

1

Data Compression

A Level Only

Photocopiable digital resources may only be copied by the purchasing institution on a single site and for their own use

INSPECTION COPY



2

Compression

Compression is the process of reducing the size of a file.

It is particularly beneficial to compress multimedia files such as images, videos and audio as they typically use a lot of storage space.

Multimedia files are often compressed so they can be sent over the Internet.

Other file types, such as text, can also be compressed.

INSPECTION COPY



3

Types of Compression

There are two types of compression:

Lossless	Lossy
The data is compressed without permanently removing any data.	The data is compressed by permanently removing some of the data.

INSPECTION COPY



4

Lossy Compression

An example of lossy compression is the MP3 audio file format.

It removes parts of the sound that are out of the range of normal human hearing.

This means that people shouldn't be able to notice any difference in the sound.

INSPECTION COPY



5

Comparison

	Advantages	Disadvantages
Lossless	All data is retained.	Can be slow and creates larger files.
Lossy	Usually results in smaller file sizes when compared to lossless compression.	Permanently removes some data.

INSPECTION COPY



6

Run-Length Encoding

Run-length encoding (RLE) is one of the simplest forms of lossless compression.

It works by identifying sequences of the same character and replacing them with a count followed by the character. For example, the sequence AAAAAA could be represented as 6A.

1100011100
2(1), 3(0), 3(1), 2(0),

INSPECTION COPY



7

Dictionary-Based

Dictionary-based methods work by identifying repeated sequences of data.

Each sequence is represented by a unique code.

1011 0011 0011
0011 1111 1011

There are only three different sequences in the data represented using the codes.

1011 = 00 0011 = 01 1111 = 11

The compressed data would then be:

00 01 01 11 01 11

INSPECTION COPY

COPYRIGHT
PROTECTED

1

Encryption

A Level
Only

Photos of this digital resource may only be copied by the purchasing institution on a single site and for their own use.

INSPECTION COPY

COPYRIGHT
PROTECTED

2

Encryption

Encryption is the process of changing a message into a code without a special key.

Key Terms:

Cipher	A method of encoding a message.
Plaintext	The message before it is encrypted.
Ciphertext	The message after it has been encrypted.

INSPECTION COPY

COPYRIGHT
PROTECTED

3

Caesar Cipher

Used by Julius Caesar, this is one of the earliest forms of encryption.

It works by offsetting the letters by a certain number. For example, with a shift of 3,

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S			

Plaintext:	COMP
Ciphertext:	FRPS

INSPECTION COPY

COPYRIGHT
PROTECTED

4

Symmetric

The symmetric encryption method uses the same key to encrypt and decrypt the message.

Symmetric encryption is relatively insecure as the key can be easily intercepted.



INSPECTION COPY

COPYRIGHT
PROTECTED

5

Asymmetric

The asymmetric encryption method uses different keys to encrypt and decrypt.

Public Key The public key can only be used to encrypt a message.

Private Key The private key is used to decrypt a message.

If someone wants to send you some information, they will share their public key with you which they can use to encrypt the message.

You can then decrypt the message using your private key as you don't share so only you can decrypt it.

INSPECTION COPY

COPYRIGHT
PROTECTED

1

Relational Database

A Level Only

Photocopiable/digital resources may only be copied by the purchasing institution on a single site and for their own use

INSPECTION COPY

COPYRIGHT PROTECTED

Zig Zag Education

2

Database

Databases are used to store data in a structured way. Here is the basic structure of a database.

Primary Key
A field that uniquely identifies each record. Ensures all records are unique.

Table
Data is stored in tables.

student_id	First Name
001	Lois
002	Mel
003	Eloise
004	Nicola

One or more

Sometimes two fields are used together to uniquely identify a record. This is known as a composite key.

INSPECTION COPY

COPYRIGHT PROTECTED

Zig Zag Education

3

Relational Database

The most commonly used database is a relational database.

It allows data to be organised in a way that makes it easy to process.

In a relational database, data is separated into tables. Each one stores data relating to a single entity.

An entity is something in the real world that is stored in a database; for example, a product.

INSPECTION COPY

COPYRIGHT PROTECTED

Zig Zag Education

4

Foreign Key

A foreign key is a primary key from another table used to link the two tables together.

Foreign Key

Primary Key

Student Table

student_id	First Name	Teacher	teacher_id
001	Alex	Bennett	002
002	James	Hadwen	001
003	Eloise	Roberts	002
004	Patrick	Dua-Brown	003

In this example the **teacher_id** field is used to link students to teachers.

INSPECTION COPY

COPYRIGHT PROTECTED

Zig Zag Education

5

Entity Relationship

We define the relationship between tables using entity relationship diagrams.

These are the most common types of relationship:

Relationship Type	Symbol	Example
One-to-one	—	Each student has one teacher.
One-to-many	—>	Each teacher has many students.
Many-to-many	>—>	Each teacher has many students and each student has many teachers.

INSPECTION COPY

COPYRIGHT PROTECTED

Zig Zag Education

6

Examples

Each student can only have one form group but each form group can have multiple students.

Form

Each teacher has one form and each form has multiple teachers.

Teacher

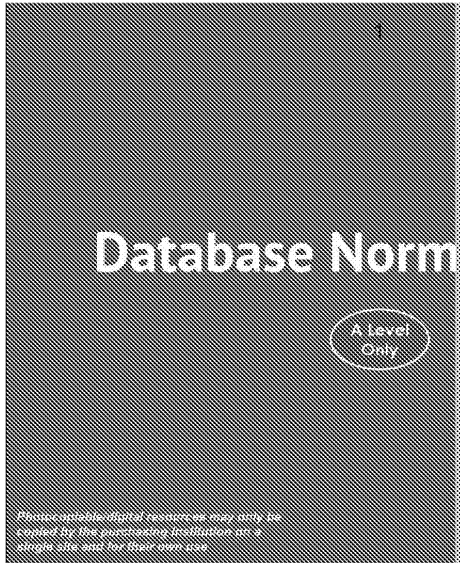
Each teacher has multiple students and each student has multiple teachers.

Teacher

INSPECTION COPY

COPYRIGHT PROTECTED

Zig Zag Education



INSPECTION COPY



2 Normalisation

Normalisation is the process of structuring data to ensure that it is stored efficiently.

Key Terms

Data Redundancy	This is when fields are repeated in different records.
Atomic Level	Data stored at an atomic level is not decomposed further; for example, into the fields Title , First Name , Second Name and Third Name .
Normal Form	A database that has been designed to meet the requirements of the normalisation process.



3 First Normal Form

There are different levels of normalisation, each with its own requirements.

Requirements

There is no repeated data and only one value per field.

The data is atomic.

Each record is unique.

INSPECTION COPY



4 1NF Example

Here is an example of repeated data. There are two records that contain email addresses (the same email address).

StudentID	FirstName	LastName	Email
1	John	Curtis	jcurtis@hotmail.com
2	Ben	Jackson	bjackson@me.com

INSPECTION COPY



5 Second Normal Form

Requirements

Meets the requirements of 1NF.

All non-key attributes should depend on all key attributes. This can be achieved by creating a primary key.

INSPECTION COPY



6 2NF Example

This **Student** table contains data about each student. The **StudentID** and **TeacherName** fields don't depend on the other fields.

StudentID	FirstName	LastName	TeacherName
001	Alex	Bennett	
002	James	Hadwen	
003	Eloise	Roberts	
004	Patrick	Dua-Brown	

INSPECTION COPY



2NF Examp

This **Student** table contains data about each student. The *StudentID* and *TeacherName* fields don't depend on the

They should therefore be in a separate

StudentID	FirstName	LastName
001	Alex	Bennett
002	James	Hadwen
003	Eloise	Roberts
004	Patrick	Dua-Brown

TeacherID	TeacherTitle	TeacherName
001	Mr	Ford
002	Mr	Smith
003	Mrs	Patel

COPYRIGHT
PROTECTED



8

Third Normal Form

Requirements
Meets the requirements for
Non-key attributes should not depend on

COPYRIGHT
PROTECTED



9

3NF Examp

The *FormRoom* field in the **Teacher** table d

StudentID	Firstname	LastName	Age
001	Alex	Bennett	15
002	James	Hadwen	16
003	Eloise	Roberts	15
004	Patrick	Dua-Brown	16

TeacherID	TeacherTitle	TeacherName	Form
001	Mr	Ford	7SFD
002	Mr	Smith	7ASM
003	Mrs	Patel	7APT

COPYRIGHT
PROTECTED



10

3NF Examp

The *FormRoom* field in the **Teacher** table d

To solve this we can create a separate

StudentID	FirstName	LastName	Year
001	Alex	Bennett	
002	James	Hadwen	
003	Eloise	Roberts	
004	Patrick	Dua-Brown	

TeacherID	TeacherTitle	TeacherName	Form
001	Mr	Ford	7SFD
002	Mr	Smith	7ASM
003	Mrs	Patel	7APT

COPYRIGHT
PROTECTED



4

Structured Query Language (SQL)

A Love
Only

Photocopyable digital resources may only be copied by the purchasing institution on a single site and for their own use.

COPYRIGHT
PROTECTED



2

Structured Query I

Structured query language (SQL) is the language used to interact with databases. Here is an example of a SQL query:

```
SELECT FirstName FROM Student WHERE
```

 Fields to show

The table

This statement selects the **FirstName** field from the **Students** table, but only shows the male students. Consider the following statement:

StudentID	FirstName	LastName	Age
1	John	Curtis	12
2	Ben	Jackson	1
3	Sarah	Smith	66

COPYRIGHT
PROTECTED



3 Results

StudentID	FirstName	LastName	Age
1	John	Curtis	12
2	Ben	Jackson	1
3	Sarah	Smith	68

SELECT FirstName FROM Student WHERE

Fields to show

The table

These are the results of the query

FirstName
John
Ben

INSPECTION COPY



4 Comparison Operators

The following comparison operators are used in SQL

Comparison operator
Equal to
Less than
Greater than
Less than or equal to
Greater than or equal to
Not equal to

INSPECTION COPY



5 Combining Conditions

StudentID	FirstName	LastName	Age
1	John	Curtis	55
2	Ben	Jackson	75
3	Sarah	Smith	80

We can use the AND operator to combine conditions

SELECT FirstName FROM Student WHERE Gender = "Male" AND Age > 50

These are the results of the query

TestScore
Ben

INSPECTION COPY



6 Inserting a Record

A new record can be added to a table using the INSERT statement

INSERT INTO Student (StudentID, FirstName, LastName, Age) VALUES (4, "Eloise", "Roberts", "9")

StudentID	FirstName	LastName	Age
1	John	Curtis	12
2	Ben	Jackson	1
3	Sarah	Smith	68
4	Eloise	Roberts	9

INSPECTION COPY



7 Deleting a Record

Records can be deleted from a table using the DELETE statement

DELETE FROM Student WHERE StudentID = 4

StudentID	FirstName	LastName	Age
1	John	Curtis	12
2	Ben	Jackson	1
4	Eloise	Roberts	9

INSPECTION COPY



8 Updating a Record

Records can be updated using the UPDATE statement

UPDATE Student SET Address = "45 Cleve" WHERE StudentID = 4

StudentID	FirstName	LastName	Age
1	John	Curtis	12
2	Ben	Jackson	4
4	Eloise	Roberts	9

INSPECTION COPY



Multiple Tables

Data can be retrieved from multiple tables

```
SELECT FirstName, LastName, T
FROM Student, Teacher
WHERE Student.TeacherID = T
```

StudentID	FirstName	LastName	TeacherID
001	Alex	Bennett	001
002	James	Hadwen	001
003	Eloise	Roberts	002
004	Patrick	Dua-Brown	003

A linking condition is used to join the

Dot notation is used to specify which

INSPECTION COPY

COPYRIGHT
PROTECTED



Sorting Results

The results of a query can be sorted using

```
SELECT FirstName FROM Student WHERE
ORDER BY LastName
```

This will order the results by the students
order.

INSPECTION COPY

COPYRIGHT
PROTECTED



Creating a Table

A new table can be created using the CREATE

```
CREATE TABLE Student (
StudentID INT PRIMARY KEY NOT NULL
FirstName VARCHAR(20)
LastName VARCHAR(20)
Gender VARCHAR(1)
)
```

NOT NULL means the field can't

VARCHAR is a string with a variable length
the value in the brackets

INSPECTION COPY

COPYRIGHT
PROTECTED



Binary and Hex

Please explain that digital resources may only be copied by the purchasing institution on a single site and for their own use.

INSPECTION COPY

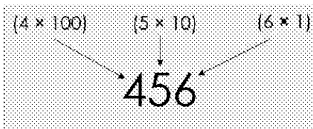
COPYRIGHT
PROTECTED



Base 10

Normally numbers are written in base

Each digit is worth 10 times more than the



456 is
lots

(4 ×

INSPECTION COPY

COPYRIGHT
PROTECTED

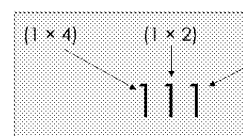


Binary

Computers work with a different

Computers use binary which only has two

In binary each digit is worth twice as much as



INSPECTION COPY

COPYRIGHT
PROTECTED



4 Binary Exam

Converting from binary (base 2) to decimal (

16s	8s	4s	2s	1s	
1	0	0	1	1	

Write out the value of each digit at t

INSPECTION COPY



5 Binary Exam

Converting from binary (base 2) to decimal (

16s	8s	4s	2s	1s	
1	0	0	1	1	(1x16)

Write out the value of each digit at t

Add together the value of the colu

INSPECTION COPY



6 Binary Exam

Converting from binary (base 2) to decimal (

16s	8s	4s	2s	1s	
1	0	0	1	1	(1x16)
0	1	1	1	0	(1x8)

Write out the value of each digit at t

Add together the value of the colu

INSPECTION COPY



7 Denoting Ba

As the digits look the same, subscripts are us
number has been writt

1111₁₀

The subscript 10 states this is a
decimal number.

