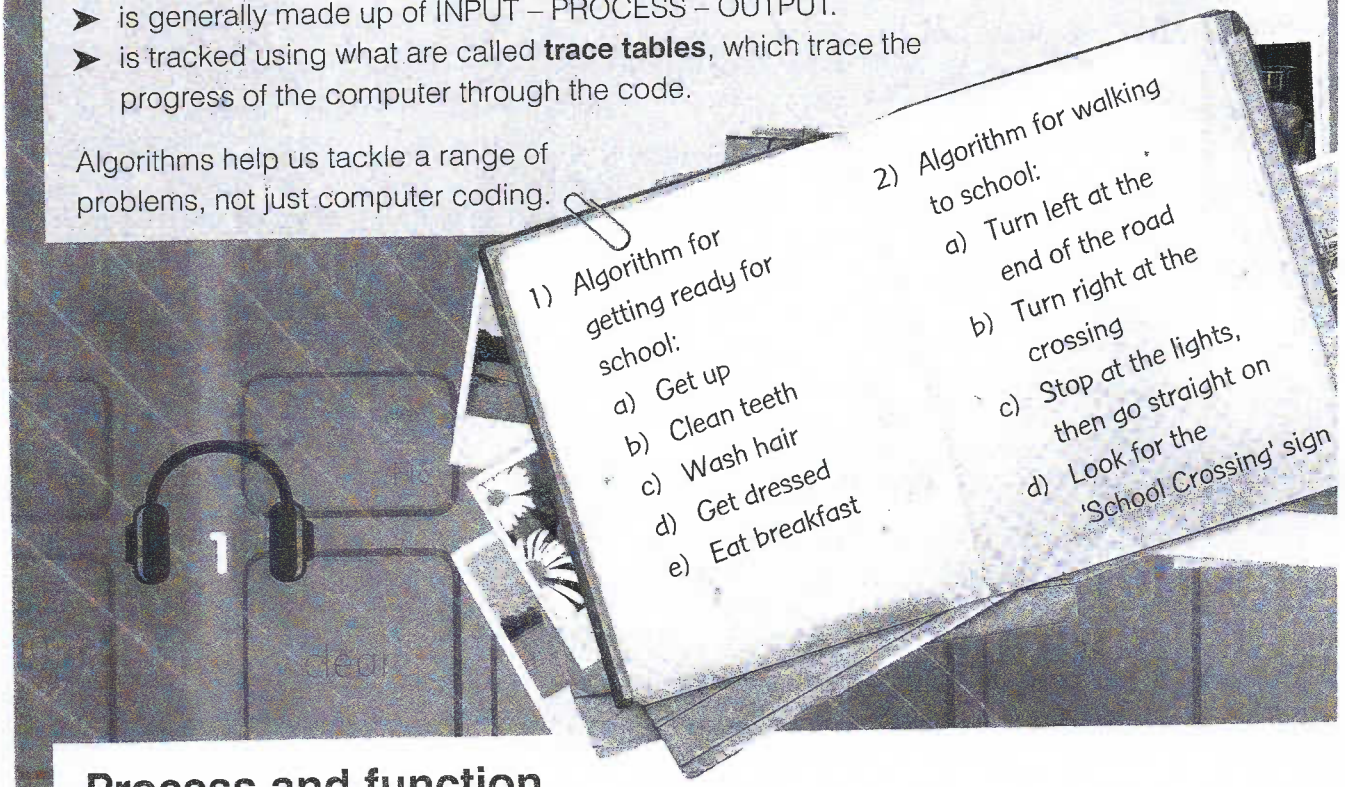
Algorithms

An algorithm is a process used to solve a problem or achieve a task. In computer science, sometimes a program is used. Be careful not to confuse the two! A program is a type of algorithm, but then so is a recipe.

An algorithm:

- is a set of steps, or instructions, to complete a task. We use them all the time – we just don't realise we are.
- is a way to solve problems, or complete tasks. There are a lot of different ways in which to complete each task, just like there are a lot of different Victoria sponge cake recipes or there is more than one way to go to school.
- is judged by whether it completes the task, or solves the problem, in the **most efficient way**. An efficient algorithm only uses the resources it absolutely needs in order to complete the task – no more, no less
- is generally made up of INPUT – PROCESS – OUTPUT.
- is tracked using what are called **trace tables**, which trace the progress of the computer through the code.

