Stan Dolan

# Discrete Mathematics

## For AQA

1

CAMBRIDGE
UNIVERSITY PRESS

PUBLISHED BY THE PRESS SYNDICATE OF THE UNIVERSITY OF CAMBRIDGE
The Pitt Building, Trumpington Street, Cambridge, United Kingdom

CAMBRIDGE UNIVERSITY PRESS
The Edinburgh Building, Cambridge CB2 2RU, UK
40 West 20th Street, New York, NY 10011-4211, USA
477 Williamstown Road, Port Melbourne, VIC 3207, Australia
Ruiz de Alarcón 13, 28014 Madrid, Spain
Dock House, The Waterfront, Cape Town 8001, South Africa

http://www.cambridge.org

First published 2000
Reprinted 2003

Printed in the United Kingdom at the University Press, Cambridge

*Typefaces* Times, Helvetica *Systems* Microsoft® Word, MathType™

*A catalogue record for this book is available from the British Library*

ISBN 0 521 79941 4 paperback

Cover image © DigitalVision
Underground Map on pages 19 and 22 reproduced with permission from
London's Transport Museum, Registered User no. 02/3870

# Contents

Introduction v

**1** Algorithms 1
**2** Graphs and networks 19
**3** Minimum connector problems 32
**4** Finding the shortest path 51
**5** Matching 60
**6** Route inspection 70
**7** The travelling salesperson problem 81
**8** Linear programming 94

Revision exercise 110
Mock examinations 116

Answers 121
Glossary 134
Summary of algorithms 136
Index 137

# 1 Algorithms

This chapter looks at the meaning of 'Discrete Mathematics' and introduces some algorithms. When you have completed it you should

- know what an algorithm is
- be able to apply the algorithms known as Bubble Sort, Shuttle Sort, Shell Sort and Quicksort
- know what is meant by correctness, finiteness, generality and stopping conditions.

## 1.1 What is Discrete Mathematics?

You will have already met, in Statistics, the distinction between continuous and discrete data. Continuous data can take any value in a numerical range: measurements of height, weight and time all produce continuous data. Discrete data can only take values which are strictly separated from each other: measurements of the number of children in a family or the number of letters in a word are discrete data which take only whole-number values.

In the 17th century, Sir Isaac Newton and other leading mathematicians started the development of calculus, which deals specifically with continuous data, and graphs which are generally smooth. Discrete Mathematics deals only with branches of mathematics which do *not* employ the continuous methods of calculus.

However, the distinction between continuous and discrete sometimes becomes blurred. For example, computers essentially deal in Discrete Mathematics, because they hold numbers using sequences of 1s and 0s, and can only hold a finite amount of information. However, advanced computers can work to a very high degree of accuracy, and can do very good approximations to continuous mathematics. They can give approximate solutions to equations which otherwise could not be solved.

Computer screens are divided into 'pixels' (the word is a contraction of 'picture elements'), and so computer and TV screens are essentially discrete devices. However, because the discrete pixels are so small, the images on the screen appear continuous.

But all this is only part of the definition of Discrete Mathematics. It is also widely (but not universally) accepted that Discrete Mathematics is restricted to branches of mathematics whose development has mainly been in the 20th century. It is no coincidence that its importance and application have arisen in the same period of history as the development of computers.

Part of working with computers is the idea of a procedure, or 'algorithm', to solve a problem. You probably know an algorithm which enables you to find the answer to a long multiplication given the two numbers you wish to multiply. Algorithms form a substantial part of Discrete Mathematics. In this course, most of the algorithms will be topics related to the best use of time and resources. These have applications in industry, business, computing and in military matters.

## 1.2 Following instructions

An algorithm is a sequence of instructions which, if followed correctly, allows anyone to solve a problem.

The mathematics problems studied in school tend to be those for which previous generations of mathematicians have already worked out the appropriate sequences of instructions. For example, consider this algorithm for finding the median of a set of numbers.

| **Find the median** | | *Example* 12, 2, 3, 8, 2, 4 |
|---|---|---|
| **Step 1** | Arrange the numbers in ascending order. | 2 2 3 4 8 12 |
| **Step 2** | Delete the end numbers. | 2 3 4 8 |
| **Step 3** | Repeat Step 2 until only one or two numbers remain. | 3 4 |
| **Step 4** | The median is the number that remains, or the average of the two numbers that remain. | 3.5 |

It is especially important to think of mathematical procedures as sequences of precise instructions when you are programming a computer to solve a problem. Computer programs are algorithms written in a language which a computer can interpret.

Other types of everyday algorithm include cookery recipes, explanations on how to set up video recorders, and assembly instructions for flat-pack furniture. The following paragraph, from some instructions recently followed by the author, illustrates some of the advantages and disadvantages of algorithmic methods.

> From Bag 46 take one $\frac{3}{8}"\times 2\frac{1}{2}"$ Hex Head Bolt and one $\frac{3}{8}"$ Nyloc Nut. Insert the bolt through the bottom hole of the bracket on part 1050 and through the hole in part 1241. Attach the Nyloc Nut finger tight.

Providing the instructions are sufficiently precise, you can carefully work through an algorithm such as this one without needing to fully understand how everything fits together. Similarly, you can follow mathematical algorithms by rote, without understanding the process.

However, if you do understand a process then you can adapt the basic algorithm to special features of the problem, and thereby solve the problem more efficiently. Just as the author eventually stopped needing detailed instructions on how to attach parts together with appropriate-sized nuts and bolts, so you would not need to follow the algorithm slavishly if asked to find the median of $\overbrace{2, 2, 2, \dots, 2}^{\text{1000 numbers}}$.