The

INSPECTION COPY



8 Hexadecim

Numbers in binary can get r

Hexadecimal (base 16) shortens the numbe
worth 16 times the one to th

$$10_{16} = 16_{10} \quad 100_{16} = 256_{10}$$

Along with 0–9, hexadecimal also u

A = 10, B = 11, C = 12, D = 13, E

$$A0_{16} = 160_{10} \quad B00_{16} = 2816_{10}$$

INSPECTION COPY



9 Hexadecimal Ex

Converting from hex to denary is als

256s	16s	1s	
1	2	3	
A	B	C	

Write out the value of each digit at t

INSPECTION COPY



10

Hexadecimal Ex

Converting from hex to denary is als

256s	16s	1s	Working
1	2	3	$(1 \times 256) + (2 \times 16) + (3 \times 1)$
A	B	C	$(10 \times 256) + (11 \times 16) + (12 \times 1)$

Write out the value of each digit at t

Multiply the value of the column by the

INSPECTION COPY

COPYRIGHT
PROTECTED

11

Hexadecimal Ex

Converting from hex to denary is als

256s	16s	1s	Working
1	2	3	$(1 \times 256) + (2 \times 16) + (3 \times 1)$
A	B	C	$(10 \times 256) + (11 \times 16) + (12 \times 1)$

Write out the value of each digit at t

Multiply the value of the column by the

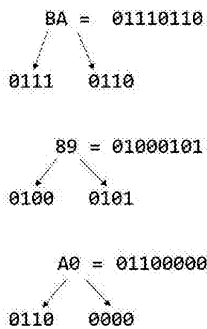
Add together the totals to get the c

INSPECTION COPY

COPYRIGHT
PROTECTED

12

Hexadecimal to

Although comp
hex is used

Each hex digit h

In binary, this is

To convert a tw
converting eOnce each in
converted, join t
together t

INSPECTION COPY

COPYRIGHT
PROTECTED

13

Binary to Hexad

Binary to hex is also a straightfo

Group the binary number into batc
(starting at the right-ha

01100101

=

0110	0101
6	5

Convert each batch of four digits to a

Then just join the hex digits b

INSPECTION COPY

COPYRIGHT
PROTECTED

14

Denary to Bi

Converting from denary to binary uses subtr

Example: 230 in decimal t

Start with the number line showing th

128	64	32	16	8

INSPECTION COPY

COPYRIGHT
PROTECTED

15

Denary to Bi

Converting from denary to binary uses subtr

Example: 230 in decimal t

Start with the number line showing th

128	64	32	16	8
1				

Locate the largest number you'll need and

INSPECTION COPY

COPYRIGHT
PROTECTED

Denary to Binary

Converting from denary to binary uses subtraction.

Example: 230 in decimal to binary

Start with the number line showing the powers of 2.

128	64	32	16	8
1				

$$230 - 128 = 102$$

Locate the largest number you'll need and subtract it from the denary you're converting.

Subtract that number from the denary you're converting and move to the next number on the number line.

INSPECTION COPY

COPYRIGHT
PROTECTED



Denary to Binary

Converting from denary to binary uses subtraction.

Example: 230 in decimal to binary

Start with the number line showing the powers of 2.

128	64	32	16	8
1	1	1	0	0

$$230 - 128 = 102$$

$$102 - 64 = 38$$

$$38 - 32 = 6$$

Locate the largest number you'll need and subtract it from the denary you're converting.

Subtract that number from the denary you're converting and move to the next number on the number line.

Repeat the process for each column, putting a 1 if it is used and a 0 if it is not.

INSPECTION COPY

COPYRIGHT
PROTECTED



Decimal to Hexadecimal

Example: 2468

$$2400 \div 16 = 154 \text{ remainder } 8$$

$$154 \div 16 = 9 \text{ remainder } 10$$

$$9 \div 16 = 0 \text{ remainder } 9$$

2468 in denary is 9A8 in hex

Divide the denary number by 16.

Write down the quotient and the remainder.

Repeat the process with the remainder until you reach 0.

Keep going until the remainder is 0.

The final remainder is the hex number. The penultimate remainder is the next hex digit.

Remember you can always convert to binary first and then to hexadecimal if you find it easier.

INSPECTION COPY

COPYRIGHT
PROTECTED



Binary Arithmetic

Proteasoft Ltd. All rights reserved. This resource may only be copied by the purchasing institution on a single site and for their own use.

INSPECTION COPY

COPYRIGHT
PROTECTED



Binary Addition

The process of performing addition in binary is similar to the process in decimal. There are four simple rules to follow.

Rules:

$$0 + 0 = 0$$

$$0 + 1 = 1$$

$$1 + 1 = 0 \text{ carry } 1$$

$$1 + 1 + 1 = 1 \text{ carry } 1$$

INSPECTION COPY

COPYRIGHT
PROTECTED



Binary Addition

The process of performing addition in binary is similar to the process in decimal. There are four simple rules to follow.

Rules:

$$0 + 0 = 0$$

$$0 + 1 = 1$$

$$1 + 1 = 0 \text{ carry } 1$$

$$1 + 1 + 1 = 1 \text{ carry } 1$$

	1				
0	1	1	0	1	
0	0	1	0	1	
					0

INSPECTION COPY

COPYRIGHT
PROTECTED



4

Binary Addition

The process of performing addition in binary is similar to the process of performing addition in decimal. There are four simple rules to follow:

Rules:

$$0 + 0 = 0$$

$$0 + 1 = 1$$

$$1 + 1 = 0 \text{ carry } 1$$

$$1 + 1 + 1 = 1 \text{ carry } 1$$

$$\begin{array}{r} 1 1 1 \\ 0 1 1 1 \\ \hline 1 1 0 0 \end{array}$$

INSPECTION COPY

COPYRIGHT
PROTECTED

5

Binary Addition

The process of performing addition in binary is similar to the process of performing addition in decimal. There are four simple rules to follow:

Rules:

$$0 + 0 = 0$$

$$0 + 1 = 1$$

$$1 + 1 = 0 \text{ carry } 1$$

$$1 + 1 + 1 = 1 \text{ carry } 1$$

$$\begin{array}{r} 1 1 1 \\ 0 1 1 1 \\ \hline 0 1 0 1 \end{array}$$

INSPECTION COPY

COPYRIGHT
PROTECTED

6

Binary Addition

The process of performing addition in binary is similar to the process of performing addition in decimal. There are four simple rules to follow:

Rules:

$$0 + 0 = 0$$

$$0 + 1 = 1$$

$$1 + 1 = 0 \text{ carry } 1$$

$$1 + 1 + 1 = 1 \text{ carry } 1$$

$$\begin{array}{r} 1 1 1 \\ 0 1 1 1 \\ \hline 0 0 1 0 \end{array}$$

INSPECTION COPY

COPYRIGHT
PROTECTED

7

Binary Addition

The process of performing addition in binary is similar to the process of performing addition in decimal. There are four simple rules to follow:

Rules:

$$0 + 0 = 0$$

$$0 + 1 = 1$$

$$1 + 1 = 0 \text{ carry } 1$$

$$1 + 1 + 1 = 1 \text{ carry } 1$$

$$\begin{array}{r} 1 1 1 \\ 0 1 1 1 \\ \hline 1 0 1 0 \end{array}$$

INSPECTION COPY

COPYRIGHT
PROTECTED

8

Binary Addition

The process of performing addition in binary is similar to the process of performing addition in decimal. There are four simple rules to follow:

Rules:

$$0 + 0 = 0$$

$$0 + 1 = 1$$

$$1 + 1 = 0 \text{ carry } 1$$

$$1 + 1 + 1 = 1 \text{ carry } 1$$

$$\begin{array}{r} 1 1 1 \\ 0 1 1 1 \\ \hline 1 0 1 0 \end{array}$$

INSPECTION COPY

COPYRIGHT
PROTECTED

9

Binary Addition

The process of performing addition in binary is similar to the process of performing addition in decimal. There are four simple rules to follow:

Rules:

$$0 + 0 = 0$$

$$0 + 1 = 1$$

$$1 + 1 = 0 \text{ carry } 1$$

$$1 + 1 + 1 = 1 \text{ carry } 1$$

$$\begin{array}{r} 1 1 1 \\ 0 1 1 1 \\ \hline 1 0 1 0 \end{array}$$

INSPECTION COPY

COPYRIGHT
PROTECTED

10

Binary Addition

The process of performing addition in binary is similar to the process of performing addition in decimal. There are four simple rules:

Rules:

$$0 + 0 = 0$$

$$0 + 1 = 1$$

$$1 + 1 = 0 \text{ carry } 1$$

$$1 + 1 + 1 = 1 \text{ carry } 1$$

$$\begin{array}{r} 1 \quad 1 \quad 1 \\ 0 \quad 1 \quad 1 \quad 0 \quad 1 \\ 0 \quad 0 \quad 1 \quad 0 \quad 1 \\ \hline 1 \quad 0 \quad 0 \quad 1 \quad 0 \end{array}$$

INSPECTION COPY

COPYRIGHT
PROTECTED

11

Binary Addition

The process of performing addition in binary is similar to the process of performing addition in decimal. There are four simple rules:

Rules:

$$0 + 0 = 0$$

$$0 + 1 = 1$$

$$1 + 1 = 0 \text{ carry } 1$$

$$1 + 1 + 1 = 1 \text{ carry } 1$$

$$\begin{array}{r} 1 \quad 1 \quad 1 \\ 0 \quad 1 \quad 1 \quad 0 \quad 1 \\ 0 \quad 0 \quad 1 \quad 0 \quad 1 \\ \hline 1 \quad 0 \quad 0 \quad 1 \quad 0 \end{array}$$

INSPECTION COPY

COPYRIGHT
PROTECTED

12

Binary Addition

The process of performing addition in binary is similar to the process of performing addition in decimal. There are four simple rules:

Rules:

$$0 + 0 = 0$$

$$0 + 1 = 1$$

$$1 + 1 = 0 \text{ carry } 1$$

$$1 + 1 + 1 = 1 \text{ carry } 1$$

$$\begin{array}{r} 1 \quad 1 \quad 1 \\ 0 \quad 1 \quad 1 \quad 0 \quad 1 \\ 0 \quad 0 \quad 1 \quad 0 \quad 1 \\ \hline 1 \quad 0 \quad 0 \quad 1 \quad 0 \end{array}$$

The total number is bigger than 5 bits (the size of the register). This is called an overflow because the computer cannot store the result. This is known as an overflow.

INSPECTION COPY

COPYRIGHT
PROTECTED

13

Binary Multiplication

Binary multiplication is a simple process.

$$\begin{array}{r} 1 \quad 0 \quad 1 \quad 0 \\ 0 \quad 1 \quad 0 \quad 1 \times \\ \hline 1 \quad 0 \quad 1 \quad 0 \\ 1 \quad 0 \quad 1 \quad 0 \\ \hline 1 \quad 1 \quad 0 \quad 0 \quad 1 \quad 0 \end{array}$$

Write the first number in the register.

Next add the number in the register.

INSPECTION COPY

COPYRIGHT
PROTECTED

14

Binary Multiplication

Here are some more examples:

$$\begin{array}{r} 1 \quad 1 \quad 0 \quad 0 \\ 0 \quad 1 \quad 1 \quad 0 \times \\ \hline 1 \quad 1 \quad 0 \quad 0 \\ 1 \quad 1 \quad 0 \quad 0 \\ \hline 1 \quad 0 \quad 0 \quad 1 \quad 0 \quad 0 \quad 0 \end{array}$$

For the last example we used the rule:

$1 + 1 + 1 + 1 = 0$ carry 0 and carry 1. This is because $1 + 1 + 1 + 1$ produces 4.

INSPECTION COPY

COPYRIGHT
PROTECTED

1

Negative Numbers

Photocopying this digital resource may only be done by the purchasing institution for a single site and for their own use.

INSPECTION COPY

COPYRIGHT
PROTECTED

2 Problem

The problem with sign and magnitude is add

```

0 0 0 0 0 1 1
1 0 0 0 0 1 0
1 0 0 0 1 1 0
  
```

INSPECTION COPY



3 Sign and Magnitude

Sign and magnitude is the simplest method for representing numbers in binary

The most significant bit is the

1 = Minus
0 = Plus

Sign Bit	64	32	16	8
0	0	0	1	0
1	0	0	1	0

INSPECTION COPY



4 Two's Complement

Two's complement is a method of representing negative numbers in binary.

The most significant bit is a negative

128	64	32	16	8
1	0	0	0	1

-128 + 8 + 4 +

It is easy to tell whether a two's complement number is positive. If the most significant bit is a 1 then it is negative. If it is 0 then it is positive.

INSPECTION COPY



5 Subtraction

Two's complement can help with binary subtraction

First we convert the second number to its negative equivalent in two's complement.

INSPECTION COPY



6 Subtraction

Two's complement can help with binary subtraction

First we convert the second number to its negative equivalent in two's complement.

We do this by reversing the bits and adding 1 to the number.