Algorithms help us tackle a range of problems, not just computer coding.



Process and function

Algorithms are generally made up of INPUT – PROCESS – OUTPUT. Keeping track of what the code is doing at any one time can be challenging, so we use trace tables, which trace the progress of the computer through the code.

When you use a trace table, you go through your code as if you are the computer, writing down the inputs, the processes and the outputs. This is an essential part of testing and can help you see where the computer code would go wrong before you even enter it, which saves time in class – and money in business.

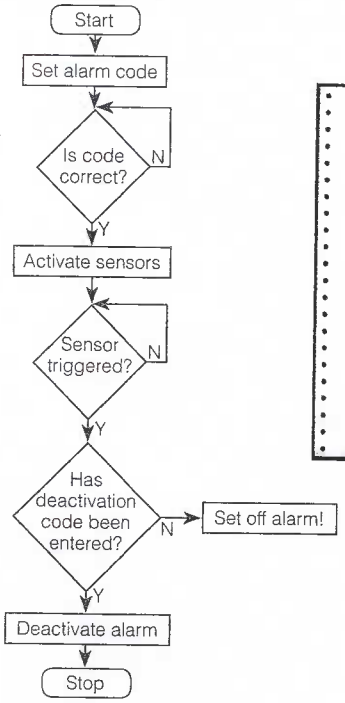
KEYWORDS

Efficiency ➤ The extent to which a task is completed in the shortest possible amount of code. This also relates to the amount of memory used and speed of execution of the code.
Trace tables ➤ A template to help you follow each instruction in your program

Flowcharts

Flowcharts are a great visual way to plan a program through a sequence of events.

Simple Burglar Alarm Set:




```
Consider the following
short counting algorithm:
x = 1
OUTPUT x
WHILE x <= 3
        y = x * 5
        PRINT y
OUTPUT "Complete"
END
```

Trace tables

A trace table allows you to work through an algorithm or program to iron out any potential problems.

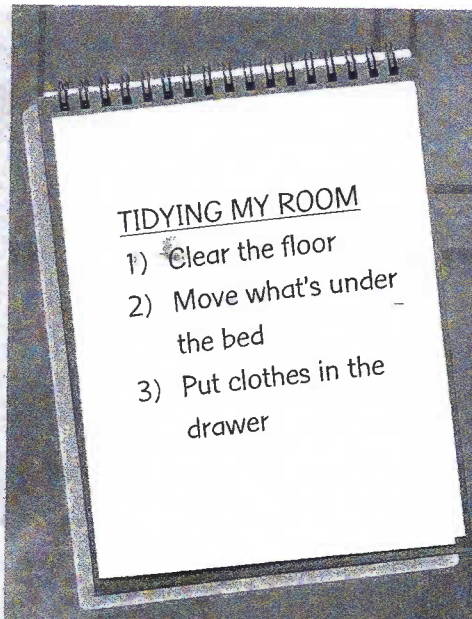
Line	x	y	Output
1	1		
2			1
3			
4		5	
5			5
3	2		
4		10	
5			10
3	3		
4		15	
5			15
6			Complete

Decomposition

When you decompose a task, you break it into smaller tasks.

Decomposition:

- helps you write your code because it is usually easier to write code for each section at a time, rather than trying to write the whole thing in one go. So, an alarm system for your home will have a 'set/unset' program and a 'live' program.
- makes life a lot easier when you are testing your code because you can test each section before you run it all together. Think about the code needed to run a driverless car – you can see why it is a good idea to test the bits of code (like stopping when something is in the way) before you put the car on the road.



Abstraction

Abstraction is the way we remove unneeded information from a task so a computer can complete it. In the exam, you will need to be able to read code, or a brief, and identify the 'real problem'. For example, if you are programming a driverless car, does it matter if the car is pink or yellow? What is more important is whether speed or direction is set accurately.

Make a model of a computer showing INPUT – PROCESS – OUTPUT, or be more ambitious and include storage! Show how data travels through the system.

