Edexcel A Level Further Maths: Decision Maths 1

Your notes

Algorithms

Contents

General Algorithms

Your notes

Introduction to Algorithms

What is an algorithm?

- An **algorithm** is a set of **precise** instructions, that if strictly followed, will result in the **solution** to a problem
  - Algorithms are particularly useful for programming computers
    - computers can process huge amounts of data and perform millions of calculations in very short time frames
  - Robots would be programmed to follow an algorithm in order to complete a task
    - e.g. a robot vacuum cleaner or lawn mower
  - Algorithms can be performed by human beings too!
    - e.g. the recipe and cooking instructions for a cake, the instructions to build a Lego set

What does an algorithm look like?

- Algorithms may be presented in a variety of ways
  - A list of **instructions,** in order, written in **words/text**
  - **Pseudocode** - this is a mixture of text and very basic computer commands/code
    - **let** statements, **if** statements and **loops** are common but easy enough to follow without any knowledge of a formal computer programming language
  - **Flow charts** are a visual way of presenting the steps of an algorithm
    - they clearly show the order of instructions and any parts of an algorithm that may need repetition

What else do I need to know about algorithms?

- Some algorithms do not always provide an **optimal** (best) solution
  - but will give a solution that is **sufficient** for the purpose
- For example, a band touring the UK may use an algorithm to find the shortest route between the different cities/venues to minimise their travelling expenses
  - if the algorithm is inaccurate by say, 200 miles, it is not going to make much difference overall - the band will be travelling thousands of miles in total so 200 miles is relatively little
- In modern, complicated real-life situations a compromise between the speed and efficiency of an algorithm and the accuracy of the solution it provides is often needed
- When using an algorithm with a **small** amount of data - as exam questions will do due to time restraints - it is tempting to use common sense, intuition and basic mathematical skills to 'see' the solution
  - to show understanding of how an algorithm works it is crucial to stay in **'robot mode'**
    - i.e. follow the algorithm precisely and accurately without taking any 'shortcuts' (however obvious they may seem)

Examiner Tip

- Showing that you have followed an algorithm precisely includes
  - knowing, or using checks built into the algorithm to know, that an algorithm is complete
  - stating (or 'printing') any 'output' at the end if that is what an algorithm instruction requires
    - do not 'assume' the output is the last number in your working

Your notes

Flow Charts

Your notes

What is a flow chart?

- A **flow chart** is a **diagrammatic** way of listing the **instructions** to an algorithm
- Flow charts lend themselves to algorithms that have
  - **conditional** instructions
    - e.g. Is $a > b$? If the answer is yes, the flow chart directs the user to one instruction but if the answer is no, the flow chart directs the user to a different instruction
  - **repetitive parts** or stages

What do the different shaped boxes in a flow chart mean?

- Each command in a flow chart is written inside a shape - the actual shape will depend on the type of command
  - **Ovals** are used to represent the **start** and the **end**
  - **Rectangles** are used to represent **instructions**
  - **Diamonds** are used to represent **questions**

- Alternative names are sometimes used for these boxes
  - The start and end are sometimes called **termination**
  - Question boxes are sometimes called **decision boxes**
  - The word **'print'** may be used instead of 'output' for the **final** answer/result

How do I work with a flow chart?

- Following a flow chart is generally intuitive but make sure you follow the instructions carefully
  - write down any values when instructed to
    - this will often involve completing a table of values
    - if there is a specific instruction to 'output' the final answer, make sure it is separate to the table
- Some questions may ask for a **description** of what the **flow chart/algorithm** produces

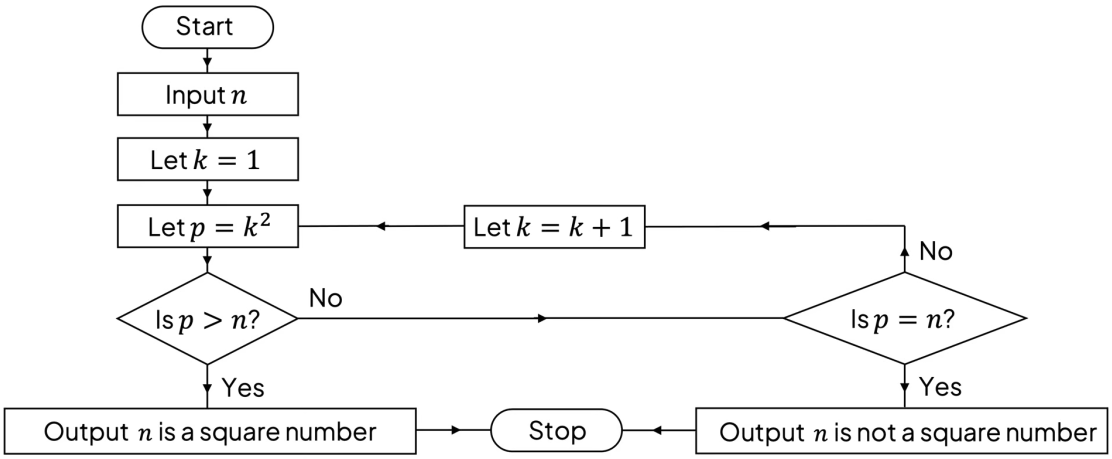
Examiner Tip

- If a question provides a table that is to be filled in with values that change, also bear in mind that some values may *not* change
  - In such instances, only the first entry for a value may end up being entered
    - Not every cell in every row/column will necessarily need completing
    - Follow the instructions from the flow chart ('robot mode'!)

Your notes

Worked example

An algorithm is presented as a flow chart, shown below.



a) Describe what the algorithm achieves.