INSPECTION COPY



7 Subtraction

Two's complement can help with binary subtraction

First we convert the second number to its negative equivalent in two's complement.

We do this by reversing the bits and adding 1 to the number.

INSPECTION COPY



8

Subtractio

Two's complement can help with b

First we convert the second number to its negative equivalent in two's complement.

We do this by reversing the bits and adding 1 to the number.

Next we add the numbers together using the rules of binary addition.

INSPECTION COPY

COPYRIGHT
PROTECTED



9

Subtractio

Two's complement can help with b

First we convert the second number to its negative equivalent in two's complement.

We do this by reversing the bits and adding 1 to the number.

Next we add the numbers together using the rules of binary addition.

If there is an overflow it is discarded.

INSPECTION COPY

COPYRIGHT
PROTECTED



10

Another Exar

INSPECTION COPY

COPYRIGHT
PROTECTED



11

Another Exar

Reverse the bits of the second number.

INSPECTION COPY

COPYRIGHT
PROTECTED



12

Another Exar

Reverse the bits of the second number.

Add 1 to the second number.

INSPECTION COPY

COPYRIGHT
PROTECTED



13

Another Exar

Reverse the bits of the second number.

Add 1 to the second number.

Add the numbers together using the rules of binary addition.

INSPECTION COPY

COPYRIGHT
PROTECTED



Another Exam

Reverse the bits of the second number.

Add 1 to the second number.

Add the numbers together using the rules of binary addition.

Discard any overflow.

INSPECTION COPY

COPYRIGHT
PROTECTED



Fraction

Physical/digital resources may only be copied by the purchasing institution on a single site and for their own use

INSPECTION COPY

COPYRIGHT
PROTECTED



Fixed Point

Fixed point is the simplest method of representing a number using binary.

In a fixed-point binary number the values point are halved.

8	4	2	1	1/2
1	0	1	0	1

$$8 + 2 = 10$$

$$10.5$$

INSPECTION COPY

COPYRIGHT
PROTECTED



Another Exam

$$\frac{1}{2} = 0.5$$

$$\frac{1}{4} = 0.25$$

$$\frac{1}{8} = 0.125$$

$$\frac{1}{16} = 0.0625$$

8	4	2	1	1/2
1	0	1	1	1

$$8 + 2 + 1 = 11 \quad 0.5 +$$

$$11.625$$

INSPECTION COPY

COPYRIGHT
PROTECTED



Floating Point

Floating point is an alternative method of representing a number with a fractional part.

As the name suggests, the binary point can be in a fixed position.

A floating-point number is divided into the mantissa and the exponent.

0	1	1	0	1	0	0	0
---	---	---	---	---	---	---	---

Mantissa

This is the actual number

INSPECTION COPY

COPYRIGHT
PROTECTED



Floating Point

There are many different formats of floating point, but we are going to be working with the two's complement format.

0	1	1	0	1	0	0	0
---	---	---	---	---	---	---	---

Mantissa

INSPECTION COPY

COPYRIGHT
PROTECTED



Floating Po

There are many different formats of floating we are going to be working with the two

8	4	2	1	1/2	1/4	1/8	1/16
0	1	1	0	1	0	0	0

Mantissa

The value of the exponent is 3, so we need three places to the right

INSPECTION COPY

COPYRIGHT PROTECTED



Floating Po

There are many different formats of floating we are going to be working with the two

8	4	2	1	1/2	1/4	1/8	1/16
0	1	1	0	1	0	0	0

Mantissa

The value of the exponent is 3, so we need three places to the right

$$4 + 2 + 0.5 = 6.5$$

INSPECTION COPY

COPYRIGHT PROTECTED



Another Exar

Both the mantissa and the exponent are in If either starts with a 1 it needs to be converted by flipping the bits and adding 1

8	4	2	1	1/2	1/4	1/8	1/16
1	0	1	0	1	1	0	1

Mantissa

INSPECTION COPY

COPYRIGHT PROTECTED



Another Exar

Both the mantissa and the exponent are in If either starts with a 1 it needs to be converted by flipping the bits and adding 1

8	4	2	1	1/2	1/4	1/8	1/16
0	1	0	1	0	0	1	1

Mantissa

INSPECTION COPY

COPYRIGHT PROTECTED



Another Exar

Both the mantissa and the exponent are in If either starts with a 1 it needs to be converted by flipping the bits and adding 1

16	8	4	2	1	1/2	1/4	1/8
0	1	0	1	0	0	1	1

Mantissa

$$8 + 2 + 0.25 + 0.125 = 10.375$$

INSPECTION COPY

COPYRIGHT PROTECTED



Another Exar

Both the mantissa and the exponent are in If either starts with a 1 it needs to be converted by flipping the bits and adding 1

16	8	4	2	1	1/2	1/4	1/8
0	1	0	1	0	0	1	1

Mantissa

$$8 + 2 + 0.25 + 0.125 = 10.375$$

We know the result is a negative number because the sign bit is 1.

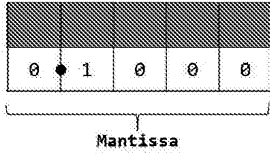
INSPECTION COPY

COPYRIGHT PROTECTED



Negative Expo

If the exponent is negative we shift the bin than to the right.



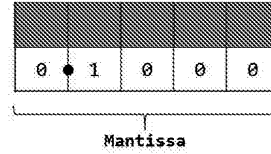
INSPECTION COPY

COPYRIGHT
PROTECTED
Zig
Zag
Education

Negative Expo

If the exponent is negative we shift the bin than to the right.

First we have to convert the exponent to a the number of places the binary po



INSPECTION COPY

COPYRIGHT
PROTECTED
Zig
Zag
Education

Negative Expo

If the exponent is negative we shift the bin than to the right.

First we have to convert the exponent to a the number of places the binary po



INSPECTION COPY

COPYRIGHT
PROTECTED
Zig
Zag
Education

Negative Expo

If the exponent is negative we shift the bin than to the right.

First we have to convert the exponent to a the number of places the binary po



INSPECTION COPY

COPYRIGHT
PROTECTED
Zig
Zag
Education

Negative Expo

If the exponent is negative we shift the bin than to the right.

First we have to convert the exponent to a the number of places the binary po



INSPECTION COPY

COPYRIGHT
PROTECTED
Zig
Zag
Education

Negative Expo

If the exponent is negative we shift the bin than to the right.

First we have to convert the exponent to a the number of places the binary po



INSPECTION COPY

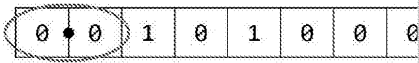
COPYRIGHT
PROTECTED
Zig
Zag
Education

0.25

Normalisati

When representing numbers in binary we want the smallest number of bits possible; to do this

To normalise a number you look at the standard deviation and whether there are any repeats.



COPYRIGHT

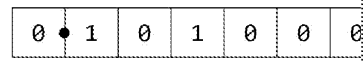


INSPECTION COPY

Normalisati

When representing numbers in binary we want the smallest number of bits possible; to do this

To normalise a number you look at the standard deviation and whether there are any repeated values.



The repeated values need to be removed and

COPYRIGHT

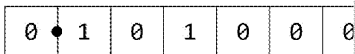


INSPECTION COPY

Normalisati

When representing numbers in binary we want the smallest number of bits possible; to do this

To normalise a number you look at the standard deviation and whether there are any repeats



The repeated values need to be removed and

The binary point has moved one place so the value is updated by subtracting 1

COPYRIGHT
PROTECTED



INSPECTION COPY

Comparison

It is faster to process calculations using fixed-point numbers than floating-point numbers.

Floating-point numbers can represent a large range of values, including fractional parts when compared to fixed-point numbers.

This means that floating-point numbers are not exact, where you need to represent a wide range of values.

On the other hand, fixed-point numbers are of processing is more important:

COPYRIGHT
PROTECTED



INSPECTION COPY

Rounding Er

Some numbers cannot be represented using the bits allocated to them. In this case the number is rounded to the nearest representable number.

The rounding error is the difference between the original value and the rounded value. There are two different methods for rounding: round up and round down. The precision of a rounded value is the number of digits after the decimal point.

Absolute Error	
The difference between the actual number and the rounded number	The difference between the actual number and the rounded number

COPYRIGHT
PROTECTED



INSPECTION COPY

Example

If we wanted to represent the decimal value 10 in binary we would end up

101.10011
(5.59375)

The absolute error

5.6 - 5.59375

The relative error is

$$0.00625 / 5.6 = 0.0011$$

COPYRIGHT
PROTECTED



INSPECTION COPY

1

Arrays

Photo credit: digital resources may only be copied by the purchasing institution on a single site and for their own use

INSPECTION COPY



2

Arrays

A variable can only store one value; if you need to store more than one value, you will need to use an array.

An array is a set of values of the same data type, each identified by a unique identifier.

Here is an example array designed to store names.

Index	0	1
Value	Alex	James

INSPECTION COPY



3

Index

We access the values stored in an array using an index.

Index	0	1
Value	Alex	James

If this array was called 'names' we could access this value using: `names[1]`

INSPECTION COPY



4

Two-Dimensional Arrays

Arrays can also be two-dimensional, allowing them to store multiple values. They are basically arrays of arrays.

Here is an example two-dimensional array used to store scores in a computer game.

		Columns (Levels)		
		0	1	2
Rows (Players)	0	4	4	3
	1	3	5	3
	2	2	4	3
	3	5	3	4

INSPECTION COPY



5

Pseudocode

We can create the one-dimensional names array using the following syntax:

```
names ← ["Alex", "James", "Paige"]
```

We can create the two-dimensional scores array using the following syntax:

```
scores ← [[4, 4, 3, 5], [3, 5, 3, 4], [2, 1, 4, 3]]
```

INSPECTION COPY



6

FOR Loops and Arrays

FOR loops and arrays go well together; they allow us to cycle through each element in an array.

This example program cycles through each element in the names array.

```
FOR i ← 0 TO length(array)
    OUTPUT names[i]
NEXT i
```

```
names ← ["Alex", "James", "Paige"]
```

INSPECTION COPY



FOR Loops and

FOR loops and arrays go well together; they cycle through each element in an array.

This example program cycles through each element in an array.

```
FOR i ← 0 to length(array)
  OUTPUT names[i]
NEXT i
```

names ← ["Alex", "James", "Patricia"]

INSPECTION COPY



FOR Loops and

FOR loops and arrays go well together; they cycle through each element in an array.

This example program cycles through each element in an array.

```
FOR i ← 0 to length(array)
  OUTPUT names[i]
NEXT i
```

names ← ["Alex", "James", "Patricia"]

INSPECTION COPY



FOR Loops and

FOR loops and arrays go well together; they cycle through each element in an array.

This example program cycles through each element in an array.

```
FOR i ← 0 to length(array)
  OUTPUT names[i]
NEXT i
```

names ← ["Alex", "James", "Patricia"]

INSPECTION COPY



FOR Loops and

FOR loops and arrays go well together; they cycle through each element in an array.

This example program cycles through each element in an array.

```
FOR i ← 0 to length(array)
  OUTPUT names[i]
NEXT i
```

names ← ["Alex", "James", "Patricia"]

INSPECTION COPY



FOR Loops and

FOR loops and arrays go well together; they cycle through each element in an array.

This example program cycles through each element in an array.

```
FOR i ← 0 to length(array)
  OUTPUT names[i]
NEXT i
```

names ← ["Alex", "James", "Patricia"]

INSPECTION COPY



FOR Loops and

FOR loops and arrays go well together; they cycle through each element in an array.

This example program cycles through each element in an array.

```
FOR i ← 0 to length(array)
  OUTPUT names[i]
NEXT i
```

names ← ["Alex", "James", "Patricia"]

INSPECTION COPY



13

FOR Loops and

FOR loops and arrays go well together; they cycle through each element in an array.

This example program cycles through each element in an array.

```
FOR i ← 0 to length(array)
    OUTPUT names[i]
NEXT i
```

```
names ← ["Alex", "James", "Patricia"]
```

INSPECTION COPY



14

FOR Loops and

FOR loops and arrays go well together; they cycle through each element in an array.

This example program cycles through each element in an array.

```
FOR i ← 0 to length(array)
    OUTPUT names[i]
NEXT i
```

```
names ← ["Alex", "James", "Patricia"]
```

INSPECTION COPY



15

FOR Loops and

FOR loops and arrays go well together; they cycle through each element in an array.

This example program cycles through each element in an array.

```
FOR i ← 0 to length(array)
    OUTPUT names[i]
NEXT i
```

```
names ← ["Alex", "James", "Patricia"]
```

INSPECTION COPY



16

FOR Loops and

FOR loops and arrays go well together; they cycle through each element in an array.

This example program cycles through each element in an array.

```
FOR i ← 0 to length(array)
    OUTPUT names[i]
NEXT i
```

```
names ← ["Alex", "James", "Patricia"]
```

INSPECTION COPY



17

FOR Loops and

FOR loops and arrays go well together; they cycle through each element in an array.

This example program cycles through each element in an array.

```
FOR i ← 0 to length(array)
    OUTPUT names[i]
NEXT i
```

```
names ← ["Alex", "James", "Patricia"]
```

INSPECTION COPY



18

Starting Index

In most programming languages the index of the first element in an array is 0.

The pseudocode featured in most exams uses the index of the first element.