1. What is an algorithm? Explain using **two** different examples: one in human terms, and one referring to computers.
2. What do we mean by an 'efficient algorithm'?
3. What are the main components of algorithms?
4. How does a trace table help with efficient code design?

Searching

Searching is a basic and common task for an algorithm. For example, your phone searches for a signal and your computer looks for an internet link. There are **two** main types of search algorithm that you need to know: **binary** and **linear**.

Binary search

A binary search chooses the middle value in a sorted **array**, and works out whether the value sought is higher or lower than the middle value. Then it splits the array again to check again – is the value higher or lower than the middle value? The list is repeatedly split in half and half and half until the item is located.

This means that the computer only ever searches half of the data, because it is either on the correct number, or pointed at the correct half of the data that is in the array.

We use this method in our own lives, too. If you're looking for a student in a class whose surname begins with H, you would not start at A.

In pseudocode, the binary search algorithm is short. Follow the code through step by step:

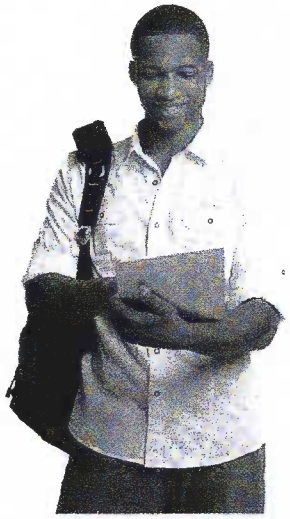
```
ClassList= [5,6,8,9,15]
INPUT A #number you're looking for
IF (first > last) return –1; # number not found represented by –1
ELSE
{
    INT mid = (first + last)/2;
    IF (value = = A[mid])
        RETURN mid;
    ELSE IF (value < A[mid])
        BinarySearch(A, first, mid-1, value);
    ELSE
        BinarySearch(A, mid+1, last, value);
}
END
```

The content of the array is inside the [] brackets.

identifies a comment ignored by the computer.

== means "is equal to"

The program is contained within the {} brackets.



Another example of this is shown below:

A group of students have received their test results. To find out which student got 9, the computer would first sort these students and their grades in the left column into grade order in the right column:

Alex: 5	Alex: 5
Bailey: 9	Riley: 6
Chris: 8	Chris: 8
Jamie: 15	Bailey: 9
Riley: 6	Jamie: 15

In a binary search, the computer would start in the middle, with Chris, at 8 marks. That's too low, so it would split the top half in two, which would leave the middle entry at Bailey, who gained 9 marks.

KEYWORDS

Array ➤ A set of data items of the same type grouped together using one identifying label or identifier

Linear search

A linear search:

- is used when you can't sort the data into any useful order, or when there are very few items to search.
- is also known as 'brute force' searching. The computer searches from the first item until either the end of the data or until the item sought is located, one by one.
- is best used when the data to be searched is very small, or is so frequently amended that sorting is a waste of time.
- can take a really, really long time because it checks every single item, which is why it is used only when a binary search isn't possible or appropriate.

The pseudocode for this type of search is much more straightforward. Follow it through in the graphic below.

```
ClassList= ["Alex", "Riley", "Chris", "Bailey", "Jamie"]
INPUT A #name you're looking for
FOREACH [item in list]:
    IF [item] = = A,
    OUTPUT result stop search
ELSE RETURN
END
```

The content of the array is inside the [ ] brackets.

The program is within the FOREACH command, which specifies that the computer must examine every single entry in the class list array.

Binary vs linear

When selecting the most suitable search algorithm, consider these summary points:

- Data must be already sorted before a binary search can be carried out.
- A linear search can be slower as it checks every piece of data.
- With very small data sets neither method would have a particular speed advantage.
- A linear search can be completed with a shorter pseudocode sequence.

In pairs: use a suit of playing cards – so just the spades or hearts for example – and while one of you thinks of a card in the suit, the other one uses binary search to locate it. Compare that with wild guessing to see the effectiveness of a systematic search.

1. Imagine you are looking for a telephone number for K. Lombard in London, which of the two search techniques would be the best? Explain your answer.
2. Show how you would use binary search to find the person with 17 marks from this group: 22, 19, 17, 15, 2, 9.
3. Which search technique requires data to be sorted?