The final instructions give us a clear idea of what is going on in this case!

The algorithm works out if the input value n is a square number or not

b) For the input $n = 23$, use the flow chart to perform the algorithm.
Complete the table.

n	k	p	Is $p > n$?	Is $p = n$?
Output:				

The first instruction is to input n, so fill 23 under n in the first row of the table

The next two instructions are to let $k = 1$ and to let $p = k^2$, so also in the first row we can fill in 1

Your notes

and 1 for k and p

n	k	p	Is $p > n$?	Is $p = n$?
23	1	1		

The next instruction is the first question, is $p > n$; the answer is no

For the answer no, the flow chart directs us to another question, is $p = n$; the answer is no and so we can complete the top row of the table

n	k	p	Is $p > n$?	Is $p = n$?
23	1	1	NO	NO

The flow chart now updates the value of k; indicate this on the second row of the table

The flow chart then takes us back round the loop of finding and testing the value of p

n does not get updated so only appears in the first row of the table

n	k	p	Is $p > n$?	Is $p = n$?
23	1	1	NO	NO
	2	4	NO	NO

Continuing to precisely follow the flow chart the table can be completed

The output needs stating at the end as per the flow chart instructions

Note how the question "is $p = n$?" does not get considered as the algorithm outputs and stops following the answer 'yes' to the question "is $p > n$?"

n	k	p	Is $p > n$?	Is $p = n$?
23	1	1	NO	NO
	2	4	NO	NO
	3	9	NO	NO

	4	16	NO	NO
	5	25	YES	
Output: **23 is not a square number**				

Your notes

Pseudocode & Text-based Algorithms


Your notes

What is pseudocode?

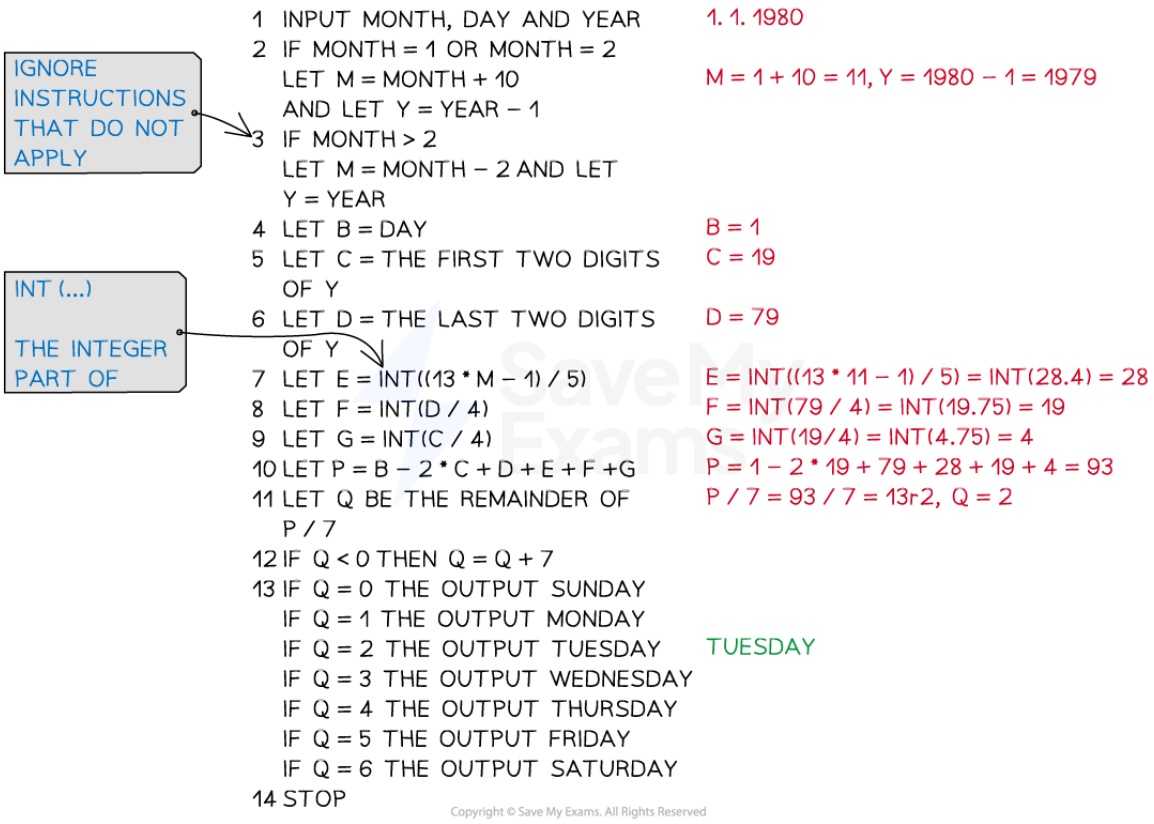
- **Pseudocode** is a way of writing an algorithm that is a mixture of words and computer programming language
- Pseudocode does not follow any conventions that a formal computer programming language (such as Python) would
  - knowledge or experience of computer programming is not expected

What commands are used in pseudocode?

- **Commands** given in pseudocode are intuitive to follow
  - Examples include
    - '**Input** ...' - the value(s)/data that the algorithm needs to be told
    - '**Let** ..." - assign or update a value to a variable
    - '**If** ...' - a conditional statement, the outcome of which will lead to different parts of the algorithm
    - '**Int** ...' - the **integer** part of a value
    - '**While** ...' - whilst the condition that follows is true, keep doing
    - '**Output** ...' or '**Print** ...' - usually at the end of an algorithm this instruction essentially means 'write down the final answer' !

How do I work with an algorithm in pseudocode?