An **algorithm** is a finite sequence of instructions for solving a problem. It enables a person or a computer to solve the problem without needing to understand the whole process.

You might wonder why the word ‘finite’ is necessary. The reason is that there are processes which are essentially infinite, like finding the sum of a series such as

$$1+\frac{1}{2}+\frac{1}{4}+\frac{1}{8}+\dots$$

by adding successive terms to the ‘sum so far’. This never ends, and is not an algorithm.

In this book you will learn some algorithms which have been developed to solve particular problems in Discrete Mathematics. You will need to know how to carry these algorithms out by hand, although most real-world applications involve so many steps that they require the use of a computer. You will also need to have some idea of why the methods work.

## 1.3 Sorting algorithms

Any collection of data, such as a telephone directory, is only of value if information can be found quickly when needed. Alongside the development of computer databases, many algorithms have been developed to speed up the modification, deletion, addition and retrieval of data. This section will consider just one aspect of this, the sorting of a list of numbers into numerical order.

There is no single ‘best’ algorithm for sorting. The size of the data set, and how muddled up it is initially, both affect which algorithm will sort the data most efficiently.

**Bubble Sort**
This algorithm is so called because the smaller numbers gradually rise up the list like bubbles in a glass of lemonade. The algorithm depends upon successive comparisons of pairs of numbers, as follows.

- Compare the 1st and 2nd numbers in the list, and swap them if the 2nd number is smaller.
- Compare the 2nd and 3rd numbers and swap if the 3rd is smaller.
- Continue in this way through the entire list.

Consider the application of this procedure, called a **pass**, to the list of numbers

5, 1, 2, 6, 9, 4, 3.

The numbers are first placed vertically in the left column. After each comparison the list is rewritten to the right. Fig. 1.1 shows one pass of Bubble Sort.

You can see that this pass has required 6 comparisons and 4 swaps.

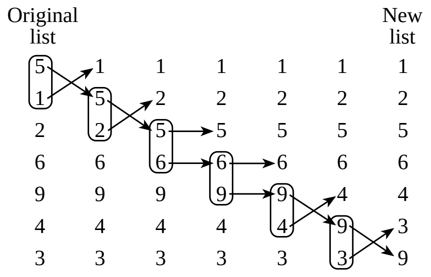


Fig. 1.1

The result of this pass through the list is that the numbers 1, 2, 4 and 3 (the bubbles) have each moved up one place. The other numbers have either stayed in place or moved down. In particular, you should be able to see that the largest number (in this case the 9) will always move to the bottom.

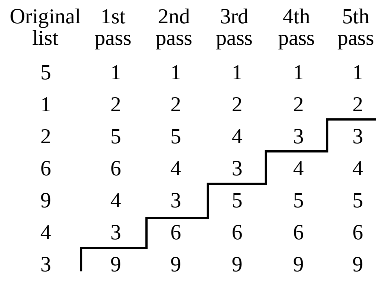
The complete Bubble Sort algorithm can be written as follows.

**Bubble Sort**

**Step 1** If there is only one number in the list then stop.

**Step 2** Make one pass down the list, comparing numbers in pairs and swapping as necessary.

**Step 3** If no swaps have occurred then stop. Otherwise, ignore the last element of the list and return to Step 1.



| Original list | 1st pass | 2nd pass | 3rd pass | 4th pass | 5th pass |
|---|---|---|---|---|---|
| 5 | 1 | 1 | 1 | 1 | 1 |
| 1 | 2 | 2 | 2 | 2 | 2 |
| 2 | 5 | 5 | 4 | 3 | 3 |
| 6 | 6 | 4 | 3 | 4 | 4 |
| 9 | 4 | 3 | 5 | 5 | 5 |
| 4 | 3 | 6 | 6 | 6 | 6 |
| 3 | 9 | 9 | 9 | 9 | 9 |

Fig. 1.2

Each pass alters the list of numbers as in Fig. 1.2, and the list ends up in order. The numbers under the ‘steps’ are the ones that are ignored.

Table 1.3 shows the numbers of swaps and comparisons which are required at each pass.

| | 1st pass | 2nd pass | 3rd pass | 4th pass | 5th pass | Totals |
|---|---|---|---|---|---|---|
| Comparisons | 6 | 5 | 4 | 3 | 2 | 20 |
| Swaps | 4 | 2 | 2 | 1 | 0 | 9 |

Table 1.3

One disadvantage of Bubble Sort is that once the data have been sorted, another complete pass through the data is necessary to ensure that the sorting has been finished. The next algorithm partially overcomes this problem.

**Shuttle Sort**

This algorithm is so called because numbers can move up more than one place in a pass.

**Shuttle Sort**

**1st pass** Compare the 1st and 2nd numbers in the list and swap if necessary.

**2nd pass** Compare the 2nd and 3rd numbers in the list and swap if necessary. If a swap has occurred, compare the 1st and 2nd numbers and swap if necessary.

**3rd pass** Compare the 3rd and 4th numbers in the list and swap if necessary. If a swap has occurred, compare the 2nd and 3rd numbers, and so on up the list.

And so on, through the entire list.