Sorting

An important part of managing data, sorting algorithms allows us to find things quickly by putting data into some sort of order. This might be putting names into alphabetical order or finding the cheapest online console game.

Merge sort

Merge sort is also known as 'divide and conquer' because, like the binary search, the merge sort splits the data in half and works on each half in turn. The computer then sorts the smaller groups before merging the two together again. This is rather like when we shuffle cards, only we mess up the sort while the computer code resets it.

To complete a merge sort, the computer follows two steps.

1. It repeatedly splits the data into two halves until each 'list' contains only a single data item.
2. Having broken it into smaller parts, it repeatedly merges these 'lists' back together, this time putting them in their required order (ascending or **descending** in value).

```
MergeSort (Array(First..Last))
BEGIN
IF Array contains only one element THEN
    RETURN Array #already sorted
ELSE
    Middle= ((Last + First)/2) #rounded down to the nearest integer
    LeftSide = MergeSort(Array(First..Middle))
    RightSide = MergeSort(Array(Middle+1..Last))
Result = Merge(LeftSide, RightSide)
RETURN Result
ENDIF
END MergeSort
```

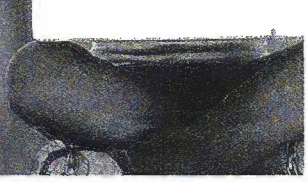
Bubble sort

Bubble sort comes from the way that stones settle in a tank of bubbling water: the larger stones sink to the bottom of the tank while the small ones appear to rise to the top. That is why a bubble sort is called a sinking sort. This is very rarely used, but it illustrates important principles so is included in all computer science study. Its slow methodical nature means it is normally only used for very small volumes of data.

- A bubble sort compares the first two items, checks which one is larger, and swaps them if necessary so that the larger is first.
- Then it checks the next pair, and so on.
- If there have been any position changes, the whole process is begun again until the computer can go from start to finish and there are no changes to be made. At this point the data is in descending order.

KEYWORDS

Descending ➤ Decreasing in number to 1 downwards, or in value from A



A practical example of this is worked through below.

I have a list of grades [5,6,8,9,15] and need to sort them from highest to lowest.
The computer looks at the first two: [5,6]
The second (right hand) is larger, so they are swapped [6,5]. Then the next two are checked [5,8].
Again, these need to be swapped [8,5].
The next two are [5,9] which, again, have to be changed (see how the 5 is being pushed down to the bottom) [9,5]
Finally [5,15] have to be swapped to [15,5].

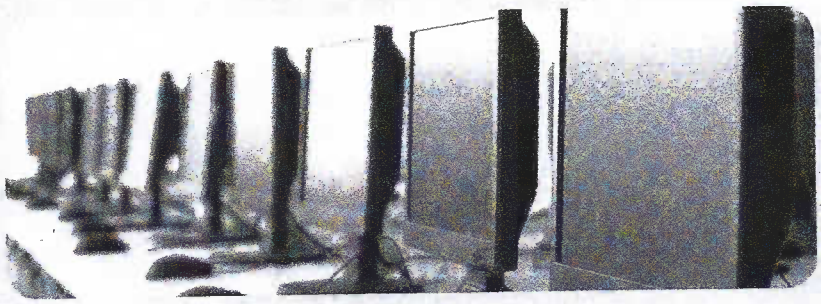
In one pass, the list has been changed from [5,6,8,9,15] to [6,8,9,15,5]. The computer will have to do this sequence again, from the start, to change the list and slowly, pass by pass, move this to [15,9,8,6,5]. The numbers would change like this, in ELEVEN steps:

[5,6,8,9,15]
[6,5,8,9,15]
[6,8,5,9,15]
[6,8,9,5,15]
[6,8,9,15,5]
[8,6,9,15,5]
[8,9,6,15,5]
[8,9,15,6,5]
[9,8,15,6,5]
[9,15,8,6,5]
[15,9,8,6,5]

Values that are already in the 'right place' don't move again.