- To show an algorithm in pseudocode has been followed **precisely**, the result of each command is usually jotted down
  - this may be in the form of a table
  - some values may be changed or updated during the algorithm
- Ignore any commands that do not apply
  - e.g. in an if statement
- **Zeller's algorithm** is given in pseudocode below
  - This finds the day of the week for any date in the Gregorian calendar (1582 onwards)
    - This version of the algorithm assumes that days will be inputted as 1–31, months 1–12 (rather than words) and years are four digits
  - The algorithm may give inaccurate results or 'crash' if unexpected values are input
    - Try using the algorithm to find the day of the week 30th February 2024 falls on
    - No such date exists, but 29th February 2024 is a Thursday - does the algorithm give the answer Friday?



- Zeller's algorithm is most commonly used to find which day of the week on which a person was born using their date of birth
- There are many variations of Zeller's algorithm, some are more complicated than others!

What are text-based algorithms?

- **Text-based algorithms** refer to instructions that are given in sentences
  - Each instruction may be labelled with 1, 2, 3, and so on
  - Just like in many of our **Save My Exams** Revision Notes!
    - Step 1 ..., Step 2 ..., etc
- The following text-based algorithm is for a very basic single player dice game
  - The aim of the game is to minimise the number of rolls of the dice needed to reach 'Stop'

Your notes

1	ROLL A DICE
2	IF THE NUMBER SHOWN IS EVEN AND LESS THAN SIX, SCORE FOUR POINTS
3	IF THE NUMBER SHOWN IS ODD SCORE TWO POINTS
4	IF THE NUMBER SHOWN IS SIX SCORE ZERO POINTS
5	REPEAT INSTRUCTIONS 1 TO 4 UNTIL THE TOTAL SCORE IS AT LEAST TWENTY
6	STOP


Your notes

Order of an Algorithm

Your notes

What is the order of an algorithm?

- The **order** of an **algorithm** refers to its complexity
  - The complexity of an algorithm will depend on
    - the number of **steps/instructions** involved
    - the amount of **'input'** data
  - A well constructed algorithm will be both **accurate** and **efficient**
- The **order** of an **algorithm** is usually determined by the **maximum** number of steps an algorithm involves
  - For example, an algorithm that finds a particular item in a list would, at most, search every item on that list
    - the number of steps involved is a function of the number of items on the list
    - the algorithm would be of **linear** order
  - **Quadratic** order would mean the number of steps involved is a function of the **square** of the size of the input data
  - **Cubic** order would be a function of the **cube** of the size of the input data

How is the order of an algorithm described?

- The order of an algorithm is described by a mathematical **function**
  - **square roots, logarithms, exponentials,** etc
  - **linear, quadratic** and **cubic** are the most common
- Two different algorithms may both be of **quadratic** order but still take differing amount of times to complete
  - this is because the **exact form** of the **quadratic functions** may differ
  - e.g. For an input of size n, one of the algorithms may have the number of steps $\frac{1}{2}n(n+1)$ but the other $\frac{1}{2}n(n-1)$
    - both are quadratic functions but for the same value of n $(n \neq 0)$ they take different values

How do I find the order of an algorithm?

- In simple cases the order of an algorithm can be determined by considering the **maximum number** of **steps** involved
  - the **order** is determined by considering the *worst case* scenario
  - an algorithm finding an item in a list would, *at most,* search every item in the list
    - the algorithm is therefore of **linear** order
- In many cases - particularly where the order is not easily determined, it will be given
- The **order** of an **algorithm** can be used to find an **approximate time** of **completion**
  - The time will only be an **approximation** as
    - only the **order** (not the exact function) is being used
    - the order is determined by the **maximum number** of **steps** involved (the worst case scenario)

How do I find (an estimate for) the time an algorithm will take to complete using its order?

Your notes

- The time it takes an **algorithm** to complete is **estimated** using **proportionality**
  - **Doubling** the size of the input data to an algorithm of **linear** order would increase the **approximate** time of completion **2** times
  - Doubling for **quadratic** order would increase the **approximate** time of completion $2^2 = 4$ times
  - Doubling for **cubic** order would increase the **approximate** time of completion $2^3 = 8$ times
- The same use of proportion applies to less common orders too
  - for an algorithm of order $\sqrt{n}$, **doubling** the size of the input data would increase the **approximate** time of completion $\sqrt{2}$ times
  - e.g. if an algorithm takes 5 seconds to process an input of size 20, then the algorithm will take approximately $5 \times \sqrt{4} = 5 \times 2 = 10$ seconds to process an input of size 80
- Depending on the nature of an algorithm, the use of the letter n may take different meanings
  - the 'input size' n may refer to the ***number of data items*** or it could be the ***actual size*** of the input
    - e.g. in a sorting algorithm n may be the number of items to be sorted but in an algorithm to determine if a number is prime, n may be the number being tested
  - when referring to the **order** of an algorithm, n may be the **scale** by which the input size has **increased**
    - e.g. in an algorithm of **quadratic** order, increasing the input size by n, will approximately increase the time the algorithm takes to complete by n^2

Examiner Tip

- The words order and complexity are often used interchangeably
  - **quadratic order** and **quadratic complexity** are the same thing

Worked example

An algorithm for sorting a list of n values into ascending order has order $n \ln n$.
A computer running the algorithm takes 0.12 seconds to apply the algorithm to a list of 3000 values.

Estimate the time it would take the computer to apply the algorithm to a list of 60 000 values.It takes 0.12 seconds for an algorithm of order $3000 \ln (3000)$

Divide the time by the order and then multiply it by the order of an algorithm for a list with 60 000 items

$$\frac{0.12}{3000 \ln (3000)} \times 60000 \ln (60000) = 3.298 \ldots$$

An estimate of the time the computer would take to apply the algorithm to 60 000 values is 3.30 seconds (3 s.f.)