The results of successive passes of Shuttle Sort on the list

5, 1, 2, 6, 9, 4, 3

are shown in Fig. 1.4. The numbers above the stepped line are those which have been compared at each pass.

| Original list | 1st pass | 2nd pass | 3rd pass | 4th pass | 5th pass | 6th pass |
|---|---|---|---|---|---|---|
| 5 | 1 | 1 | 1 | 1 | 1 | 1 |
| 1 | 5 | 2 | 2 | 2 | 2 | 2 |
| 2 | 2 | 5 | 5 | 5 | 4 | 3 |
| 6 | 6 | 6 | 6 | 6 | 5 | 4 |
| 9 | 9 | 9 | 9 | 9 | 6 | 5 |
| 4 | 4 | 4 | 4 | 4 | 9 | 6 |
| 3 | 3 | 3 | 3 | 3 | 3 | 9 |

Fig. 1.4

Table 1.5 shows the numbers of swaps and comparisons which are required at each pass of Shuttle Sort.

| | 1st pass | 2nd pass | 3rd pass | 4th pass | 5th pass | 6th pass | Totals |
|---|---|---|---|---|---|---|---|
| Comparisons | 1 | 2 | 1 | 1 | 4 | 5 | 14 |
| Swaps | 1 | 1 | 0 | 0 | 3 | 4 | 9 |

Table 1.5

Shuttle Sort has involved the same number of swaps as Bubble Sort, but far fewer comparisons: $14$ as opposed to $20$.

## Exercise 1A

**1** (a) Apply Bubble Sort to the reverse-ordered list 5, 4, 3, 2, 1. Keep a count of the number of comparisons and swaps.

(b) Apply Shuttle Sort to the list 5, 4, 3, 2, 1. Again, keep a count of the number of comparisons and swaps.

(c) Compare the two sorting algorithms for lists in reverse order.

**2** (a) Apply Bubble Sort to a list of 6 numbers. What is the maximum possible number of comparisons and swaps that would need to be made for a list of 6 numbers?

(b) Generalise your answer to part (a) for a list of $n$ numbers.

**3** Apply Shuttle Sort to the list 4, 1, 6, 8, 2. Show the result of each pass and keep a count of the number of comparisons and swaps.

**4** Another sorting algorithm, called the Interchange algorithm, is defined as follows.

**Step 1** If there is only one number in the list then stop.
**Step 2** Find the smallest number in the list and interchange it with the first number.
**Step 3** Ignore the first element of the list and return to Step 1.

(a) Write down your own sub-algorithm to ‘find the smallest number in a list’. How many comparisons are needed when applying your algorithm to a list containing $n$ numbers?

(b) How many comparisons and swaps are needed when applying the Interchange algorithm to the list 5, 4, 3, 2, 1?

**5** Here are two algorithms. In each case, find the output if $m = 4$ and $n = 3$, and decide whether the algorithm would still work if either or both $m$ and $n$ were negative.

(a)
```
Step 1  Read the positive integers $m$ and $n$.
Step 2  Replace $m$ by $m-1$, and $n$ by $n+1$.
Step 3  If $m > 0$, go to Step 2. Otherwise write $n$.
```

(b)
```
Step 1  Read the positive integers $m$ and $n$.
Step 2  Let $p = 0$.
Step 3  Replace $m$ by $m-1$, and $p$ by $p+n$.
Step 4  If $m > 0$, go to Step 3. Otherwise write $p$.
```

## 1.4 Some important terms

You have already met the idea of finiteness; that is, an algorithm must stop after a finite number of steps. An even more fundamental idea is that of **correctness**, that is, the algorithm does actually do what it is intended to do!

**Example 1.4.1**
The following algorithm is designed to produce a sequence of $N$ prime numbers.

```
Step 1  Read $N$.
Step 2  Let $X = 1$.
Step 3  Let $P = X + 1$.
Step 4  Print $P$.
Step 5  Replace $X$ by $X \times P$.
Step 6  Replace $N$ by $N - 1$.
Step 7  If $N \neq 0$ go to Step 3. Otherwise stop.
```

By considering the case $N = 5$, comment on the correctness of this algorithm.

Following the instructions gives the results shown in the table.

| | | | | | | |
|---|---|---|---|---|---|---|
| $N$ | 5 | 4 | 3 | 2 | 1 | 0 |
| $X$ | 1 | 2 | 6 | 42 | 1806 | |
| $P$ | 2 | 3 | 7 | 43 | 1807 | |

The integers 2, 3, 7 and 43 are all prime. However, $1807 = 13 \times 139$, and is therefore not prime, so the algorithm is not correct.

Proving the correctness or otherwise of an algorithm can be extremely difficult. However, now that so much of international military and commercial affairs is dependent upon automated procedures carried out by computers, considerable effort has to be put into checking the correctness of these procedures and debugging (correcting) any which have flaws.

Working through the steps of an algorithm to check its correctness is sometimes called **tracing** the algorithm.

Another feature of algorithms illustrated by Example 1.4.1 is that of a stopping condition. As soon as $N = 0$ in Step 7, the procedure stops.

The algorithm in Example 1.4.2 has two stopping conditions. These occur in Step 2 and in Step 5.

**Example 1.4.2**
An algorithm is defined as follows.

```
Step 1   Read $X$.
Step 2   If $X < 0$ then stop.
Step 3   Let $A = 1$.
Step 4   Replace $A$ by $\frac{1}{2}\left(A + \frac{X}{A}\right)$.
Step 5   If $\left|X - A^2\right| < 0.001$, then stop.
Step 6   Go to Step 4.
```

(a) What is the purpose of this algorithm?
(b) What would happen if Step 2 were omitted?

(a) For any inputted number $X$, the algorithm is attempting to find a number $A$ such that $X - A^2$ is very small, that is, such that $A$ is an approximation to $\sqrt{X}$.

(b) $A^2$ can never be negative and so can never be very close to any value of $X$ which is negative. The additional stopping condition of Step 2 is therefore necessary to make sure the procedure stops when the number $X$ which is input is negative.