The code for this is a lot shorter:

```
FOR i from 1 to N
    FOR j from 0 to N - 1
        IF a[j] > a[j + 1]
            SWAP (a[j], a[j + 1])
```



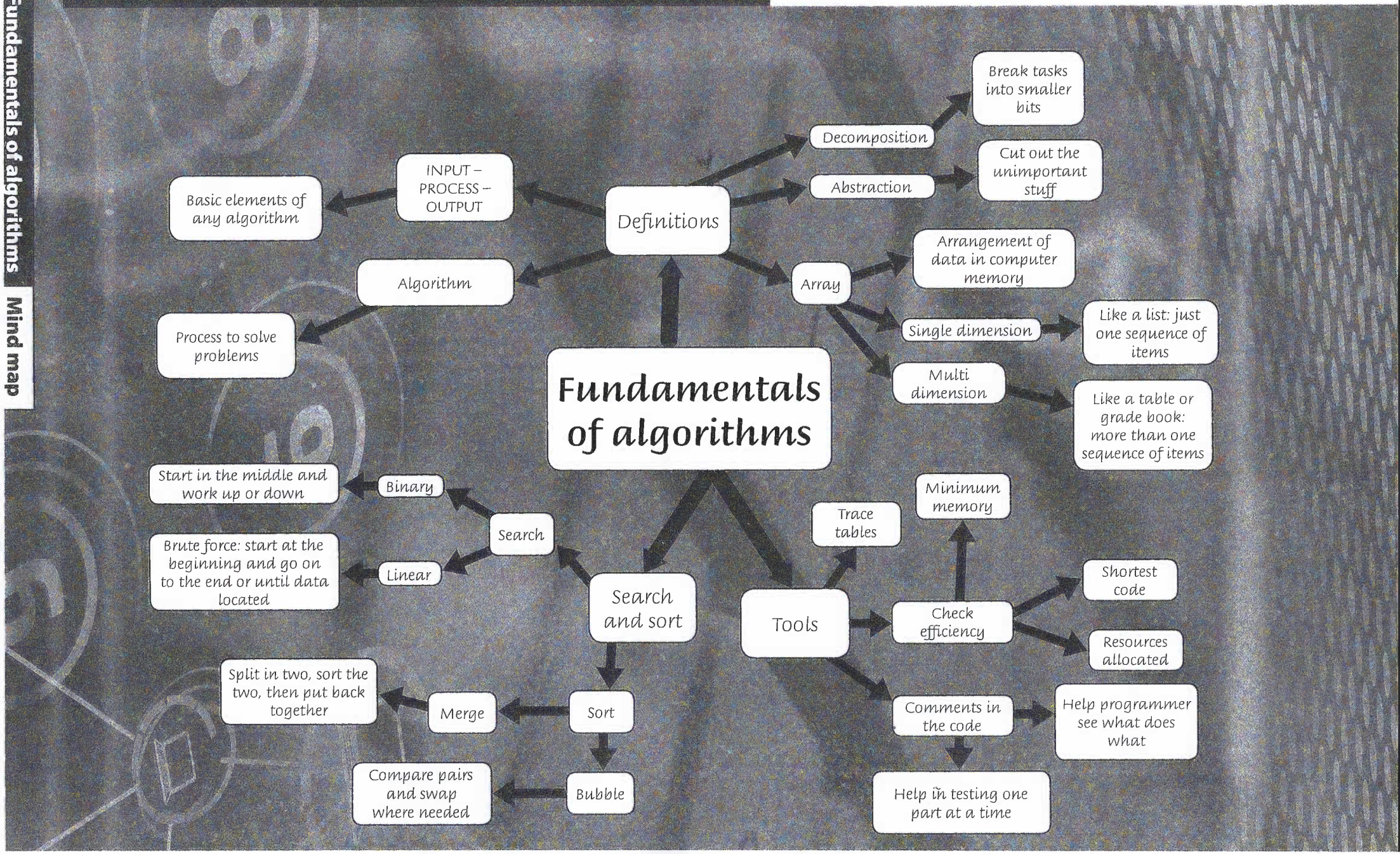
For the exam, you need to be able to compare and contrast these two types of sorting algorithms.