Index	1	2
Value	Alex	James

INSPECTION COPY



1

Stacks and Queues

Photoshop/digital resources may only be copied by the purchasing institution on a single site and for their own use

INSPECTION COPY



2

Stacks

A stack is a data type that operates on either **first-in last-out (FILO)** or **first-out last-in (FIFO)**

Items can either be added to or removed from the stack

Susan
Steven

INSPECTION COPY



3

Operation

There are three main operations that can be performed on a stack

Push

Adds a new item to the top of the stack.

Barbara
Susan
Steven

INSPECTION COPY



4

Operation

There are three main operations that can be performed on a stack

Push

Adds a new item to the top of the stack.

Pop

Removes the top item from the stack

Susan
Steven

INSPECTION COPY



5

Operation

There are three main operations that can be performed on a stack

Push

Adds a new item to the top of the stack.

Pop

Removes the top item from the stack

Susan
Steven

INSPECTION COPY



6

Pushing

Stacks can be implemented using **arrays**. As the array is filled, a variable is used to record the position of the top item in the stack.

This is the pseudocode that can be used to push an item onto a stack.

```
IF Stack is full THEN
    Error
ELSE
    t = t + 1
    Stack[t] ← "Ian"
END IF
```

INSPECTION COPY



7

```

IF Stack is empty THEN
    Error
ELSE
    Stack[t] ← NULL
    t = t - 1
END IF

```

INSPECTION COPY

COPYRIGHT
PROTECTED

**Zig
Zag**
Education

3

Ian	Barbara	Su
-----	---------	----

Front Re



COPYRIGHT
PROTECTED

**Zig
Zag**
Education

9

Jan	Barbara	Su
-----	---------	----

Front



INSPECTION COPY

COPYRIGHT
PROTECTED

**Zig
Zag**
Education

10

	Index	Value
Rear \longrightarrow	1	Harry
Front \longrightarrow	2	Barbara
	3	Susan
	4	Steven
	5	Sarah

COPYRIGHT
PROTECTED

**Zig
Zag**
Education

11

	Index	Value
Front \rightarrow	1	
	2	Barbara
Rear \rightarrow	3	Susan
	4	
	5	

INSPECTION COPY

COPYRIGHT
PROTECTED

**Zig
Zag**
Education

12

	Index	Value
	1	
Front →	2	Barbara
Rear →	3	Susan
	4	
	5	

COPYRIGHT
PROTECTED

**Zig
Zag**
Education

13

Circular Queue

Queues can be created using an array; however, when an element is removed it leaves a space that cannot be used.

Circular queues solve this problem by using the space at the beginning of the array when the end of the array has been reached.

Index	Value
1	
Front → 2	Barbara
3	Susan
Rear → 4	Steven
5	

INSPECTION COPY

COPYRIGHT
PROTECTED

14

Circular Queue

Queues can be created using an array; however, when an element is removed it leaves a space that cannot be used.

Circular queues solve this problem by using the space at the beginning of the array when the end of the array has been reached.

Index	Value
1	
Front → 2	Barbara
3	Susan
4	Steven
Rear → 5	Sarah

INSPECTION COPY

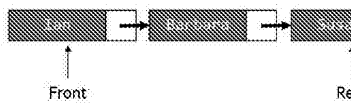
COPYRIGHT
PROTECTED

15

Linear Queue

Queues can also be created using linked lists. A linear queue is a queue implemented using a linked list.

Items are added to the rear of the queue and removed from the front of the queue.



INSPECTION COPY

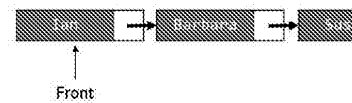
COPYRIGHT
PROTECTED

16

Linear Queue

Queues can also be created using linked lists. A linear queue is a queue implemented using a linked list.

Items are added to the rear of the queue and removed from the front of the queue.



INSPECTION COPY

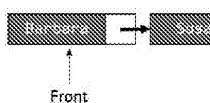
COPYRIGHT
PROTECTED

17

Linear Queue

Queues can also be created using linked lists. A linear queue is a queue implemented using a linked list.

Items are added to the rear of the queue and removed from the front of the queue.



INSPECTION COPY

COPYRIGHT
PROTECTED

Linked Lists Hash Table

A Level
Only

Photocopying this digital resource may only be copied by the purchasing institution for a single site and for their own use.

INSPECTION COPY

COPYRIGHT
PROTECTED

2 Arrays

Arrays allow us to store a series of values.
Each element has an index and can be accessed by its index.

When an array is created, its size is declared.
This means it occupies a fixed section of memory; this makes it efficient.

Linked lists consist of **nodes** and each node contains a pointer to the next node in the list, making them dynamic.

INSPECTION COPY

COPYRIGHT
PROTECTED
Zig
Zag
Education

3 Linked Lists

Each node in a linked list contains a pointer to the next node in the list.



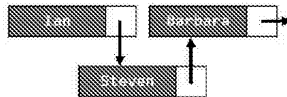
The last node in the list is indicated by a null pointer.

INSPECTION COPY

COPYRIGHT
PROTECTED
Zig
Zag
Education

4 Linked Lists

Each node in a linked list contains a pointer to the next node in the list.



The last node in the list is indicated by a null pointer.

New nodes can easily be added or removed from the list.

INSPECTION COPY

COPYRIGHT
PROTECTED
Zig
Zag
Education

5 Disadvantages

A disadvantage of linked lists is that elements are not stored in order. The entire list has to be traversed to find a specific element.

Hash tables offer a solution to this problem by storing data in a way that enables direct access to the data.

Hash tables are used when speedy insertion and retrieval of data is required.

INSPECTION COPY

COPYRIGHT
PROTECTED
Zig
Zag
Education

6 Hash Tables

Hash tables consist of two parts: an array and a hash function.

The hash function takes a piece of data known as a key and returns a hash value; this is used as the index to access the array.

In this example each value is assigned a pointer to its first letter.



INSPECTION COPY

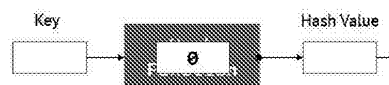
COPYRIGHT
PROTECTED
Zig
Zag
Education

7 Hash Tables

Hash tables consist of two parts: an array and a hash function.

The hash function takes a piece of data known as a key and returns a hash value; this is used as the index to access the array.

In this example each value is assigned a pointer to its first letter.



INSPECTION COPY

COPYRIGHT
PROTECTED
Zig
Zag
Education

8

Hash Table

Hash tables consist of two parts: an array and a hash function.

The hash function takes a piece of data known as a key and produces a hash value; this is used as the index into the array.

In this example each value is assigned a position based on its first letter.



INSPECTION COPY

COPYRIGHT
PROTECTED

Zig
Zog
Education

9

Hash Table

Hash tables consist of two parts: an array and a hash function.

The hash function takes a piece of data known as a key and produces a hash value; this is used as the index into the array.

In this example each value is assigned a position based on its first letter.



INSPECTION COPY

COPYRIGHT
PROTECTED

Zig
Zog
Education

10

Hash Table

Hash tables consist of two parts: an array and a hash function.

The hash function takes a piece of data known as a key and produces a hash value; this is used as the index into the array.

In this example each value is assigned a position based on its first letter.



INSPECTION COPY

COPYRIGHT
PROTECTED

Zig
Zog
Education

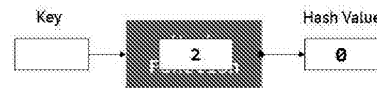
11

Hash Table

Hash tables consist of two parts: an array and a hash function.

The hash function takes a piece of data known as a key and produces a hash value; this is used as the index into the array.

In this example each value is assigned a position based on its first letter.



INSPECTION COPY

COPYRIGHT
PROTECTED

Zig
Zog
Education

12

Hash Table

Hash tables consist of two parts: an array and a hash function.

The hash function takes a piece of data known as a key and produces a hash value; this is used as the index into the array.

In this example each value is assigned a position based on its first letter.



INSPECTION COPY

COPYRIGHT
PROTECTED

Zig
Zog
Education

13

Hash Table

Hash tables consist of two parts: an array and a hash function.

The hash function takes a piece of data known as a key and produces a hash value; this is used as the index into the array.

In this example each value is assigned a position based on its first letter.



INSPECTION COPY

COPYRIGHT
PROTECTED

Zig
Zog
Education

14

Hash Function

This is the pseudocode for a simple hash function that takes a letter of a key and converts it to a hash value.

The letter is converted to its ASCII value (for example, 'A' is 65) and then a constant value is subtracted from it.

In this example the key is 'A'.

```
FUNCTION hashFunction(key)
    hash = ASCII(key[0]) - 66
    RETURN hash
END FUNCTION
```

INSPECTION COPY



15

Hash Function

This is the pseudocode for a simple hash function that takes a letter of a key and converts it to a hash value.

The letter is converted to its ASCII value (for example, 'A' is 65) and then a constant value is subtracted from it.

In this example the key is 'A'.

```
FUNCTION hashFunction(key)
    hash = ASCII(key[0]) - 66
    RETURN hash
END FUNCTION
```

INSPECTION COPY



16

Hash Function

This is the pseudocode for a simple hash function that takes a letter of a key and converts it to a hash value.

The letter is converted to its ASCII value (for example, 'A' is 65) and then a constant value is subtracted from it.

In this example the key is 'A'.

```
FUNCTION hashFunction(key)
    hash = ASCII(key[0]) - 66
    RETURN hash
END FUNCTION
```

INSPECTION COPY



17

Collisions

If two keys produce the same value, a collision occurs. For example, Alex and Andy would both have a hash value of 0.



There are two methods of dealing with collisions: linear probing and separate chaining.

INSPECTION COPY



18

Linear Probing

The linear probing method stores the value in the next available position.



The function 'probes' the list until it finds an empty position.

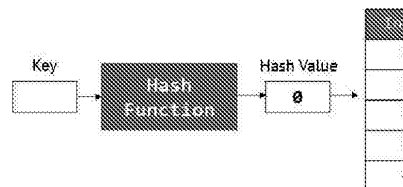
INSPECTION COPY



19

Separate Chaining

The separate chaining method uses lists to store values that hash to the same position.



Values that hash to the same position are placed in a list at that position.

INSPECTION COPY



1

Graphs and

A Level
Only

Photocopyable digital resources may only be copied by the purchasing institution on a single site and for their own use

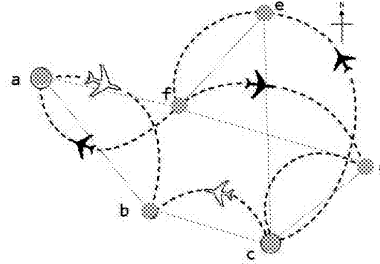
INSPECTION COPY

COPYRIGHT
PROTECTED
Zig Zag
Education

2 Graphs

A graph is a data structure used to represent connections between

Example: airline flight paths between



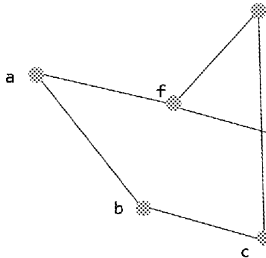
COPYRIGHT
PROTECTED
Zig Zag
Education

3

Structure

Each object in a graph is called a vertex
are called vertices

Each connection in a graph is called an edge



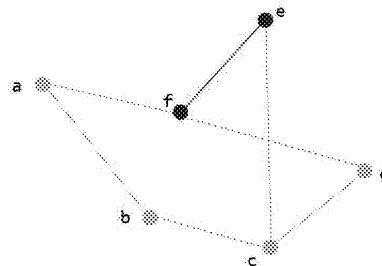
INSPECTION COPY

COPYRIGHT
PROTECTED
Zig Zag
Education

4

Neighbours and

Neighbours are vertices connected by an edge
for example, E and F are neighbours



INSPECTION COPY

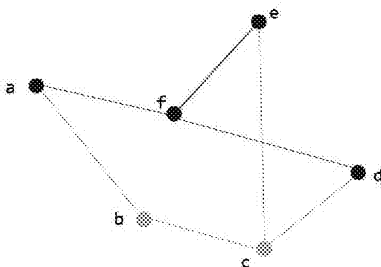
COPYRIGHT
PROTECTED
Zig Zag
Education

5

Neighbours and

Neighbours are vertices connected by an edge
for example, E and F are neighbours

The degree of a vertex is the number of edges connected to it
for example, the degree of vertex E is 3



INSPECTION COPY

COPYRIGHT
PROTECTED
Zig Zag
Education

6

Weighted and Directed

A weighted graph is one in which the edges are labelled with distances for example.

In a directed graph each edge has a direction associated with it.

INSPECTION COPY

COPYRIGHT
PROTECTED
Zig Zag
Education

Adjacency Matrix

An adjacency matrix can be used to represent a graph that is processed by a computer.

If two vertices are connected, a 1 is placed in the matrix in two positions.

For example, A and D are connected so a 1 is placed in the two positions in the matrix where the vertices meet.

The remaining spaces are then filled with 0s or the null symbol (\emptyset).

If graphs are weighted, the weight is placed in the matrix.

The adjacency matrix for an undirected graph is always symmetrical.

INSPECTION COPY

COPYRIGHT
PROTECTED

Adjacency List

An adjacency list is an alternative method of representing a graph.

For each vertex it lists the vertices that it is connected to.

Vertex	Adjacent Vertices
A	D, E
B	A, D, C
C	B, D
D	A, B, C

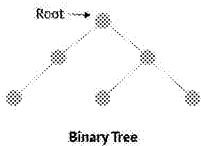
Adjacency lists are best used when there are many vertices.