Your notes

Sorting Algorithms


Your notes

Introduction to Sorting Algorithms

- **Sorting algorithms** arrange items (usually **values, letters** or **words)** into **ascending** or **descending** order
  - whilst arranging a (small) list of items is easy for a human being, machines, robots and computers, etc would need programming to do so
  - as a list of items becomes larger, it becomes increasingly difficult for a human to sort
    - programming a computer to use a sorting algorithm makes the process both **accurate** and **efficient** (first time!)
- With sorting algorithms in particular, it is important to stay in **'robot mode'**
  - do not be tempted to take shortcuts or miss parts of the algorithm out because the answer can be 'seen'
  - follow each step of the **algorithm precisely,** in **order,** exactly as a robot would

Bubble Sort

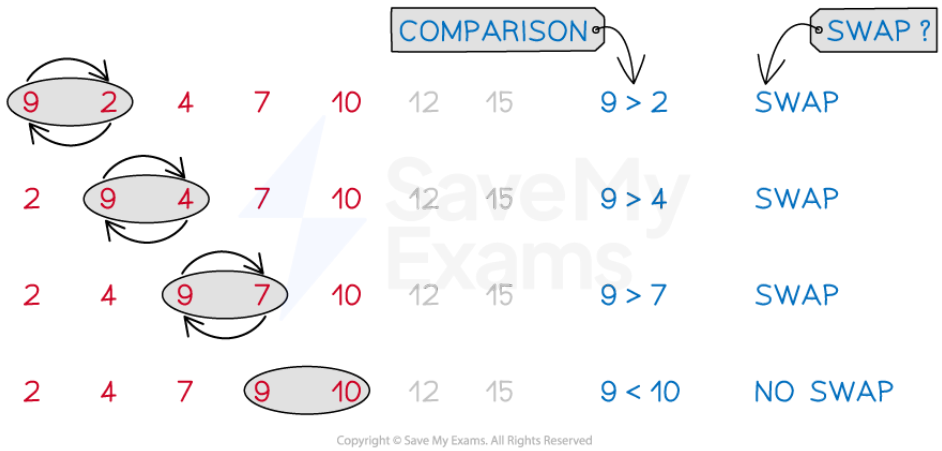
Your notes

What is the bubble sort algorithm?

- The **bubble sort** algorithm arranges items into either **ascending** or **descending** order
  - Items are usually **values**
    - but could be letters or words and similar
    - for questions given in context **values** will be measures such as **weights, lengths, scores** or **times**

How does the bubble sort algorithm work?

- **Bubble sort** works by **comparing** pairs of items on the **working list**
  - the first item is **compared** to the second, the second to the third, and so on
  - a **swap** occurs if a pair of items are not in order
    - a pair of **equal** items would **not count** as a swap



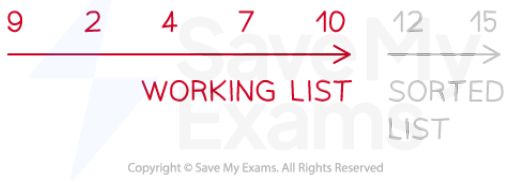
- This comparison of pairs and possible swaps continues until the end of the working list is reached
  - this is called a **pass**
  - several **passes** are usually required to order a list of items
- At the **end** of each **pass,** the item at the end of the **working list** is in the **correct** place
  - for **ascending** order, the **highest value** 'bubbles' to the **top**
- Bubble sort, for n items, is **complete** when **either**
  - a **pass** involves with **no swaps,** or,
  - after the $(n-1)^{\text{th}}$ pass
    - i.e. when only **one item** would remain on the **working list**
      - so comparisons are neither possible nor necessary

What is the working list?

- The **working list** is the list of items that a pass of the bubble sort algorithm will apply to
  - For the **first pass every item** on the list is in the **working list**

Your notes

- After the **first pass,** the **final item** is in the correct place
  - other items may be too but the algorithm **ensures** the **highest** (for an ascending list) or **smallest** (for descending) item is at the **end** of the list
  - this item need not be involved in further comparisons and so it is removed from the working list
- After the **second** pass, the **last two** items will be in the **correct place**
  - After the **third** pass, the last **three** items will be correctly sorted, and so on
  - Reducing the working list by one after each pass helps keep the bubble sort **efficient** as possible by not requiring unnecessary comparisons
- For a list of n items
  - the first pass will have n items on the working list
  - the second pass will have $n-1$ items on the working list
  - the third pass will have $n-2$ items on the working list
  - (if required) the $(n-1)^{\text{th}}$ pass will have 2 items on the working list
- Alongside the **working list** for **unordered** items, the **ordered items** may be referred to as the **sorted list**
- The image below shows a list at the end of the **second pass** of an **ascending bubble sort**
  - **two** items are on the sorted list (12 and 15)
  - the 10 being in the correct position is irrelevant at this stage ('robot mode'!)
  - the *third* pass will confirm its position and that's when it will become part of the sorted list



How do I work out the (maximum) number of passes in the bubble sort algorithm?

- For a list of n items
  - the **maximum number** of **passes** will be $n-1$
    - this is because after the $(n-1)^{\text{th}}$ pass, $n-1$ items will be in order, so the last remaining item must be in order too
    - without actually carrying out a few passes it is hard to predict or work out how many passes bubble sort will need
      - but it may be possible to tell how many **more passes** are required when only a few items are out of order
  - If the item **at the end** of the list **should be at the start** then will take $n-1$ passes
    - This is because this item will **only move one space closer** to the start after each pass

How do I work out the (maximum) number of comparisons in the bubble sort algorithm?