	Bubble	Merge
Efficiently handle big data sets?	No	Yes
Speed	Usually slower	Faster
Space used in memory	Stable (never changes since items are just swapped)	Variable (splits the data in two)

In a box, mix up small and large stones or marbles. If you shake them (gently) you will see bubble sort in process, as the larger stones or marbles come to the top.

1. You need to work out the oldest person in a class of 20. Which would be the most appropriate of the two sort techniques? Explain your answer.
2. Show how you would use bubble sort to create a descending list from this group: 22, 19, 17, 15, 2, 9.
3. Which sort technique is appropriate for large amounts of data?

Fundamentals of algorithms Mind map



Answer the following questions.

1. Complete the trace table for the algorithm below.

```
number = 5
OUTPUT number
FOR i from 1 to 3
    number = number +3
    OUTPUT number
END
```

Line	Number	i	Output
1	5		
2			5
3		1	
4			
5			
6			
3			
4			
5			
3			
4			
5			

(8 marks)

2. Decompose the task: an automatic car wash program that will wash all vehicles from a small car to a minibus. **(5 marks)**

3. Decompose the task: make your bedroom neat enough for an inspection. **(2 marks)**

4. Explain how decomposition helps programmers code efficiently. **(3 marks)**

5. Explain how abstraction helps programmers to plan a coding solution to a problem. **(3 marks)**

6. Describe what this algorithm does.

```
counter = 1
WHILE counter < 11
    PRINT counter
    counter = counter + 1
PRINT "All done"
END
```

(2 marks)

7. Explain how the efficiency of algorithms is judged. **(3 marks)**

8. Compare and contrast linear and binary search algorithms. **(5 marks)**

9. Compare and contrast merge sort and bubble sort algorithms. **(3 marks)**

Main data types

Data types allow us to categorise the range of data that is used in programs. Computer programs treat different types of data in different ways.

Here are the main data types:

Integer is a whole number, no decimal point or fraction, no letters. It might be used to define a count for a loop – you can only perform an instruction a whole number of times, not 2½ times, for example. (It's a term you might have met in maths.)

Boolean data is digital: yes/no; on/off; true/false. This is essential for loops – for example, is the temperature over a set point? Has the loop been run x number of times? (You should have met this as part of your work learning how to search the Internet effectively.)

Real data tries to reflect 'real' numbers, so this includes decimals: these are sometimes described as 'float' numbers because the number of decimal places can vary depending on the level of detail needed. It might be used to show the weight of an item in kilos, or fractions of distance measurement. Being specific about 4.75 kilos, not 5, for example, could make a difference for dosage of medication or fertiliser.

Character is any one of the letters and symbols on a keyboard. You can also define a number as a character when you don't want to use it for a mathematical operation.

String can hold any number of alphanumeric characters: text, number, symbol. This can include name, address and telephone number entries, for example. Although a telephone number is constructed of numbers, we don't use it for mathematical operations: it is effectively an address, so it is stored as a string.

Data Type	Example Use
Integer	loopCount = int(a/b)
Boolean	heatingOn = true
Real	weightInKilos = 2.742
Character	Sex = "F"
String	PhoneNumber = "012115489721"

Programming concepts

There are three main constructs within coding: **sequence**, **iteration** (or **repetition**) and **selection** (or **choice**).

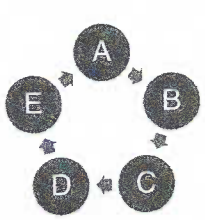
Constructs

Sequence is the simplest form, or structure, of computer code. The computer moves from one statement to the next in sequence. There are no loops, no branches, and no instructions can be skipped.

Sequence structure follows one path from start to finish.

Iteration (or **repetition**) refers to the number of times a computer passes through a set of instructions. One iteration is once through, and would be shown in a sequence-based code. We tend to use this term when referring to loops in code – for example, how many times (or iterations) will the computer complete this section of a code? There are two forms of iteration: **definite** (also called count control) sets the specific number of times an instruction must be performed and **indefinite** (also called condition control) allows for a variable number of times to perform the instruction, dependent on another factor, such as temperature, or weight or time.