Example 1.4.2 also illustrates the idea of generality. Without Step 2, the algorithm in Example 1.4.2 would be correct for any positive value of $X$ but would not be correct 'in general'. An important aspect of any algorithm is whether or not it is correct for all possible inputs or whether it is only correct for a certain range of inputs.

## 1.5 The Shell Sort algorithm

When either the Bubble Sort algorithm or the Shuttle Sort algorithm is applied to a set of $n$ numbers, the number of comparisons can be as high as $1 + 2 + \ldots + (n-1) = \frac{1}{2}n(n-1)$. For small $n$, this number of comparisons can be made very quickly. However, for large $n$, an appreciably quicker method is the Shell Sort, named after D. L. Shell, who introduced this method in 1959. Shell's idea was to split a large set of numbers into smaller subsets, apply the Shuttle Sort algorithm to each subset and then gradually merge subsets together, using further applications of the Shuttle Sort algorithm as the subsets are merged.

To sort a list of $n$ numbers, $A(1), A(2), \ldots, A(n)$:

```
Shell Sort

Step 1   Let $X = n$.

Step 2   Replace $X$ by $\frac{1}{2}X$, ignoring any remainder.

Step 3   Apply the Shuttle Sort algorithm to each of the $X$ sublists:
             $A(1)$,      $A(X+1)$,    $A(2X+1)$,   …
             $A(2)$,      $A(X+2)$,    $A(2X+2)$,   …
             …            …            …            …
             …            …            …            …
             $A(X-1)$,    $A(2X-1)$,   $A(3X-1)$,   …
             $A(X)$,      $A(2X)$,     $A(3X)$,     …

Step 4   If $X > 1$ then go to Step 2.

Step 5   Stop.
```

This complicated-looking algorithm will be illustrated with an example and a commentary.

**Example 1.5.1**
(a) Apply the Shell Sort algorithm to the list of numbers which are initially in reverse order:

9, 8, 7, 6, 5, 4, 3, 2, 1, 0.

(b) Count the number of comparisons required for the list in part (a). How many comparisons would have been needed if the Shuttle Sort algorithm had initially been applied to the entire list?

(a) The first time Step 2 is performed, $X$, which was $10$, becomes $5$. So the 5 sublists to be sorted by Step 3 are:

$A(1)$, $A(6)$; $A(2)$, $A(7)$; $A(3)$, $A(8)$; $A(4)$, $A(9)$; $A(5)$, $A(10)$.

The working for Step 3 can then be laid out as shown below.

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 9 | | | | | 4 | | | | |
| | 8 | | | | | 3 | | | |
| | | 7 | | | | | 2 | | |
| | | | 6 | | | | | 1 | |
| | | | | 5 | | | | | 0 |

Sorting each pair then produces:

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 4 | 3 | 2 | 1 | 0 | 9 | 8 | 7 | 6 | 5 |

The second time Step 2 is performed, $X$ is made equal to $2$ because $\frac{5}{2} = 2.5$, which, when you ignore the remainder, becomes 2. So the sublists to be sorted are

$A(1)$, $A(3)$, $A(5)$, $A(7)$, $A(9)$ and $A(2)$, $A(4)$, $A(6)$, $A(8)$, $A(10)$,

where these numbers apply to the new list.

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 4 | 3 | 2 | 1 | 0 | 9 | 8 | 7 | 6 | 5 |
| 4 | | 2 | | 0 | | 8 | | 6 | |
| | 3 | | 1 | | 9 | | 7 | | 5 |

Sorting each subset of five elements then produces:

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

The third time Step 2 is performed, $X$ is made equal to $\frac{2}{2}=1$ and so the final pass of the Shuttle Sort algorithm takes place on the whole list, producing:

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

(b) The first time Step 3 was performed, the number of comparisons was 5, that is, 1 for each of the five sets.

The second time Step 3 was performed, the number of comparisons was:

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 4 | 2 | 0 | 0 | 0 | 3 | 1 | 1 | 1 | 1 |
| 2 | 4 | 2 | 2 | 2 | 1 | 3 | 3 | 3 | 3 |
| 0 | 0 | 4 | 4 | 4 | 9 | 9 | 9 | 7 | 5 |
| 8 | 8 | 8 | 8 | 6 | 7 | 7 | 7 | 9 | 7 |
| 6 | 6 | 6 | 6 | 8 | 5 | 5 | 5 | 5 | 9 |

Comparisons $1+2+1+2=6$ Comparisons $1+1+2+3=7$

The third and final time Step 3 was performed, the number of comparisons was 9. The total number was therefore $5+6+7+9=27$.

Applying the Shuttle Sort algorithm directly would have required $1+2+\ldots+9=45$ comparisons.

Note that the Shell Sort algorithm has produced a considerable saving in computing time even for a list with as few as 10 elements.

## 1.6 The Quicksort algorithm

You have seen that the Shell Sort algorithm is relatively fast compared with the Shuttle Sort algorithm for large values of $n$. Another sorting method which is similarly fast for large values of $n$ is called the Quicksort algorithm. Like the Shell Sort algorithm, the Quicksort algorithm is based upon the idea of splitting the original list into sublists. However, it operates in the opposite fashion by gradually dealing with smaller and smaller sublists.

**Quicksort**

**Step 1** Choose the number located in the middle of the list as the **pivot**, $P$. {If the list has an even number of elements then choose either of the middle two numbers.}

**Step 2** Go through the list putting any numbers less than $P$ to the left of $P$, and any greater than $P$ to the right. This creates two new sublists. {Note that the numbers in each sublist are kept in their original order.}

**Step 3** If each sublist has just one element, then stop. Otherwise go to Step 2 for each sublist.

Note that although Step 1 tells you to choose the middle number as the pivot, in fact any number in the list could be chosen as the pivot. Choosing the middle number is best when the original list is already roughly in order but, otherwise, it can be simpler just to choose the first number in each list as the pivot.