- For a list of n items

Your notes

- the **number** of **comparisons** at **each pass** will always be **one less** than the **number** of **items** on the **working list**
  - the first pass will have n items are on the working list so there will be $n-1$ comparisons
  - the second pass will have $n-2$ comparisons
  - the k^{th} pass will have $n-k$ comparisons
  - (if required) the $(n-1)^{\text{th}}$ pass will have 1 comparison
- the **maximum number** of **comparisons** for the entire bubble sort algorithm would be

$$(n-1)+(n-2)+(n-3)+\ldots+3+2+1$$

or

$$\sum_{r=1}^{n-1} r = \frac{1}{2}n(n-1)$$

- understanding the logic for the number of comparisons is easier than trying to remember a formula
- the final formula, $\frac{1}{2}n(n-1)$ is quadratic and so **bubble sort** is of **quadratic order**

How do I work out the (maximum) number of swaps in the bubble sort algorithm?

- For a list of n items
  - the **maximum number** of **swaps** in a pass would be the same as the **number** of **comparisons** for that **pass**
    - i.e. every comparison results in a swap being made
    - this would occur if the **first item** on the working list is the **largest** (or smallest for descending)
      - so the **maximum number** of swaps for the **entire algorithm** would be needed to if a list started in **reverse** order
    - with a **small** working list it is possible to 'see' or count the number of swaps required for a pass
      - it can be awkward keeping track in longer lists so be wary of relying on this sort of **inspection** technique
      - keeping a **tally** of swaps as you perform a pass would be more reliable

Examiner Tip

- Think '**robot mode**' !
  - A crucial part of bubble sort is the **last pass** has **no swaps**
    - If you are not following the algorithm precisely, this pass can easily be missed out
- Make it clear when you have completed the bubble sort algorithm, and state how you know it is complete
  - e.g. "there were no swaps in the fifth pass so the algorithm is complete and the list is in order"
  - e.g. (for 10 items) "that is the end of the 9^{th} pass so the algorithm is complete and the list is in order"
- After completing your solution, double-check whether a question required the list to be in **ascending** or **descending** order
  - If you've got it the wrong way round, there's no need to redo the question but
    - make it clear that after the algorithm is completed, you are reversing the list
    - make sure your **final answer** is in the **order required** by the **question**


Your notes

Your notes

Worked example

The time, to the nearest second, taken by 7 participants to answer a general knowledge question are 5, 7, 4, 9, 3, 5, 6.
The times are to be sorted into ascending order using the bubble sort algorithm.

a) Write down

i) the maximum number of passes the algorithm will require,

ii) the number of comparisons required for the third pass,

iii) the number of swaps there will be in the first pass.

i) The maximum number of passes is $n - 1$

$$n = 7$$
$$7 - 1 = 6$$

The maximum number of passes will be 6

ii) The k^{th} pass will have $n - k$ comparisons

$$7 - 3 = 4$$

The number of comparisons required in the third pass will be 4

iii) Carefully do this by inspection (or you can carry out the first pass to be sure but that takes time)
The first swap will be the 7 and 4
The 9, being the highest value on the list, will then swap with each of the values that follow it (3 values)

The number of swaps in the first pass is 4

b) After the second pass of the bubble sort algorithm, the times are in the order 4, 5, 3, 5, 6, 7, 9.
Perform the third pass of the bubble sort algorithm and state the number of swaps made.
As this is the third pass, the last two items on the list, 7 and 9, will already be in the correct place
The working list (that we apply a pass of the bubble sort algorithm to) will be 4, 5, 3, 5, 6

4	5	3	5	6	7	9	4 < 5, no swap
4	5	3	5	6	7	9	5 > 3, swap
4	**3**	**5**	5	6	7	9	5 = 5, no swap
4	3	5	5	6	7	9	5 < 6, no swap
4	**3**	**5**	**5**	**6**	**7**	**9**	**end of pass 3**

Number of swaps: 1

c) Explain why two further passes of the bubble sort algorithm are required to ensure the list is in ascending order.

At this stage of the algorithm the answer should be able to be deduced by inspection of the current list
Using the answer to part b), the current list is 4, 3, 5, 5, 6, 7, 9

The next pass will involve one swap – the 4 and the 3
The pass after that will involve no swaps and so the algorithm will be complete and the list will be in order
Thus, two further passes are required

It is important to state that the algorithm is complete and the reason why


Your notes

Quick Sort


Your notes

What is the quick sort algorithm?

- The quick sort algorithm arranges items into either **ascending** or **descending** order
  - Items are usually **values**
    - but could be letters or words and similar
    - for questions given in context **values** will be measures such as **weights, lengths, scores** or **times**

How does the quick sort algorithm work?

- Each **pass** of the **quick sort algorithm** works by splitting a **sub-list** of the items into two halves around a **pivot**
  - one half will be the **sub-list** of values that are **less** than the pivot
    - for **ascending** order this sub-list would come **before** the pivot
  - the other half will be the **sub-list** of values that are **greater** than the pivot
    - for **ascending** order this sub-list would come **after** the pivot
  - values that are **equal** to the **pivot** can be listed in either half
    - for **consistency** we suggest **always** listing items **equal** to the **pivot** in the half **greater** than the pivot
  - on both sides of the pivot values would be listed in the **order** they appeared in the **original** list
- The quick sort algorithm is **complete** when every item on the (original) list has been designated as a **pivot**

Which item is the pivot in the quick sort algorithm?