Selection (or **choice**) is best demonstrated using the 'IF, THEN, ELSE' description. If the temperature is 14 degrees Celsius, then turn on the boiler, else leave the boiler off. Most computer code involves some element of selection.

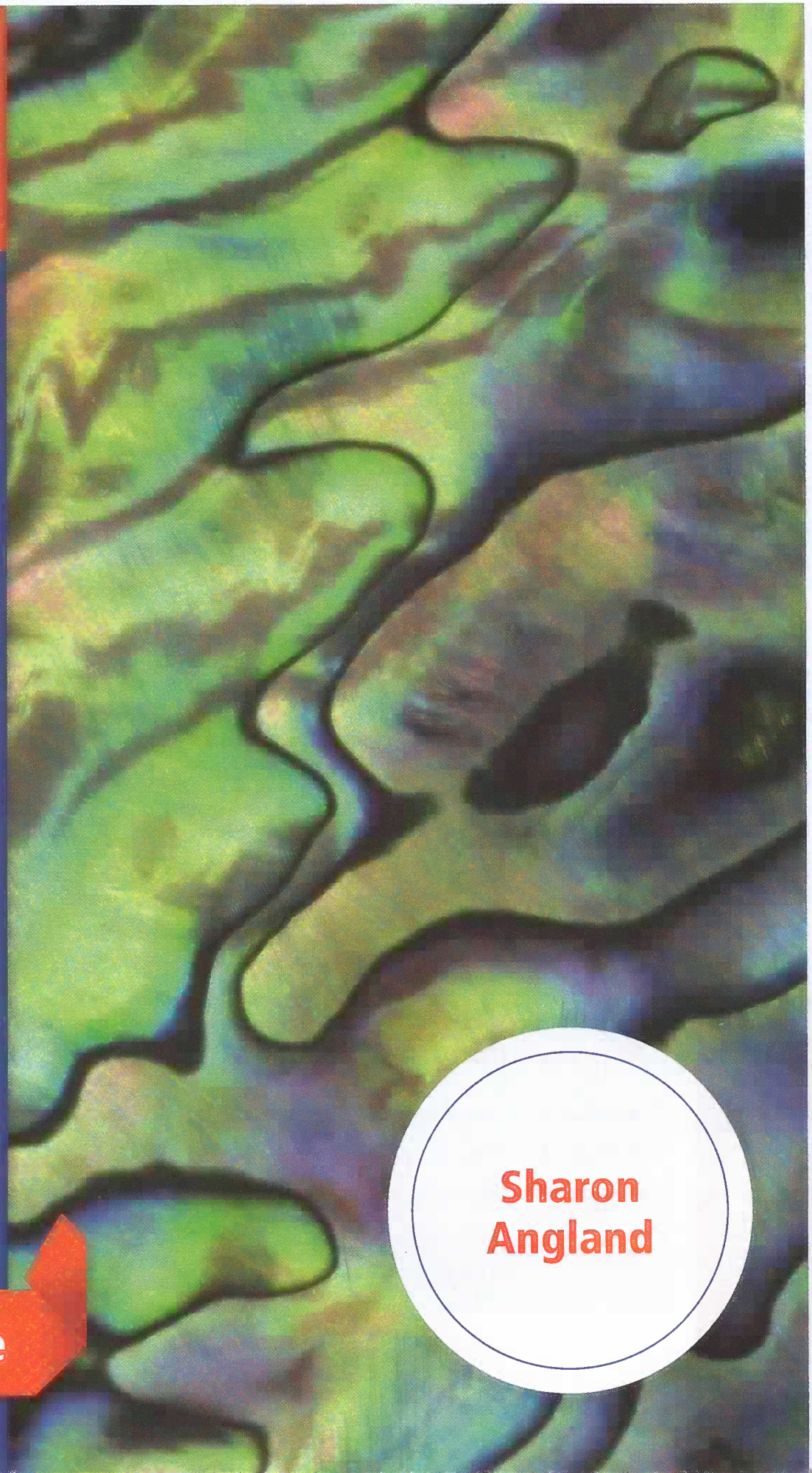


Selection allows for a decision to change the flow of the program.

Letts

GCSE Success

Computer Science



Revision Guide

Sharon Angland