**Example 1.6.1**
Use the Quicksort algorithm to rearrange the following numbers into ascending order, showing the new arrangement at each stage. Take the middle number in any list as the pivot.

$$12 \quad 3 \quad 20 \quad 15 \quad 1 \quad 25 \quad 17$$

The first step is to choose the pivot. This is the middle member of the list, and it is put into a box, for convenience.

$$12 \quad 3 \quad 20 \quad \boxed{15} \quad 1 \quad 25 \quad 17$$

Putting the elements less than the pivot on the left, and the others on the right gives the following list.

$$12 \quad 3 \quad 1 \quad \boxed{15} \quad 20 \quad 25 \quad 17$$

Looking at the sublist to the left of the pivot, the new pivot is 3. Arranging the sublist so that the terms which are less than 3 are to the left of it, and the terms which are greater to the right of it gives this new list.

$$1 \quad \boxed{3} \quad 12 \quad \boxed{15} \quad 20 \quad 25 \quad 17$$

Applying the same procedure to the sublist which is to the right of the pivot, the new pivot is 25, and the new list is

$$1 \quad \boxed{3} \quad 12 \quad \boxed{15} \quad 20 \quad 17 \quad \boxed{25}$$

There is still a sublist of two numbers, 20, 17 which needs to be sorted, so applying Step 2 with 20 as pivot gives

$$1 \quad \boxed{3} \quad 12 \quad \boxed{15} \quad 17 \quad \boxed{20} \boxed{25}$$

Each sublist now has one element, so the algorithm stops.

In practice, for large values of $n$ and for randomly ordered initial lists, the Quicksort algorithm turns out to be the quickest of the algorithms you have met in this chapter – hence its name!

## Exercise 1B

**1** (a) Carry out the procedure of Example 1.4.1 in the case $N = \frac{1}{2}$. What happens?

(b) Modify the algorithm to overcome the problem you noted in part (a).

**2** Use the Shell Sort algorithm to rearrange the following numbers into ascending order. Show the new arrangement at each stage and count the number of comparisons required.

3, 1, 8, 7, 5, 9, 2, 6.

**3** (a) Use the Quicksort algorithm to rearrange the following colours into alphabetical order, showing the new arrangement at each stage. Take the first colour in any list as the pivot.

Pink, Black, Red, Green, Orange, White.

(b) Find the maximum number of comparisons that could be needed to sort a list of six words into alphabetical order. Take the first word in any list as the pivot.

(c) Find an expression for the maximum number of comparisons that could be needed to sort a list of $n$ words into alphabetical order. (AQA)

**4** (a) Use the Quicksort algorithm to rearrange the following numbers into order, showing the new arrangement at each stage. Take the first number in any list as the pivot.

9, 5, 7, 11, 2, 8, 6, 17.

(b) A Shuttle Sort algorithm is to be used to rearrange a list of numbers into order.

(i) Find the maximum number of comparisons that would be needed to be certain that a list of eight numbers was in order.

(ii) Find, in a simplified form, an expression for the maximum number of comparisons that would be needed to be certain that a list containing $n$ numbers was in order. (AQA)

**5** (a) Consider this algorithm which operates on $N$ items: $A(1), A(2), \dots, A(N)$.

```
For $I = 2$ to $N$
For $J = 0$ to $I - 2$
If $A(I - J) < A(I - J - 1)$ then exchange $A(I - J)$ and $A(I - J - 1)$
Next $J$
Next $I$
```

(i) Trace the algorithm in the case when $N = 4$ and the array contains

$A(1) = 47$, $A(2) = 69$, $A(3) = 8$, $A(4) = 52$.

(ii) Write down the name of this algorithm.

(b) Use the Shell Sort algorithm to sort the list {4, 10, 3, 7, 2, 1, 8, 11, 7, 12}, indicating where it uses the algorithm specified in part (a), the output of which may be written down directly each time it is used. (AQA, adapted)

## 1.7 Flow diagrams

A flow diagram is a pictorial representation of an algorithm. Differently shaped boxes are used for different types of instruction. Fig. 1.6 shows you which instructions go into which boxes.

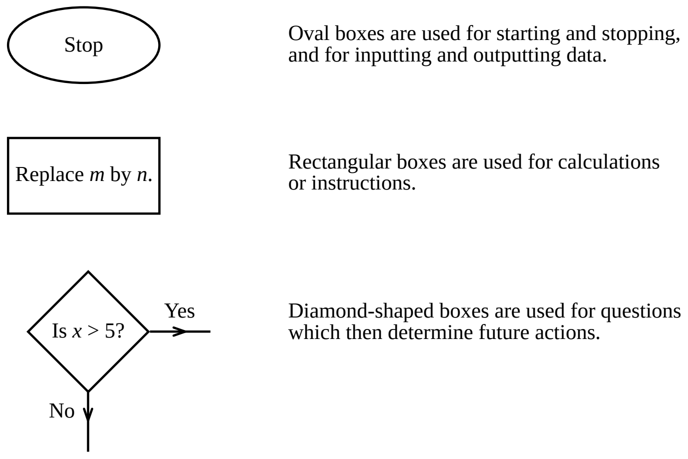

Stop
Oval boxes are used for starting and stopping, and for inputting and outputting data.
Replace *m* by *n*.
Rectangular boxes are used for calculations or instructions.
Is *x* > 5?
Yes
No
Diamond-shaped boxes are used for questions which then determine future actions.