INSPECTION COPY

COPYRIGHT
PROTECTED

Trees

A tree is a graph that takes on the form of a tree. It is a graph with no cycles (only one path between any two vertices).



A rooted tree has a vertex that is designated as the root.

A binary tree is a rooted tree with a maximum of two children per vertex.

INSPECTION COPY

COPYRIGHT
PROTECTED

Boolean Algebra

Photocopiable/digital resources may only be copied by the purchasing institution on a single site and for their own use.

INSPECTION COPY

COPYRIGHT
PROTECTED

Karnaugh Maps

Karnaugh maps allow us to simplify truth table patterns.

In this example we are going to simplify the expression:

$$(A \wedge \neg B) \vee (A \wedge C) \vee (B \wedge \neg C)$$

The first stage is to create a truth table for the expression.

INSPECTION COPY

COPYRIGHT
PROTECTED

Karnaugh Maps

The next stage is to convert the truth table into a Karnaugh map.

A	B	C	Output
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

A \ B C	00	01	11	10
0	0	0	1	0
1	1	1	1	1

INSPECTION COPY

COPYRIGHT
PROTECTED

4

Karnaugh M

The next stage is to convert the truth tab

A	B	C	Output
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

A \ B C	00	01	11	10
0	0	0	0	0
1	1	0	1	0

COPYRIGHT
PROTECTED

INSPECTION COPY

5

Karnaugh M

The next stage is to convert the truth tab

A	B	C	Output
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

A \ B C	00	01	11	10
0	0	0	0	0
1	1	0	1	0

COPYRIGHT
PROTECTED

INSPECTION COPY

6

Karnaugh M

The next stage is to convert the truth tab

A	B	C	Output
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

A \ B C	00	01	11	10
0	0	0	0	0
1	1	0	1	0

COPYRIGHT
PROTECTED

INSPECTION COPY

7

Karnaugh M

The next stage is to convert the truth tab

A	B	C	Output
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

A \ B C	00	01	11	10
0	0	0	0	0
1	1	0	1	0

COPYRIGHT
PROTECTED

INSPECTION COPY

8

Karnaugh M

The next stage is to convert the truth tab

A	B	C	Output
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

A \ B C	00	01	11	10
0	0	0	0	0
1	1	0	1	0

We can then
use theseCOPYRIGHT
PROTECTED

INSPECTION COPY

9

Simplifying the Ex

We need to look at each group and work out
constant throughout each

A \ B C	00	01	11	10
0	0	0	0	0
1	1	1	1	1

A is the constant in the red group, so A forms
expression.B and C are constant in the second group so
of the expression

$$A + B + C$$

COPYRIGHT
PROTECTED

INSPECTION COPY

De Morgan's Laws

When designing a circuit it is usually cheaper to use a single type of logic gate, so it's useful to be able to design systems using only AND gates or only OR gates.

We use De Morgan's Laws to change between AND and OR gates.

To change OR to an AND function

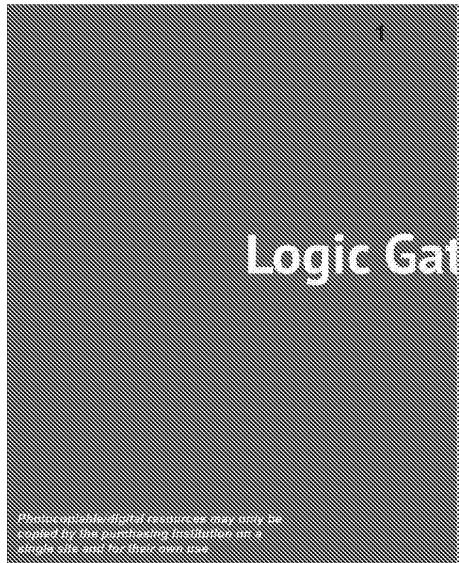
Change the operator to an AND

Invert each part of the expression

Invert the whole expression

De Morgan's Laws: $\neg(A \vee B) = \neg A \wedge \neg B$ and $\neg(A \wedge B) = \neg A \vee \neg B$

INSPECTION COPY

COPYRIGHT
PROTECTED

INSPECTION COPY

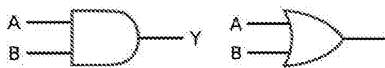
COPYRIGHT
PROTECTED

Please note that digital resources may only be copied by the purchasing institution on a single site and for their own use.

Logic Gate

Logic gates are the basic components of digital circuits. Each logic gate performs a different logical operation.

These are the three basic logic gates.



AND Gate

Outputs a 1 if both inputs are 1



OR Gate

Outputs a 1 if one or both of the inputs are 1

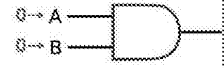
INSPECTION COPY

COPYRIGHT
PROTECTED

Truth Table

All the possible outcomes of a logic diagram are listed in a truth table.

This is the truth table for the AND gate.



Input A	Input B	Output Y
0	0	0
0	1	0
1	0	0
1	1	1

INSPECTION COPY

COPYRIGHT
PROTECTED

Truth Table

All the possible outcomes of a logic diagram are listed in a truth table.

This is the truth table for the AND gate.



Input A	Input B	Output Y
0	0	0
0	1	0
1	0	0
1	1	1

INSPECTION COPY

COPYRIGHT
PROTECTED

Truth Table

All the possible outcomes of a logic diagram are listed in a truth table.

This is the truth table for the AND gate.



Input A	Input B	Output Y
0	0	0
0	1	0
1	0	0
1	1	1

INSPECTION COPY

COPYRIGHT
PROTECTED

6

Truth Table

All the possible outcomes of a logic diagram are shown in a truth table.

This is the truth table for the



Input A	Input B
0	0
0	1
1	0
1	1

INSPECTION COPY

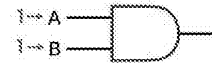
COPYRIGHT
PROTECTED

7

Truth Table

All the possible outcomes of a logic diagram are shown in a truth table.

This is the truth table for the



Input A	Input B
0	0
0	1
1	0
1	1

INSPECTION COPY

COPYRIGHT
PROTECTED

8

Truth Table

OR Gate



Input A	Input B	Output Y
0	0	0
0	1	1
1	0	1
1	1	1

INSPECTION COPY

COPYRIGHT
PROTECTED

9

Other Gate

There are some other logic gates you



XOR Gate

Only outputs a 1 if one of the inputs is 1 (not both)

NOR Gate

This is equivalent to an OR gate followed by a NOT gate

INSPECTION COPY

COPYRIGHT
PROTECTED

10

Truth Table

XOR Gate



Input A	Input B	Output Y
0	0	0
0	1	1
1	0	1
1	1	0

Input A	Input B	Output Y
0	0	0
0	1	1
1	0	1
1	1	0

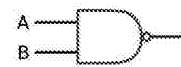
INSPECTION COPY

COPYRIGHT
PROTECTED

11

Truth Table

NAND Gate



Input A	Input B	Output Y
0	0	1
0	1	1
1	0	1
1	1	0

INSPECTION COPY

COPYRIGHT
PROTECTED

12

Boolean Expressions

Logic diagrams can also be represented using Boolean expressions.

Logic Gate

AND		
OR		
NOT		
XOR		

INSPECTION COPY

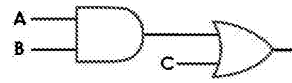
COPYRIGHT PROTECTED



13

Combining Gates

Logic gates can be combined to build more complex circuits.



Boolean Expression: $Y = (A \cdot B) + C$

INSPECTION COPY

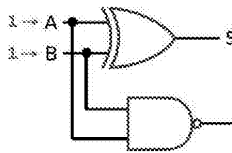
COPYRIGHT PROTECTED



14

Half Adder

The half adder circuit is used to add two single-bit numbers, producing a sum and a carry.



$1 + 1 = 0 \text{ carry}$

INSPECTION COPY

COPYRIGHT PROTECTED

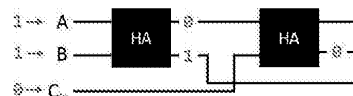


15

Full Adder

Two half adders are joined together along with an OR gate to form a full adder.

A full adder enables the carry from the last addition to be carried forward.



INSPECTION COPY

COPYRIGHT PROTECTED

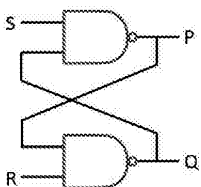


16

Flip-Flops

Flip-flops are digital circuits made up of logic gates that store a single bit of memory.

They stay in the same state even after the clock signal has stopped.



This is the basic SR flip-flop.

INSPECTION COPY

COPYRIGHT PROTECTED



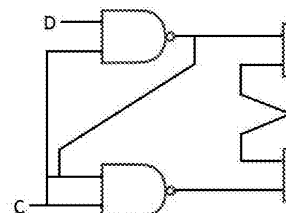
17

D-Type Flip-Flop

Two flip-flop circuits can be joined together to form a D-type flip-flop.

The D-type flip-flop is used to delay a signal.

It is delayed by one pulse of the clock signal.



INSPECTION COPY

COPYRIGHT PROTECTED



1

Sequence and Statement

Photo credit: digital resources may only be copied by the purchasing institution on a single site and for their own use

INSPECTION COPY

COPYRIGHT
PROTECTED

Zig
Zag
Education

2

Sequence

The statement is the simplest program

It represents a single instruction and usually a line of code like the one shown

```
area = width * height
```

INSPECTION COPY

COPYRIGHT
PROTECTED

Zig
Zag
Education

3

Sequence

The statement is the simplest program

It represents a single instruction and usually a line of code like the one shown

```
INPUT width, height
area = width * height
OUTPUT area
```

A sequence of statements (carried out in order)

This is an example of a sequence that is designed to calculate the area of a rectangle.

INSPECTION COPY

COPYRIGHT
PROTECTED

Zig
Zag
Education

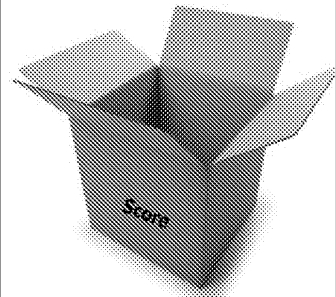
4

Variables

Where a value is stored, and can be changed

A variable is a location in memory that stores a value

Think of it like a box



INSPECTION COPY

COPYRIGHT
PROTECTED

Zig
Zag
Education

5

Variables

There are three variables in this program: width, height, and area. We will look at how they change as the program runs.

```
INPUT width, height
area = width * height
OUTPUT area
```

Var
wid
hei
are
Out

The user has inputted the values 3 and 5. The program has calculated the area of the rectangle.

INSPECTION COPY

COPYRIGHT
PROTECTED

Zig
Zag
Education

6

Variables

There are three variables in this program: width, height, and area. We will look at how they change as the program runs.

```
INPUT width, height
area = width * height
OUTPUT area
```

Var
wid
hei
are
Out

The user has inputted the values 3 and 5. The program has calculated the area of the rectangle.

The value stored in the width variable is multiplied by the value stored in the height variable; the result is stored in the area variable.

INSPECTION COPY

COPYRIGHT
PROTECTED

Zig
Zag
Education

Variables

There are three variables in this program: width, height and area. We will see how they change as the program runs.

```
INPUT width, height
area = width * height
OUTPUT area
```

The user has inputted the values 3 and 5. The program has calculated the area of the rectangle.

The value stored in the width variable is multiplied by the value stored in the height variable; the result is stored in the area variable.

The value stored in the area variable is 15.

INSPECTION COPY

COPYRIGHT
PROTECTED



Selection

```
INPUT guess
IF guess == 4 THEN
    OUTPUT "Well done"
ELSE
    OUTPUT "Try again"
END IF
```

In selection statements, the program decides which path to take.

In this example, the program checks if the user has guessed 4. If they have, it outputs "Well done". If not, it outputs "Try again".

INSPECTION COPY

COPYRIGHT
PROTECTED



Selection

```
INPUT guess
IF guess == 4 THEN
    OUTPUT "Well done"
ELSE
    OUTPUT "Try again"
END IF
```

In selection statements, the program decides which path to take.

In this example, the program checks if the user has guessed 4. If they have, it outputs "Well done". If not, it outputs "Try again".

This is a simple selection statement.

INSPECTION COPY

COPYRIGHT
PROTECTED



Selection

```
INPUT guess
IF guess == 4 THEN
    OUTPUT "Well done"
ELSE
    OUTPUT "Try again"
END IF
```

In selection statements, the program decides which path to take.

In this example, the program checks if the user has guessed 4. If they have, it outputs "Well done". If not, it outputs "Try again".

This is a simple selection statement.

If the result of the condition is TRUE then the program will execute the code in the THEN block. If the result is FALSE, the program will execute the code in the ELSE block.

INSPECTION COPY

COPYRIGHT
PROTECTED



Selection

```
INPUT guess
IF guess == 4 THEN
    OUTPUT "Well done"
ELSE
    OUTPUT "Try again"
END IF
```

In selection statements, the program decides which path to take.

In this example, the program checks if the user has guessed 4. If they have, it outputs "Well done". If not, it outputs "Try again".

This is a simple selection statement.

If the result of the condition is TRUE then the program will execute the code in the THEN block. If the result is FALSE, the program will execute the code in the ELSE block.

Otherwise this line of code will not be executed.

INSPECTION COPY

COPYRIGHT
PROTECTED



Selection

Operator	Meaning
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to
==	Equal to
!=	Not equal to

In selection statements, the program decides which path to take.

In this example, the program checks if the user has guessed 4. If they have, it outputs "Well done". If not, it outputs "Try again".