- The **pivot** is the middle value in a **sub-list** of the items, **without** reordering
  - If there are **two items** in the middle of a **sub-list** then either item can be taken as the **pivot**
    - for **consistency** we suggest **always** choosing the item in the **upper position** as the **pivot**
- In the first pass, there will be one pivot - the middle item in the whole list
- In the second pass, there *could* be **two** (new) **pivots** - one in the lower half/sub-list, one in the upper half/sub-list
  - The only time this would **not** be the case is when the pivot is the **lowest (ascending)** or **greatest (descending)** item on a sub-list
  - The number of pivots *could* **double** at each **pass**

How do I apply the quick sort algorithm?

- The following list of steps describe the **quick sort algorithm** for **arranging** a list of items into **ascending** order
  - Arranging into descending order would be the same, but the 'ordering' in Step 2 would be reversed
- **STEP 1**
  Find and **highlight** the **pivot (middle** value) for each **sub-list** of items
  For the **first pass** only, the sub-list will be the same as the entire list of items

- **STEP 2**
  List items **lower** than the **pivot,** in the order they appear, **before** the **pivot**, creating a new (smaller) sub-list
  Then write the pivot as the next number on the list
  List items **greater** than or **equal** to the **pivot,** in the order they appear, **after** the **pivot**, creating another new (smaller) sub-list
- **STEP 3**
  Repeat steps 1 and 2 until every item on the original list has been designated a pivot
  The **original** (full) list of items will now be in **ascending** order

Examiner Tip

- Make it clear which items you have chosen as pivots at each pass of the quick sort algorithm
  - However, do not use different colours on the exam paper - they will not show up
    - (papers are usually scanned into marking systems in black and white)
  - use different styles instead such as
    - **underlining** values using solid, dotted, wavy, double, etc lines
    - drawing different **shaped boxes** (rectangle, circle, star, etc) around values


Your notes

Worked example

The following values are to be sorted into descending order using the quick sort algorithm.

5, 3, 7, 2, 4, 5, 9, 6

Clearly showing your choice of pivots at each pass, perform the quick sort algorithm to sort the list into descending order.

- **STEP 1**
  This is the first pass so the entire list will be the sub-list
  There are 8 values so the pivot will be the 5th value on the list
  Pass 1 (pivot 4)

  5 3 7 2 (4) 5 9 6

- **STEP 2**
  As **descending** order is required, list items greater than the pivot (4) before it; items lower than the pivot after it

  5 7 5 9 6 (4) 3 2

- **STEP 3**
  The two sub-lists 5, 7, 5, 9, 6 and 3, 2
  Repeat steps 1 and 2 on these sub-lists
  Pass 2 (pivots 5 and 2)

  5 7 [5] 9 6 (4) 3 [2]

  The (repeated) 5 will go into the 'greater than or equal to' sub-list (before the pivot 5 as descending)
  There is no sub-list after the pivot 2 as it is the lowest value in its current sub-list

  7 9 6 5 [5] (4) 3 [2]

  Continuing repeating steps 1 and 2 on each new sub-list until every item has been designated a pivot
  Pass 3 (pivots 6 and 3)

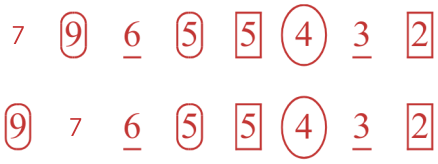
  7 9 6 5 [5] (4) 3 [2]

  7 9 6 5 [5] (4) 3 [2]

  Notice that pass 3 did not involve any reordering but it is still crucial to show the pivots so that the algorithm has been precisely followed

Your notes

Pass 4 (pivots 9 and 5)



Pass 5 (pivot 7)

Every value has now been designated as a pivot so the algorithm is complete and the list is in descending order

9, 7, 6, 5, 5, 4, 3, 2

Your notes

Bin Packing Algorithms


Your notes

Introduction to Bin Packing Algorithms

- **Bin packing algorithms** organise **objects** into as few **containers** as possible
- The **containers** are called **bins**
  - In problems **all bins** will be of equal **size**
    - **Size** may refer to **length, width, capacity** (volume), **weight**, etc
- **'Packing'** is a loose term that could mean **'separating', 'cutting'** or similar
- Examples of bin packing problems include
  - loading **pallets (objects)** of varying weights into **lorries (bins)** without exceeding their weight **capacity**
    - with the aim of **minimising** the number of lorries required
  - cutting short **strips of cable (objects)** from **large reels of cable (bins)**
    - with the aim of **minimising wastage** at the end of a reel
- There are **two** bin packing algorithms covered
  - the **first-fit bin packing** algorithm
  - the **first-fit decreasing bin packing** algorithm
  - Both are **heuristic** algorithms - they will provide a (good) solution
    - but **not** necessarily an **optimal** solution
    - and **not** necessarily the **only** solution
- The full-bin algorithm is also covered

How do I find the lower bound for the number of bins?

- The **lower bound** (LB) for the **number** of **bins** required in the first-fit bin packing problem is the **smallest integer** that **satisfies**

$$\text{LB} \geq \frac{\text{Total capacity of all objects}}{\text{Size of each bin}}$$

- As it is a **number** of bins, the **lower bound** will be an **integer**
  - even if the calculation gave the value 3.01, say, more than 3 bins would be required, and so the lower bound will be 4
  - i.e. always **round up** (unless the value happens to be an exact integer)
- Note that this is exactly the same for all bin-packing algorithms

First-Fit Bin Packing Algorithm


Your notes

What is the first-fit bin packing algorithm?