Fig. 1.6

The algorithm given in Section 1.2 for finding the median of a set of numbers can be represented by the flow diagram in Fig. 1.7.

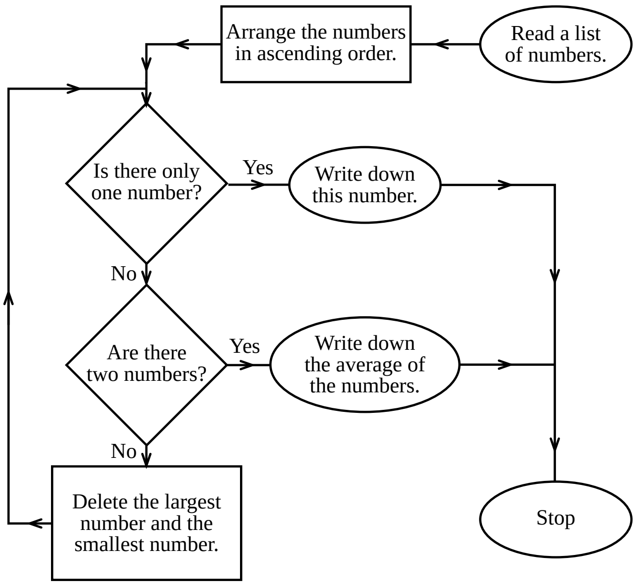

Read a list of numbers.
Arrange the numbers in ascending order.
Is there only one number?
Yes
Write down this number.
No
Are there two numbers?
Yes
Write down the average of the numbers.
No
Delete the largest number and the smallest number.
Stop

Fig. 1.7

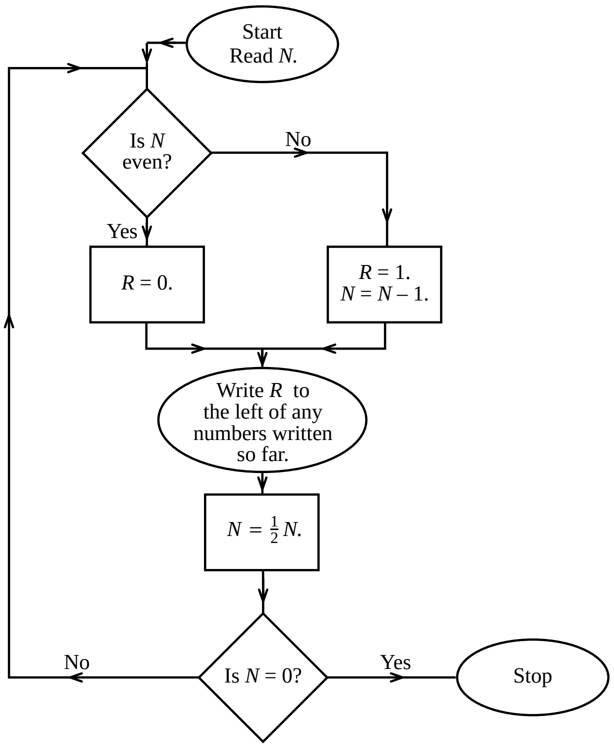
## 1.8 Notation for algorithms

In Fig. 1.6, the rectangle contained the instruction ‘Replace $m$ by $n$’. Think of $m$ and $n$ as labels of pigeon-holes, each of them containing a number. This instruction means take the number which is in pigeon-hole $n$ and put it into pigeon-hole $m$, replacing the number which is already there. The notation used in this book for this instruction will be $m = n$.

Similarly, $m = 2$ means put the number 2 into pigeon-hole $m$; and $m = m - 1$ means take the number already in pigeon-hole $m$, subtract 1 from it, and put the result back into pigeon-hole $m$.

These pigeon-holes are usually called **stores**.

**Example 1.8.1**
An algorithm has a flow diagram which is shown below.
(a) What is the output if $N = 57$?
(b) What has this algorithm been designed to do?



(a) After successive passes around the flow diagram, the values of $N$, $R$ and the numbers written down so far are as shown in the table.

| Pass | $N$ | $R$ | Written down |
|---|---|---|---|
| 1 | 28 | 1 | 1 |
| 2 | 14 | 0 | 01 |
| 3 | 7 | 0 | 001 |
| 4 | 3 | 1 | 1001 |
| 5 | 1 | 1 | 11001 |
| 6 | 0 | 1 | 111001 |

(b) The algorithm converts $N$ into a binary number. So, for example,

$$57 = (1\times 32) + (1\times 16) + (1\times 8) + (0\times 4) + (0\times 2) + (1\times 1) = 111001_2.$$

Sometimes when you see algorithms, you will see explanatory comments written about some or all of the steps. These will usually be put into curly brackets, {}, often called braces.