This is a simple selection statement.

If the result of the condition is TRUE then the program will execute the code in the THEN block. If the result is FALSE, the program will execute the code in the ELSE block.

Otherwise this line of code will not be executed.

These are the different comparison operators used in selection statements.

INSPECTION COPY

COPYRIGHT
PROTECTED



1

Iteration

Photo credit: digital resources may only be copied by the purchasing institution on a single site and for their own use

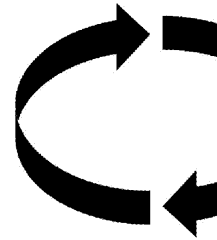
INSPECTION COPY



2

Iteration

When programming, it is often necessary to repeat instructions several times.



Iteration allows us to do this by repeating a group of instructions a number of times or until a condition is met.

INSPECTION COPY



3

Types of Iteration

There are two types of iteration:

Count-controlled

Used to repeat a group of statements a set number of times.

Examples:

FOR loop

INSPECTION COPY



4

FOR Loop

A FOR loop uses a variable as a counter; the statements are repeated until the counter reaches the specified value.

Watch what happens to the output and the statements below as the loop is executed.

```
FOR i = 1 to 3
    OUTPUT i*i
NEXT i
```

INSPECTION COPY



5

FOR Loop

A FOR loop uses a variable as a counter; the statements are repeated until the counter reaches the specified value.

Watch what happens to the output and the statements below as the loop is executed.

```
FOR i = 1 to 3
    OUTPUT i*i
NEXT i
```

INSPECTION COPY



6

FOR Loop

A FOR loop uses a variable as a counter; the statements are repeated until the counter reaches the specified value.

Watch what happens to the output and the statements below as the loop is executed.

```
FOR i = 1 to 3
    OUTPUT i*i
NEXT i
```

INSPECTION COPY



7 FOR Loop

A FOR loop uses a variable as a counter; the number of times the statements are repeated until it reaches the end of the loop.

Watch what happens to the output and the code below is executed

```
FOR i = 1 to 3
    OUTPUT i*i
NEXT i
```

INSPECTION COPY



8 FOR Loop

A FOR loop uses a variable as a counter; the number of times the statements are repeated until it reaches the end of the loop.

Watch what happens to the output and the code below is executed

```
FOR i = 1 to 3
    OUTPUT i*i
NEXT i
```

INSPECTION COPY



9 FOR Loop

A FOR loop uses a variable as a counter; the number of times the statements are repeated until it reaches the end of the loop.

Watch what happens to the output and the code below is executed

```
FOR i = 1 to 3
    OUTPUT i*i
NEXT i
```

INSPECTION COPY



10 WHILE Loop

A WHILE loop uses a condition to determine if statements should be executed. The statements are repeated as long as the result of the condition is true.

Watch what happens to the output and the code below is executed

```
i = 1
WHILE i < 4
    OUTPUT i*i
    i = i + 1
END WHILE
```

INSPECTION COPY



11 WHILE Loop

A WHILE loop uses a condition to determine if statements should be executed. The statements are repeated as long as the result of the condition is true.

Watch what happens to the output and the code below is executed

```
i = 1
WHILE i < 4
    OUTPUT i*i
    i = i + 1
END WHILE
```

INSPECTION COPY



12 WHILE Loop

A WHILE loop uses a condition to determine if statements should be executed. The statements are repeated as long as the result of the condition is true.

Watch what happens to the output and the code below is executed

```
i = 1
WHILE i < 4
    OUTPUT i*i
    i = i + 1
END WHILE
```

INSPECTION COPY



13

WHILE Loop

A WHILE loop uses a condition to determine if statements should be executed. The statements are repeated as long as the result of the condition is true.

Watch what happens to the output and the code below is executed

```
i = 1
WHILE i < 4
    OUTPUT i*i
    i = i + 1
END WHILE
```

INSPECTION COPY



14

WHILE Loop

A WHILE loop uses a condition to determine if statements should be executed. The statements are repeated as long as the result of the condition is true.

Watch what happens to the output and the code below is executed

```
i = 1
WHILE i < 4
    OUTPUT i*i
    i = i + 1
END WHILE
```

INSPECTION COPY



15

WHILE Loop

A WHILE loop uses a condition to determine if statements should be executed. The statements are repeated as long as the result of the condition is true.

Watch what happens to the output and the code below is executed

```
i = 1
WHILE i < 4
    OUTPUT i*i
    i = i + 1
END WHILE
```

INSPECTION COPY



16

WHILE Loop

A WHILE loop uses a condition to determine if statements should be executed. The statements are repeated as long as the result of the condition is true.

Watch what happens to the output and the code below is executed

```
i = 1
WHILE i < 4
    OUTPUT i*i
    i = i + 1
END WHILE
```

INSPECTION COPY



17

REPEAT UNTIL

In a REPEAT UNTIL loop the statements are repeated as long as the condition is true. The condition is tested after the statements are executed.

Watch what happens to the output and the code below is executed

```
i = 1
REPEAT
    OUTPUT i*i
    i = i + 1
UNTIL i = 4
```

INSPECTION COPY



18

REPEAT UNTIL

In a REPEAT UNTIL loop the statements are repeated as long as the condition is true. The condition is tested after the statements are executed.

Watch what happens to the output and the code below is executed

```
i = 1
REPEAT
    OUTPUT i*i
    i = i + 1
UNTIL i = 4
```

INSPECTION COPY



19

REPEAT UNTIL

In a REPEAT UNTIL loop the statements are executed until the condition is tested. The statements are repeated as long as the condition is false.

Watch what happens to the output and the condition below is executed

```
i = 1
REPEAT
  OUTPUT i*i
  i = i + 1
UNTIL i = 4
```

INSPECTION COPY



20

REPEAT UNTIL

In a REPEAT UNTIL loop the statements are executed until the condition is tested. The statements are repeated as long as the condition is false.

Watch what happens to the output and the condition below is executed

```
i = 1
REPEAT
  OUTPUT i*i
  i = i + 1
UNTIL i = 4
```

INSPECTION COPY



21

REPEAT UNTIL

In a REPEAT UNTIL loop the statements are executed until the condition is tested. The statements are repeated as long as the condition is false.

Watch what happens to the output and the condition below is executed

```
i = 1
REPEAT
  OUTPUT i*i
  i = i + 1
UNTIL i = 4
```

INSPECTION COPY



22

REPEAT UNTIL

In a REPEAT UNTIL loop the statements are executed until the condition is tested. The statements are repeated as long as the condition is false.

Watch what happens to the output and the condition below is executed

```
i = 1
REPEAT
  OUTPUT i*i
  i = i + 1
UNTIL i = 4
```

INSPECTION COPY



23

REPEAT UNTIL

In a REPEAT UNTIL loop the statements are executed until the condition is tested. The statements are repeated as long as the condition is false.

Watch what happens to the output and the condition below is executed

```
i = 1
REPEAT
  OUTPUT i*i
  i = i + 1
UNTIL i = 4
```

INSPECTION COPY



1

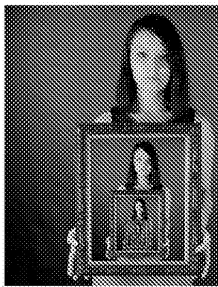
Recursion

INSPECTION COPY



Printed copy of this resource may only be copied by the purchasing institution for a single site and for their own use.

2 Recursion



Recursion occurs when a subrou

INSPECTION COPY

COPYRIGHT
PROTECTED
Zig
Zog
Education

3 Example

The recursive procedure below counts down
passed to it.

You will now see the result of calling the c
countdown(2).

```
1 PROCEDURE countdown(n)
2   IF n <= 1 THEN
3     OUTPUT 1
4   ELSE
5     OUTPUT n
6     countdown(n-1)
7   END IF
8 END PROCEDURE
```

countdown(2)
2 (FALSE)
4
5 OUTPUT
6 countdown(1)
7
8

COPYRIGHT
PROTECTED
Zig
Zog
Education

INSPECTION COPY

4 Trace Table

You need to be able to trace the execution of
a trace table.

In this example, the countdown procedure
countdown(3):

```
1 PROCEDURE countdown(n)
2   IF n <= 1 THEN
3     OUTPUT 1
4   ELSE
5     OUTPUT n
6     countdown(n-1)
7   END IF
8 END PROCEDURE
```

INSPECTION COPY

COPYRIGHT
PROTECTED
Zig
Zog
Education

5 Trace Table

You need to be able to trace the execution of
a trace table.

In this example, the countdown procedure
countdown(3):

```
1 PROCEDURE countdown(n)
2   IF n <= 1 THEN
3     OUTPUT 1
4   ELSE
5     OUTPUT n
6     countdown(n-1)
7   END IF
8 END PROCEDURE
```

COPYRIGHT
PROTECTED
Zig
Zog
Education

INSPECTION COPY

6 Trace Table

You need to be able to trace the execution of
a trace table.

In this example, the countdown procedure
countdown(3):

```
1 PROCEDURE countdown(n)
2   IF n <= 1 THEN
3     OUTPUT 1
4   ELSE
5     OUTPUT n
6     countdown(n-1)
7   END IF
8 END PROCEDURE
```

INSPECTION COPY

COPYRIGHT
PROTECTED
Zig
Zog
Education

7 Trace Table

You need to be able to trace the execution of
a trace table.

In this example, the countdown procedure
countdown(3):

```
1 PROCEDURE countdown(n)
2   IF n <= 1 THEN
3     OUTPUT 1
4   ELSE
5     OUTPUT n
6     countdown(n-1)
7   END IF
8 END PROCEDURE
```

COPYRIGHT
PROTECTED
Zig
Zog
Education

INSPECTION COPY

8

Trace Table

You need to be able to trace the execution of a program using a trace table.

In this example, the countdown procedure is called with the argument 3: `countdown(3);`

```
1 PROCEDURE countdown(n)
2   IF n <= 1 THEN
3     OUTPUT 1
4   ELSE
5     OUTPUT n
6     countdown(n-1)
7   END IF
8 END PROCEDURE
```

INSPECTION COPY



9

Trace Table

You need to be able to trace the execution of a program using a trace table.

In this example, the countdown procedure is called with the argument 3: `countdown(3);`

```
1 PROCEDURE countdown(n)
2   IF n <= 1 THEN
3     OUTPUT 1
4   ELSE
5     OUTPUT n
6     countdown(n-1)
7   END IF
8 END PROCEDURE
```

INSPECTION COPY



10

Infinite Recursion

If a subroutine calls itself, it is possible to have infinite recursion.

This means the subroutine will keep running out of available memory.

```
1 PROCEDURE countdown
2   IF n <= 1 THEN
3     OUTPUT 1
4   ELSE
5     OUTPUT n
6     countdown
7   END IF
8 END PROCEDURE
```

To avoid this you need a stopping condition.

INSPECTION COPY



1

Subroutine

Procedures and subroutines may only be copied by the purchasing institution on a single site and for their own use.

INSPECTION COPY



2

Subroutine

When programming we often need to perform the same task many times, at different points in the program; this is where subroutines are handy.

A subroutine is a section of code that performs a specific task and can be called from the main routine.

Program

Subroutine 1

Subroutine 2

INSPECTION COPY



3

Types of Subroutine

There are two types of subroutine:

Procedure

A subroutine that does not normally return a value to the main program.

A subroutine that returns a value to the main program.

INSPECTION COPY



4

Parameter

When writing a subroutine you need to tell the computer what variables it needs to use.

This is usually done by adding variable names to the subroutine definition. These variables are called parameters.

```
PROCEDURE Compare(a, b)
  IF a > b THEN
    OUTPUT a
  ELSE
    OUTPUT b
  END IF
END PROCEDURE
```

INSPECTION COPY



5

Argument

Subroutines won't run unless they are called. The variables that are passed to a subroutine when it is called are known as arguments.

```
PROCEDURE Compare(a, b)
  IF a > b THEN
    OUTPUT a
  ELSE
    OUTPUT b
  END IF
END PROCEDURE

Compare(4, 5) ←
```

INSPECTION COPY



6

Example

The procedure shown below accepts two variables as input, compares them, and then outputs the largest. It is called with the values 4 and 5.

```
PROCEDURE Compare(a, b)
  IF a > b THEN
    OUTPUT a
  ELSE
    OUTPUT b
  END IF
END PROCEDURE

Compare(4, 5)
```

INSPECTION COPY



7

Example

The procedure shown below accepts two variables as input, compares them, and then outputs the largest. It is called with the values 4 and 5.

```
PROCEDURE Compare(a, b)
  IF a > b THEN
    OUTPUT a
  ELSE
    OUTPUT b
  END IF
END PROCEDURE

Compare(4, 5)
```

INSPECTION COPY



8

Functions

Remember a function is a subroutine that returns a value to the calling routine.

The AreaCalc function is designed to calculate the area of a rectangle and return it to the calling routine where it can be used in further calculations.

```
FUNCTION AreaCalc(w, h)
  Area = w * h
  RETURN Area
END FUNCTION

Result = AreaCalc(5, 3)
OUTPUT Result
```

INSPECTION COPY



9

Functions

Remember a function is a subroutine that returns a value to the calling routine.

The AreaCalc function is designed to calculate the area of a rectangle and return it to the calling routine where it can be used in further calculations.

```
FUNCTION AreaCalc(w, h)
  Area = w * h
  RETURN Area
END FUNCTION

Result = AreaCalc(5, 3)
OUTPUT Result
```

INSPECTION COPY



10

Functions

Remember a function is a subroutine that returns a value.