- The **first fit bin packing algorithm** packs **objects** into **bins** in the **order** in which they are **presented (listed)**
  - e.g. organising scattered boxes in the order they are encountered instead of organising them first
- The algorithm finds the number of bins required to pack all the objects
  - ideally this would be the **minimum number** of bins required
  - but the algorithm will not necessarily provide the **optimal** solution
- An **advantage** of using the first-fit bin packing algorithm is **speed**
  - the objects do not have to be **sorted** (into size or any other criteria) first
  - in many cases, the **solution** obtained, will be **sufficient** for **business** or **practical** purposes
    - i.e. speed **(efficiency)** is of more importance than **optimisation**

How do I apply the first-fit bin packing algorithm?

- Each **object**, in turn, is placed in the **first available bin** that has enough **capacity** (room) for it
  - Object 1 (the first on the list) will be placed into bin 1
    - if bin 1 then has enough spare capacity, object 2 (the second on the list) will also go into bin 1
    - if not, object 2 will need a new bin, bin 2
    - object 3 would be considered for bin 1 first, then bin 2, and failing both, it would require a new bin, bin 3
- For each object, if any existing bin **does not** have enough capacity remaining, a **new bin** will be required
  - For example,
    - if a bin has capacity 10 and the first two objects are of capacity 6 and 3, the 6 will go into bin 1, then the 3 will go into bin 1 too
      $(6 + 3 = 9 \leq 10)$
    - if the first two objects are of capacity 5 and 9, the 5 would go in bin 1, then the 9 would go into bin 2
      $(5 + 9 = 14 > 10)$
- The algorithm is **complete** when all **objects** have been placed in a **bin**

Examiner Tip

- As you place values into bins, you may find it helpful to jot down either the **total capacity used** in each bin or the **remaining capacity** in each bin next to it
  - However, **do not** include these as part of your **final answer**

Your notes

Worked example

A gym worker is tidying up by stacking weights on shelves. The weights, given in kilograms, are stated below.

12, 16, 18, 8, 16, 12, 10, 4, 6, 10, 12, 16, 20, 10, 24

Each shelf has a load capacity of 40 kg.

a) Find a lower bound for the number of shelves needed so all the weights can be stacked on the shelves.

The lower bound will be (greater than or equal to) the total of the weights divided by the capacity (load) of one shelf

$$LB \geq \frac{12+16+18+8+16+12+10+4+6+10+12+16+20+10+24}{40}$$

$$LB \geq \frac{194}{40}$$

$$LB \geq 4.85$$

The number of shelves (bins) will need to be an integer (hence the inequality), more than four shelves are needed so round up to five

Lower bound for the number of shelves is 5

b) Use the first-fit bin packing algorithm to find the number of shelves needed to stack all of the weights.

In this case, a 'shelf' will be a 'bin'
In the first-fit bin packing algorithm, we deal with each weight in turn as they are listed
(To help, we have written the remaining capacity in brackets next to each shelf in the working area, this is optional)
The first weight, 12 kg, will go on the first shelf

Shelf 1: 12 [28]

The second weight, 16 kg will also fit on the first shelf

Shelf 1: 12 16 [12]

Next, the 18 kg will need to go on to a new shelf (shelf 2)

Shelf 1: 12 16 [12]

Shelf 2: 18 [22]

The 8 kg weight will go on the first shelf

Shelf 1: 12 16 8 [4]

Shelf 2: 18 [22]

Continuing in this manner, the 16 kg weight can join shelf 2; the second 12 kg weight will need a third shelf

Shelf 1: 12 16 8 [4]

Shelf 2: 18 16 [6]

Shelf 3: 12 [28]

Continuing in order - a fourth shelf is needed for the third 12 kg weight and the 20 kg and 24 kg weights require a shelf of their own
When finished, double check each shelf does not exceed 40 kg in total and do not include the spare capacities with the final answer

Shelf 1: 12 16 8 4

Shelf 2: 18 16 6

Shelf 3: 12 10 10

Shelf 4: 12 16 10

Shelf 5: 20

Shelf 6: 24

Notice that, whilst the lower bound was 5 shelves, the algorithm used 6 shelves

Your notes

First-Fit Decreasing Bin Packing Algorithm

Your notes

What is the first-fit decreasing bin packing algorithm?

- The **first fit decreasing bin packing algorithm** packs **objects** into **bins** once they have been arranged into **decreasing (descending) order**
  - e.g. arranging scattered boxes in order of **size/weight** first, then organising them, in **descending order, starting** with the **biggest/heaviest**
- The algorithm finds the number of bins required to pack all the objects
  - ideally this would be the **minimum number** of bins required
  - but the algorithm will not necessarily provide the **optimal** solution
- An **advantage** of using the first-fit decreasing bin packing algorithm is that it usually gets **close** to the **optimal** solution
  - this would be where **minimising** the **number** of **bins** is more desirable than **speed**

How do I apply the first-fit decreasing bin packing algorithm?

- Each **object**, in turn, is placed in the **first available bin** that has enough **capacity** (room) for it
  - Object 1 (the biggest) will be placed into bin 1
    - if bin 1 then has enough spare capacity, object 2 (the second biggest) will also go into bin 1
    - if not, object 2 will need a new bin, bin 2
    - object 3 would be considered for bin 1 first, then bin 2, and failing both, it would require a new bin, bin 3
  - For each object, if any existing bin **does not** have enough capacity remaining, a **new bin** will be required
  - For example,
    - if a bin has capacity 15 and the first two objects are of capacity 13 and 10, the 13 will go into bin 1, then the 10 will go into bin 2
      (as $13 + 10 = 23 > 15$)
    - if the first two objects are of capacity 8 and 6, the 8 would go in bin 1, then the 6 would go into bin 1 too
      (as $8 + 6 = 14 \leq 15$)
- The algorithm is **complete** when all **objects** have been placed in a **bin**

Examiner Tip

- A question may have a previous part that asks you to use a **bubble sort** or a **quick sort** to arrange values into descending order
- If not, you may be asked to name a suitable algorithm that would sort values into descending order
  - in which case **'bubble sort'** or **'quick sort'** can be stated
  - you may be asked to briefly explain these algorithms
- As you place values into bins, you may find it helpful to jot down either the **total capacity used** in each bin or the **remaining capacity** in each bin next to it
  - However, **do not** include these as part of your **final answer**

Worked example

A gym worker is tidying up by stacking weights on shelves. A list of the weights at the gym, given in kilograms, in descending order is stated below.