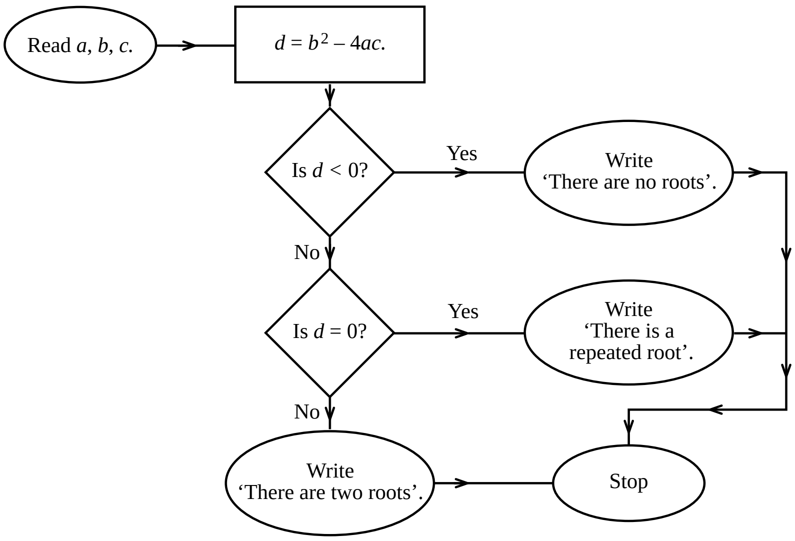
Here is an example of an algorithm with comments. It produces a permutation of the numbers 1 to $n$, that is, it produces the numbers 1 to $n$ in random order, with no repeats. Notice how the comments make the algorithm easier to understand.

```
Step 1  Read $n$.                     {$n$ is the number of numbers in the permutation.}
Step 2  $i = 1$.                      {$i$ is the number of the permutation currently being found.}
Step 3  $r = Rand(1, n)$              {$Rand(1, n)$ is a random integer
                                       between 1 and $n$ inclusive.}
Step 4  If $r$ has not already been used, write $r$, otherwise go to Step 3.
                                      {$r$ is the $i$th number in the permutation.}
Step 5  $i = i + 1$.                  {To get the next number in the permutation.}
Step 6  If $i < n$ go to Step 3, otherwise stop.
```

This algorithm is not very efficient, because you can spend much time regenerating random numbers which you have already found. You will find a much better algorithm in Miscellaneous exercise 1 Question 10.

## Miscellaneous exercise 1

**1** A flow diagram is shown below.


Read $a, b, c.$
$d = b^2 - 4ac.$
Is $d < 0$?
Yes
Write 'There are no roots'.
No
Is $d = 0$?
Yes
Write 'There is a repeated root'.
No
Write 'There are two roots'.
Stop

(a) What is the output of this algorithm in the cases

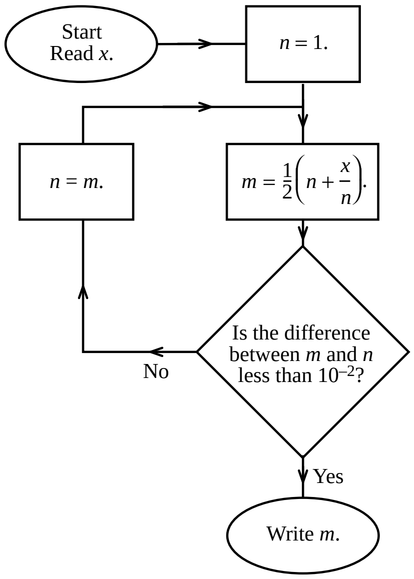
(i) $a = 1,\ b = 3,\ c = 2$; (ii) $a = 1,\ b = 2,\ c = 1$; (iii) $a = 1,\ b = 2,\ c = 3$?

(b) What has this algorithm been designed to do?

(c) How could you adapt the flow diagram to represent an algorithm to find the roots of a quadratic equation?

**2** (a) Carry out the algorithm in this flow diagram for $x = 2$, $x = 3$ and $x = 5$.

(b) For what is this algorithm designed?


Start
Read $x$.
$n = 1$.
$n = m$.
$m = \frac{1}{2}\left(n + \frac{x}{n}\right)$.
Is the difference between $m$ and $n$ less than $10^{-2}$?
No
Yes
Write $m$.

**3** Euclid's algorithm is defined as follows.

**Step 1** Read $X$ and $Y$.

**Step 2** If $X = Y$ then print $X$ and stop.

**Step 3** Replace the larger of $X$ and $Y$ by the difference between $X$ and $Y$.

**Step 4** Go back to Step 2.

(a) Carry out Euclid's algorithm for inputs of

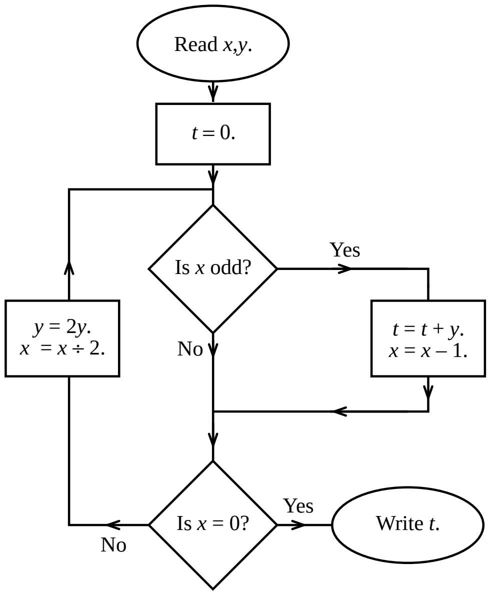
(i) 6 and 15, (ii) 4 and 18, (iii) 3 and 7.

(b) What does Euclid's algorithm find?

(c) Draw a flow diagram for this algorithm.

**4** The Russian peasant's algorithm is defined as shown in the diagram.

Complete a table showing the successive values of $x$, $y$ and $t$, when $x$ and $y$ initially take the values $11$ and $9$. Hence decide what the algorithm is designed to do.



**5** An algorithm to sort a list, $L$, of numbers into increasing order is defined as follows.

**Step 1** Write down the first number of $L$ as the start of a new list $N$.

**Step 2** Take the next number of $L$. Compare it to each number of $N$ in turn (from the left) and insert it at the appropriate place in $N$.