The AreaCalc function is designed to calculate the area of a rectangle and return it to the calling routine where it is stored in a variable.

```
FUNCTION AreaCalc(w, h)
    Area = w * h
    RETURN Area
END FUNCTION
Result = AreaCalc(5, 3)
OUTPUT Result
```

INSPECTION COPY

COPYRIGHT
PROTECTED

11

Functions

Remember a function is a subroutine that returns a value.

The AreaCalc function is designed to calculate the area of a rectangle and return it to the calling routine where it is stored in a variable.

```
FUNCTION AreaCalc(w, h)
    Area = w * h
    RETURN Area
END FUNCTION
Result = AreaCalc(5, 3)
OUTPUT Result
```

INSPECTION COPY

COPYRIGHT
PROTECTED

12

Variable Scope

There are two types of variables:

Global variables are accessible throughout the entire program, including inside subroutines.

Local variables are only accessible from within the subroutine where they were defined.

When a subroutine is called, a new instance of local variables is created; they are then destroyed when the subroutine returns.

INSPECTION COPY

COPYRIGHT
PROTECTED

1

Assembly Language

Please do not share digital resources; they may only be copied for the purchasing institution on a single site and for their own use.

INSPECTION COPY

COPYRIGHT
PROTECTED

2

Machine Code

Each type of CPU is designed to carry out a set of instructions. Each instruction is represented by a unique number. This is called machine code.

Machine Code	Assembly Language
0010	SUB
0001	ADD
0101	LDA

It is hard for humans to read and write machine code. A high-level programming language was developed to make it easier.

Assembly language features a set of mnemonics that correspond to machine code instructions.

INSPECTION COPY

COPYRIGHT
PROTECTED

3

Mnemonics

These are the mnemonics that are commonly used in assembly language.

Mnemonic	Description
ADD	Addition
SUB	Subtraction
STA	Store
LDA	Load
INP	Input
OUT	Output
HLT	End Program
DAT	Data location

The **Little Man Computer** is a simplified simulator used to teach students how the computer works.

INSPECTION COPY

COPYRIGHT
PROTECTED

4

Instruction Format

This is the standard format for writing instructions.

opcode: the operation code is a single instruction

operand: the value

ADD	Num1
MOV	Num1
SUB	Num1

INSPECTION COPY

COPYRIGHT PROTECTED



5

Example

Asks the user to input a value, and stores it in the accumulator.

INP

Stores the inputted number in the Num1 memory location.

STA Num1

Asks the user to input a value, and stores it in the accumulator.

INP

Adds the value in Num1 to the value in the accumulator.

ADD Num1

Outputs the contents of the accumulator.

OUT

Stops the program.

HLT

Reserves a memory location and labels it Num1.

Num1 DAT

COPYRIGHT PROTECTED



INSPECTION COPY

6

Branching

Branching is used to jump to different points in a program.

BRA	Always jump to a different point in the program.
BRZ	Branch to a different point in the program if the value in the accumulator is zero.
BRP	Branch to a different point in the program if the value in the accumulator is positive.

INSPECTION COPY

COPYRIGHT PROTECTED



7

Example

Here is an example program that

Label	Instruction	Operand	Description
	INP		The user input
	STA	Num1	The value in the Num1 memory location
	INP		The user input
	BRZ	numIsZero	The value in the accumulator jumps to the numIsZero label
	ADD	Num1	
numIsZero	OUT		The value in the accumulator
	HLT		The program ends
Num1	DAT		The label Num1

COPYRIGHT PROTECTED



INSPECTION COPY

8

Storage

Two different types of storage are used with instructions.

Registers	
Storage locations contained within the CPU which are much faster to access than memory	The Register File (RF) is a set of registers within the CPU.

INSPECTION COPY

COPYRIGHT PROTECTED



9

Addressing

There are different methods to access data in memory. These are called memory addressing methods.

Immediate	
The data is hard-coded into the instruction. It is the fastest method as it doesn't use memory.	The instruction register (IR) is a register within the CPU that holds the instruction being executed.
Indirect	
The address of the data is held in another location which needs to be looked up first.	A base register (BR) is a register within the CPU that holds the address of the data being accessed.

COPYRIGHT PROTECTED



INSPECTION COPY

1

Graph and Tree

A Level Only

Photoshop/digital resources may only be copied by the purchasing institution on a single site and for their own use

INSPECTION COPY

COPYRIGHT PROTECTED

Zig Zag Education

2

Tree Traversal

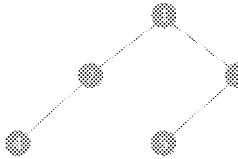
Traversal is the process of searching a tree once. There are two main types.

Depth-first Traversal

Starting at the root, each node in one branch is visited before exploring the next branch.

Breadth-first Traversal

Starting at the root, all nodes at the same level are visited before moving to the next level.



COPYRIGHT PROTECTED

Zig Zag Education

3

Pre-Order Traversal

One type of depth-first traversal is pre-order traversal.

Steps:

1. Start at the root node.

2. Explore the left sub-tree, working top-down.

3. Explore the right sub-tree, working top-down.

1 2 4 3 5 6

Uses include copying a tree and creating an expression tree.

INSPECTION COPY

COPYRIGHT PROTECTED

Zig Zag Education

4

Post-Order Traversal

One type of depth-first traversal is post-order traversal.

Steps:

1. Explore the left sub-tree, working bottom-up.

2. Explore the right sub-tree, working bottom-up.

3. Return to the root node.

4 2 5 6 3 1

Used for a binary search tree.

INSPECTION COPY

COPYRIGHT PROTECTED

Zig Zag Education

5

In-Order Traversal

Another type of traversal is in-order traversal.

Steps:

1. Explore the left sub-tree, working bottom-up.

2. Visit the root node.

3. Explore the right sub-tree, working bottom-up.

4 2 1 5 6 3

Uses include infix to Reverse Polish notation and creating an expression tree from an infix expression.

INSPECTION COPY

COPYRIGHT PROTECTED

Zig Zag Education

6

Graph Traversal

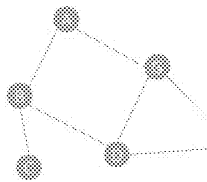
Like trees, there are two main types of graph traversal.

Depth-first Traversal

This algorithm uses a stack.

Breadth-first Traversal

This algorithm uses a queue.



COPYRIGHT PROTECTED

Zig Zag Education

INSPECTION COPY

COPYRIGHT PROTECTED

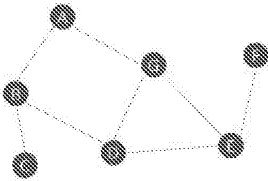
Zig Zag Education

7

Depth-First Tra

In depth-first traversal each vertex is marked when it is visited and added to the stack.

When there are no remaining unvisited neighbours, vertices are popped off the stack until one is reached that has unvisited neighbours.



INSPECTION COPY

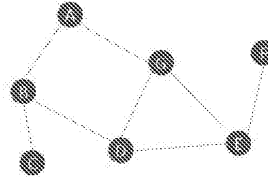
COPYRIGHT
PROTECTED
Zig
Education

8

Breadth-First Tr

In breadth-first traversal each vertex is marked when it is visited and added to the queue.

When there are no remaining unvisited neighbours for the current vertex, it moves to the next vertex in the queue.



INSPECTION COPY

COPYRIGHT
PROTECTED
Zig
Education

1

Searching Alg

A Level
Only

Photocopyable digital resources may only be copied by the purchasing institution on a single site and for their own use.

INSPECTION COPY

COPYRIGHT
PROTECTED
Zig
Education

2

Linear Search

Linear search is the simplest searching algorithm. It checks every element in the list and checks every element looking for. In this example we are looking for the number 4.

3	8	1	4	7	6	9
---	---	---	---	---	---	---

```

1 FUNCTION LSearch(List, ItemToFind)
2   FOR i = 0 to length(List)
3     IF List[i] == ItemToFind THEN
4       RETURN i
5   END IF
6   NEXT i
7   RETURN -1
8 END FUNCTION
  
```

The problem with this method is that it checks every element in the list.

INSPECTION COPY

COPYRIGHT
PROTECTED
Zig
Education

3

Binary Search

Binary search is a good way of searching lists. Each step it halves the number of items in the list. In this example we are going to search for the number 4.

0	1	2	4	5	6	8
---	---	---	---	---	---	---

```

1 FUNCTION BSearch(List, ItemToFind, Min, Max)
2   MidPoint = FindMidPoint(Min, Max)
3   IF List[MidPoint] < ItemToFind THEN
4     Bsearch(List, ItemToFind, MidPoint+1, Max)
5   ELSE IF List[MidPoint] > ItemToFind THEN
6     Bsearch(List, ItemToFind, Min, MidPoint-1)
7   ELSE
8     RETURN MidPoint
9   END IF
9 END FUNCTION
  
```

INSPECTION COPY

COPYRIGHT
PROTECTED
Zig
Education

4

Binary Search

Binary search is a good way of searching lists. Each step it halves the number of items in the list. In this example we are going to search for the number 8.

8

```

1 FUNCTION BSearch(List, ItemToFind, Min, Max)
2   MidPoint = FindMidPoint(Min, Max)
3   IF List[MidPoint] < ItemToFind THEN
4     Bsearch(List, ItemToFind, MidPoint+1, Max)
5   ELSE IF List[MidPoint] > ItemToFind THEN
6     Bsearch(List, ItemToFind, Min, MidPoint-1)
7   ELSE
8     RETURN MidPoint
9   END IF
9 END FUNCTION
  
```

INSPECTION COPY

COPYRIGHT
PROTECTED
Zig
Education

5 Binary Search

Binary search is a good way of searching lists. Each step it halves the number of items in the example we are going to search.

```
1 FUNCTION BSearch(List, ItemToFind, Min, Max)
2   MidPoint = FindMidPoint(Min, Max)
3   IF List[MidPoint] < ItemToFind THEN
4     Bsearch(List, ItemToFind, MidPoint+1, Max)
5   ELSE IF List[MidPoint] > ItemToFind THEN
6     Bsearch(List, ItemToFind, Min, MidPoint-1)
7   ELSE
8     RETURN MidPoint
9   END IF
10 END FUNCTION
```

INSPECTION COPY

COPYRIGHT
PROTECTED



6 Binary Search

Binary search is a good way of searching lists. Each step it halves the number of items in the example we are going to search.

```
1 FUNCTION BSearch(List, ItemToFind, Min, Max)
2   MidPoint = FindMidPoint(Min, Max)
3   IF List[MidPoint] < ItemToFind THEN
4     Bsearch(List, ItemToFind, MidPoint+1, Max)
5   ELSE IF List[MidPoint] > ItemToFind THEN
6     Bsearch(List, ItemToFind, Min, MidPoint-1)
7   ELSE
8     RETURN MidPoint
9   END IF
10 END FUNCTION
```

INSPECTION COPY

COPYRIGHT
PROTECTED



7 Binary Search

The drawback of the binary search method is that the list must be sorted. Even if a list is already sorted, it will still take time to add a new element as a new element is added.

The binary search tree can be used to get a list of values that can easily be added or removed.

Rules

7

The left subtree of a node contains values that are less than the value of the node.

The right subtree of a node contains values that are more than the value of the node.

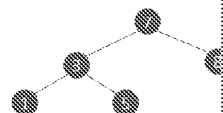
INSPECTION COPY

COPYRIGHT
PROTECTED



8 Using Binary Search

It is easy to find the highest and lowest values in a list.



Lowest Value 1

Searching for a particular value in a binary search tree. From each node we simply move either right or left depending on whether its value is higher or lower than the value we are searching for. Let's search for the value 1.

INSPECTION COPY

COPYRIGHT
PROTECTED



Sorting Algorithms

A Level
Only

Photocopiable digital resources may only be used by the purchasing institution on a single site and for their own use.

INSPECTION COPY

COPYRIGHT
PROTECTED



2 Bubble Sort

The bubble sort algorithm works through a list of numbers and swapping them if necessary.

3 8 1 4 7 6

Variable	Value
swapActive	false
i	0
temp	

```
1 swapActive = false
2 WHILE swapActive = true
3   swapActive = false
4   FOR i = 0 TO List.Count - 2
5     IF List[i] > List[i+1] THEN
6       temp = List[i]
7       List[i] = List[i+1]
8       List[i+1] = temp
9     END IF
10  NEXT i
11 END WHILE
12 END
```

Bubble sort is simple to implement.