24, 20, 18, 16, 16, 16, 12, 12, 12, 10, 10, 10, 8, 6, 4

Each shelf has a load capacity of 40 kg. The lower bound for the number of shelves required to stack all of the weights is 5.
Use the first-fit decreasing bin packing algorithm to stack the weights on to as few shelves as possible, explaining how you know your answer is optimal.

A 'shelf' will be a 'bin'
For the decreasing first-fit bin packing algorithm, assign the weights starting with the heaviest, in descending order
The first weight, 24 kg, will go on the first shelf

Shelf 1:	24		[16]

The second weight, 20 kg will not fit on the first shelf so a new shelf is needed

Shelf 1:	24		[16]
Shelf 2:	20		[20]

Shelf 2 has enough capacity to take the next, 18 kg weight

Shelf 1:	24		[16]
Shelf 2:	20	18	[2]

The first of the 16 kg weights can fill shelf 1

Shelf 1:	24	16	[0]
Shelf 2:	20	18	[2]

Thinking ahead we can now see that the lowest weight is 4 kg, but there is only 2 kg of spare capacity in the first two shelves
Therefore we know we can no longer use shelves 1 or 2
The other two 16 kg weights can both go on to shelf 3

Shelf 1:	24	16	[0]
Shelf 2:	20	18	[2]
Shelf 3:	16	16	[8]

Continuing in order - a fourth shelf is needed for the three 12 kg weights (leaving 4 kg spare) and so a fifth shelf is needed for the three 10 kg weights (leaving 10 kg spare)

Your notes

There is then spare capacity on shelves 3, 5 and 4 respectively for the 8 kg, 6 kg and 4 kg weights
Double check your final answer and do not include the spare capacities

Shelf 1:	**24**	**16**		
Shelf 2:	**20**	**18**		
Shelf 3:	**16**	**16**	**8**	
Shelf 4:	**12**	**12**	**12**	**4**
Shelf 5:	**10**	**10**	**10**	**6**

This uses the same number of bins as the lower bound stated in the question

This solution is optimal as the number of bins required is the same as the lower bound given

Other arrangements on 5 shelves may be possible


Your notes

Full-Bin Packing Algorithm


Your notes

What is the full-bin packing algorithm?

- The **full-bin packing algorithm** identifies **combinations of objects** that will **fill a bin**
  - Any objects that **cannot** be grouped to fill a bin will be packed using the **first-fit** bin packing algorithm
- This algorithm finds the number of bins required to pack all the objects
  - Ideally this would be the **minimum number** of bins required
- An **advantage** of the full-bin packing algorithm is that it generates a **good solution**
  - It will often produce the **optimal** solution
- Full-bin combinations are **identified by inspection**
  - This can be **time-consuming** for a situation with many objects
  - It is **not** suitable for situations where **speed** is important

How do I apply the full-bin packing algorithm?

- The list of **objects** is inspected to identify combinations of objects that fill a bin
  - As many **combinations of full-bins** as possible from the original list should be identified and placed into bins
    - E.g. For the list of objects: 2, 6, 4, 9, 8, 8, 2, 5, 2 and a bin capacity of 10
    - A possible set of full-bin combinations is: 2 + 8, 6 + 4, 8 + 2
    - The remaining objects are 9, 5 and 2
  - Any **remaining objects** should be placed into bins using the **first-fit** bin packing algorithm
    - E.g. for the remaining objects above
    - 9 will be place in the next available bin
    - 5 and 2 will be placed in an additional bin
- The algorithm is **complete** when all **objects** have been placed in a **bin**

Examiner Tip

- As this algorithm is mostly done by inspection, be very careful as you find the full-bin combinations
  - Check off each item as it is included in a bin
  - Check that you have the same number of objects in your bins as there are in the original list

Worked example

A gym worker is tidying up by stacking weights on shelves. The weights, given in kilograms, are stated below.

12, 16, 18, 8, 16, 12, 10, 4, 6, 10, 12, 16, 20, 10, 24

Each shelf has a load capacity of 40 kg.

Use the full-bin packing algorithm to find the number of shelves needed to stack all of the weights.

In this case, a 'shelf' will be a 'bin'

In the full-bin packing algorithm, we inspect the original list for combinations equal to the capacity of each bin

Find combinations of weights that add to 40 and place on shelves

Shelf 1:	12	12	16	
Shelf 2:	8	10	10	12
Shelf 3:	16	24		
Shelf 4:	16	20	4	

Sort the remaining objects in the order they appear using the first-fit algorithm

The remaining objects, in order, are 18, 6 and 10

These will all fit in a single bin

Shelf 1:	**12**	**12**	**16**	
Shelf 2:	**8**	**10**	**10**	**12**
Shelf 3:	**16**	**24**		
Shelf 4:	**16**	**20**	**4**	
Shelf 5:	**18**	**6**	**10**	

The lower bound is 5 shelves so the solution is optimal
There are other possible 5–shelf solutions using the full-bin algorithm

Your notes