**Step 3** Repeat Step 2 until all numbers of $L$ have been placed in $N$.

*Note that for each number from $L$, once you have found a number in $N$ that is bigger than it, you do not need to make any further comparisons.*

(a) Use this algorithm to sort $L = \{8, 12, 2, 54, 23, 31\}$. Write down the list $N$ and count the number of comparisons at each stage.

(b) Order the list $L$ in such a way that the algorithm will require the largest possible number of comparisons.

(c) What is the largest possible number of comparisons needed for this algorithm to sort a list of $n$ numbers?

**6** The following algorithm is based on a method used by Archimedes to estimate $\pi$.

**Step 1** $C = \frac{1}{2}\sqrt{3}$, $S = 3$, $T = 2\sqrt{3}$, $D = 2\sqrt{3} - 3$.

**Step 2** Repeat

$$C = \sqrt{\tfrac{1}{2}(1 + C)}.$$

$$S = \frac{S}{C}.$$

$$T = \frac{S}{C}.$$

$$D = T - S.$$

Until $D < 0.01$.

**Step 3** Print $S$.

(a) Copy and complete the following table to show the values of $C$, $S$, $T$ and $D$ (rounded to 4 decimal places) for the first three runs through the repeat loop.

| | $C$ | $S$ | $T$ | $D$ |
|---|---|---|---|---|
| Initial values | 0.8660 | 3.0000 | 3.4641 | 0.4641 |
| 1st iteration | 0.9659 | 3.1058 | 3.2154 | 0.1096 |
| 2nd iteration | | | | |
| 3rd iteration | | | | |

(b) Write down the number of runs through the repeat loop that are needed if the value 0.01 (in line 7 of the algorithm) is replaced by 0.1.

(c) Let $d_n$ be the value of $D$ from the $n$th iteration. Use the values of $d_1$, $d_2$ and $d_3$ to conjecture an approximate value for $\frac{d_{n+1}}{d_n}$.

(d) Hence suggest an approximate expression for the value of $D$ from the $n$th iteration.

(OCR, adapted)

**7** The following algorithm has been written to input a set of 30 examination marks, each expressed as an integer percentage, find the minimum mark and output the result.

<u>Line</u>

```
1   SET MIN = 100
2   FOR $I$ = l to 30
3   INPUT MARK
4   IF MARK < MIN
5   THEN MIN = MARK
6   NEXT N
7   ........................
```

(a) (i) State the purpose of Line 1 of the algorithm.

(ii) There is a mistake in Line 6. Write down a corrected version of this line.

(iii) The contents of Line 7 are missing. Write down the contents of Line 7 to ensure that the algorithm is fully complete.

(b) Show how this algorithm could be adapted if the number of examination marks to be input was unknown.

(c) Write an algorithm that would input a set of 50 examination marks, each expressed as an integer percentage, find the maximum mark and output the result. (AQA)

**8** A student is writing a computer program to calculate part of a multiplication table. This is the algorithm she uses.

```
$X = 0, K = 0$
FOR $I = 1$ TO 6
FOR $J = 1$ TO 5
$X = I \times J$
$K = K + 1$
PRINT $K$, $I$, “times”, J, “equals”, $X$
NEXT $J$
NEXT $I$
END
```

(a) State

(i) the purpose of the line $X = 0, K = 0$,

(ii) why the lines NEXT $J$, NEXT $I$ are given in that order,

(iii) the purpose of the variable $K$.

(b) When the value of $K$ printed is 8, find the corresponding printed values of $I$, $J$ and $X$.

(AQA)

**9** The Binary Search algorithm described below locates the position of a certain value, $X$, within an ordered sequence $S(1), S(2), \dots, S(N)$.

$\text{INT}(0.5(I+J))$ gives the largest integer that is less than or equal to $0.5(I+J)$.

```
Step 1  $I = 1$, $J = N$.     {$I$ and $J$ mark the boundaries of the sequence being searched.}
Step 2  If $I > J$ then print 'FAIL' and stop.     {$X$ has not been found.}
Step 3  $M = \text{INT}(0.5(I+J))$.
Step 4  If $X = S(M)$ then print $M$ and stop.     {$X$ has been found.}
Step 5  If $X < S(M)$ then $J = M - 1$, otherwise $I = M + 1$.     {Reset boundaries.}
Step 6  Go to Step 2.
```

(a) Demonstrate carefully each step of the algorithm when it is applied to the sequence

1 1 2 3 5 8 13 21

(i) with $X = 13$;

(ii) with $X = 15$.

If $N$ is a power of 2, the length of the sequence to be searched is at least halved at each iteration.

(b) If $N = 10$, work out the maximum possible length of the sequence to be searched at each iteration.

**10*** This algorithm produces a random permutation of the numbers from 1 to $n$; that is, it produces the numbers from 1 to $n$ in random order, with no repeats.

```
Step 1  Read $n$.     {$n$ is the number of numbers in the permutation.}
Step 2  For $i = 1$ to $n$, $Perm(i) = i$.     {$Perm(i)$ will be the $i$th number in the
                                                permutation. This is an initialisation step.}
Step 3  $j = 1$.
Step 4  Repeat the following.
            $r = Rand(j, n)$.     {$Rand(j, n)$ is a random number
                                   from $j$ to $n$ inclusive.}
            Swap $Perm(j)$ with $Perm(r)$,
            $j = j + 1$.
        Until $j = n - 1$.
Step 5  Write $Perm(1), Perm(2), \dots, Perm(n)$.
```

(a) Work through the algorithm with $n = 5$, in the case when the random numbers produced are successively, 3, 4, 3 and 5. What permutation do you finish with?

(b) How many times do you have to use the random number generator when there are $n$ numbers?

(c) Compare this with the algorithm on page 14.