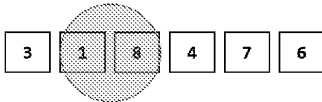
INSPECTION COPY

COPYRIGHT
PROTECTED



3 Bubble So

The bubble sort algorithm works through a list and swapping them if ne



Variable	Value
swapActive	true
i	1
temp	8

```
1 swap/
2 WHILE
3 swa
4 Fc
5
6
7
8
9
10
11 NI
12 END
```

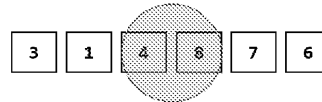
Bubble sort is simple to implement

INSPECTION COPY



4 Bubble So

The bubble sort algorithm works through a list and swapping them if ne



Variable	Value
swapActive	true
i	2
temp	8

```
1 swap/
2 WHILE
3 swa
4 Fc
5
6
7
8
9
10
11 NI
12 END
```

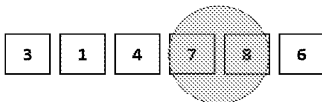
Bubble sort is simple to implement

INSPECTION COPY



5 Bubble So

The bubble sort algorithm works through a list and swapping them if ne



Variable	Value
swapActive	true
i	3
temp	8

```
1 swap/
2 WHILE
3 swa
4 Fc
5
6
7
8
9
10
11 NI
12 END
```

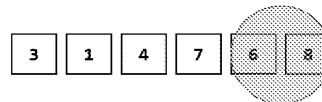
Bubble sort is simple to implement

INSPECTION COPY



6 Bubble So

The bubble sort algorithm works through a list and swapping them if ne



Variable	Value
swapActive	true
i	4
temp	8

```
1 swap/
2 WHILE
3 swa
4 Fc
5
6
7
8
9
10
11 NI
12 END
```

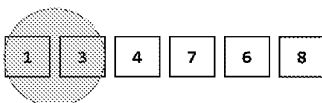
Bubble sort is simple to implement

INSPECTION COPY



7 Bubble So

The bubble sort algorithm works through a list and swapping them if ne



Variable	Value
swapActive	true
i	0
temp	3

```
1 swap/
2 WHILE
3 swa
4 Fc
5
6
7
8
9
10
11 NI
12 END
```

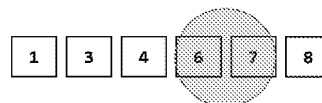
Bubble sort is simple to implement

INSPECTION COPY



8 Bubble So

The bubble sort algorithm works through a list and swapping them if ne



Variable	Value
swapActive	true
i	3
temp	7

```
1 swap/
2 WHILE
3 swa
4 Fc
5
6
7
8
9
10
11 NI
12 END
```

Bubble sort is simple to implement

INSPECTION COPY



9

Merge Sort

The merge sort algorithm works by splitting elements and then gradually merging them until they are all in one sorted list.

3	8	1	4	7	
---	---	---	---	---	--

INSPECTION COPY

COPYRIGHT
PROTECTED

10

Merge Sort

The merge sort algorithm works by splitting elements and then gradually merging them until they are all in one sorted list.

1	3	4	8
---	---	---	---

INSPECTION COPY

COPYRIGHT
PROTECTED

11

Merge Sort

The merge sort algorithm works by splitting elements and then gradually merging them until they are all in one sorted list.

1	2	3	4	5	
---	---	---	---	---	--

INSPECTION COPY

COPYRIGHT
PROTECTED

12

Insertion Sort

The insertion sort algorithm uses two lists: one for sorted elements and one for unsorted elements.

Elements are gradually moved from the unsorted list to the sorted list until they are in the correct position.

Sorted	Unsorted
3	8 1 4

INSPECTION COPY

COPYRIGHT
PROTECTED

13

Insertion Sort

The insertion sort algorithm uses two lists: one for sorted elements and one for unsorted elements.

Elements are gradually moved from the unsorted list to the sorted list until they are in the correct position.

Sorted	Unsorted
3 8	1 4

INSPECTION COPY

COPYRIGHT
PROTECTED

14

Insertion Sort

The insertion sort algorithm uses two lists: one for sorted elements and one for unsorted elements.

Elements are gradually moved from the unsorted list to the sorted list until they are in the correct position.

Sorted	Unsorted
1 3 8	4

INSPECTION COPY

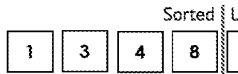
COPYRIGHT
PROTECTED

15

Insertion Sort

The insertion sort algorithm uses two lists and one for unsorted elements.

Elements are gradually moved from the unsorted list to the sorted list.



COPYRIGHT
PROTECTED

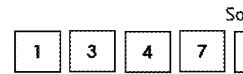


16

Insertion Sort

The insertion sort algorithm uses two lists and one for unsorted elements.

Elements are gradually moved from the unsorted list to the sorted list.



COPYRIGHT
PROTECTED

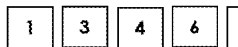


17

Insertion Sort

The insertion sort algorithm uses two lists and one for unsorted elements.

Elements are gradually moved from the unsorted list to the sorted list.



COPYRIGHT
PROTECTED

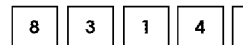


18

Quick Sort

One element in the list is selected as a pivot and elements are placed on the left of it and the right of it.

This process is then repeated for the sublists until the entire list has been a pivot.



COPYRIGHT
PROTECTED

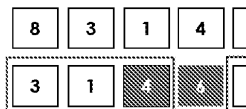


19

Quick Sort

One element in the list is selected as a pivot and elements are placed on the left of it and the right of it.

This process is then repeated for the sublists until the entire list has been a pivot.



COPYRIGHT
PROTECTED

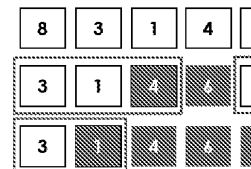


20

Quick Sort

One element in the list is selected as a pivot and elements are placed on the left of it and the right of it.

This process is then repeated for the sublists until the entire list has been a pivot.



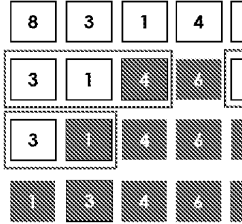
COPYRIGHT
PROTECTED



Quick Sort

One the elements in the list is selected as elements are placed on the left of it and then to the right.

This process is then repeated for the sublists until the entire list has been a pivot.



INSPECTION COPY

COPYRIGHT
PROTECTED



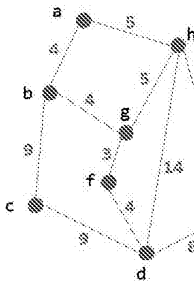
INSPECTION COPY

COPYRIGHT
PROTECTED



2 Dijkstra's Algo

We use Dijkstra's shortest path algorithm to find the shortest path between vertices in a weighted graph.



INSPECTION COPY

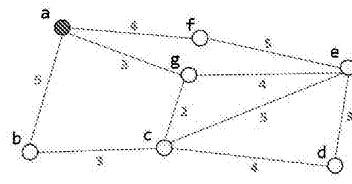
COPYRIGHT
PROTECTED



3 Worked Example

We start by finding the distance from the starting vertex to its neighbours.

Next we repeat the process with the vertex that is closest to the starting vertex.



INSPECTION COPY

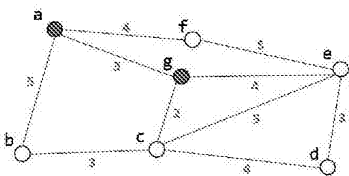
COPYRIGHT
PROTECTED



4 Worked Example

We start by finding the distance from the starting vertex to its neighbours.

Next we repeat the process with the vertex that is closest to the starting vertex.



INSPECTION COPY

COPYRIGHT
PROTECTED

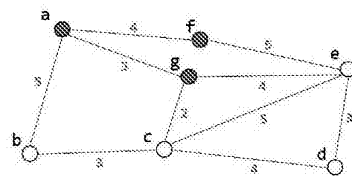


5 Worked Example

We start by finding the distance from the starting vertex to its neighbours.

Next we repeat the process with the vertex that is closest to the starting vertex.

The distance is only updated if a shorter route is found.



INSPECTION COPY

COPYRIGHT
PROTECTED

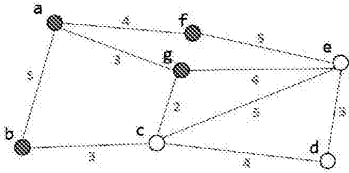


6 Worked Example

We start by finding the distance from the starting vertex to its neighbours.

Next we repeat the process with the vertex closest to the starting vertex.

The distance is only updated if a shorter route is found.



INSPECTION COPY

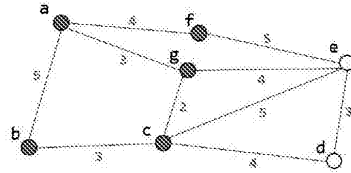
COPYRIGHT
PROTECTED
Zig
Zag
Education

7 Worked Example

We start by finding the distance from the starting vertex to its neighbours.

Next we repeat the process with the vertex closest to the starting vertex.

The distance is only updated if a shorter route is found.



INSPECTION COPY

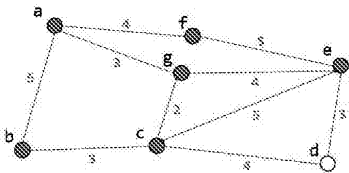
COPYRIGHT
PROTECTED
Zig
Zag
Education

8 Worked Example

We start by finding the distance from the starting vertex to its neighbours.

Next we repeat the process with the vertex closest to the starting vertex.

The distance is only updated if a shorter route is found.



INSPECTION COPY

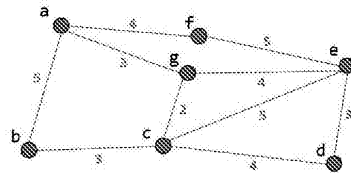
COPYRIGHT
PROTECTED
Zig
Zag
Education

9 Worked Example

We start by finding the distance from the starting vertex to its neighbours.

Next we repeat the process with the vertex closest to the starting vertex.

The distance is only updated if a shorter route is found.



INSPECTION COPY

COPYRIGHT
PROTECTED
Zig
Zag
Education

10 Common Uses

There are a number of uses for the shortest path algorithm.

Finding the shortest route to drive between two addresses.

Finding the shortest route for packets to take between two devices in a network.

INSPECTION COPY

COPYRIGHT
PROTECTED
Zig
Zag
Education

11 A* Algorithm

The A* algorithm is an alternative to Dijkstra's algorithm.

It uses heuristics to estimate the distance to the end node.

The use of heuristics makes it more efficient than Dijkstra's path algorithm.

It is commonly used in games to allow characters to find a path through a virtual world.

INSPECTION COPY

COPYRIGHT
PROTECTED
Zig
Zag
Education

PageRank Alg

A Level
Only

Photocopying digital resources may only be copied by the purchasing institution on a single site and for their own use.

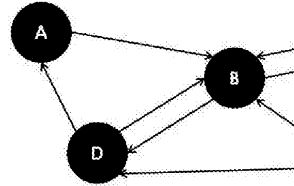
INSPECTION COPY

COPYRIGHT
PROTECTED
Zig
Zag
Education

PageRank

PageRank is the algorithm used by search engines to rank webpages and therefore where they appear in search results.

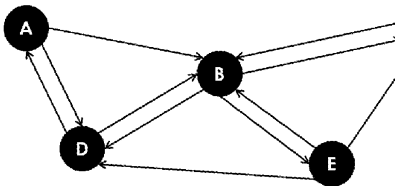
This graph represents a simple online voting system. The arrows represent links in the system.



COPYRIGHT
PROTECTED
Zig
Zag
Education

Voting

The algorithm uses the number of inbound links to calculate its rank. Each inbound link is considered a 'vote' for the importance of the page.

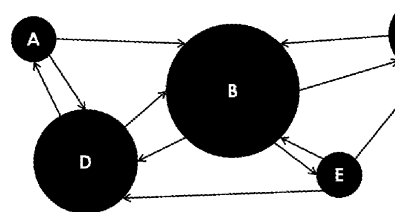


INSPECTION COPY

COPYRIGHT
PROTECTED
Zig
Zag
Education

Voting

The algorithm uses the number of inbound links to calculate its rank. Each inbound link is considered a 'vote' for the importance of the page.



INSPECTION COPY

COPYRIGHT
PROTECTED
Zig
Zag
Education

Weighting

In the PageRank algorithm not all votes are equal. Pages with more inbound links are given a greater weight.

This is based on the number of inbound links. The more inbound links, the greater the weight.

INSPECTION COPY

COPYRIGHT
PROTECTED
Zig
Zag
Education

Damping Factor

It is unlikely that the user would continue to click on links forever. This is factored in using the damping factor.

The damping factor is a value between 0 and 1, representing the probability of the user continuing to click on links.

INSPECTION COPY

COPYRIGHT
PROTECTED
Zig
Zag
Education

1

Big O Notation

A Level Only

Photoshop/digital resources may only be copied by the purchasing institution on a single site and for their own use

INSPECTION COPY

COPYRIGHT PROTECTED

Zig Zag Education

2

Big O Notation

Big O notation is used to describe how the algorithm grow in relation to the number

This allows algorithms to be compared complexity.

Big O looks at how long an algorithm takes

An O is used as a prefix for all expressions

n is used to refer to the num

COPYRIGHT PROTECTED

Zig Zag Education

INSPECTION COPY

COPYRIGHT PROTECTED

Zig Zag Education

3

Constant Complexity

The time complexity remains the same regardless of the number of items.

O(1)

This graph illustrates constant complexity.

For example, finding the first item in a list has to be evaluated regardless of the number of items.

Time to Complete

COPYRIGHT PROTECTED

Zig Zag Education

INSPECTION COPY

COPYRIGHT PROTECTED

Zig Zag Education

4

Logarithmic Complexity

A logarithm is the inverse of an exponential function.

The increase in time complexity decreases as the number of items increases.

O(log(n))

This graph illustrates logarithmic complexity.

Examples of algorithms with logarithmic complexity are binary search and binary tree traversal.

Time to Complete

COPYRIGHT PROTECTED

Zig Zag Education

INSPECTION COPY

COPYRIGHT PROTECTED

Zig Zag Education

5

Linear Complexity

The time complexity is proportional to the number of items.

O(n)

This graph illustrates linear complexity.

Linear search is an example of linear complexity as in a list has to be evaluated to find the item.

Time to Complete

COPYRIGHT PROTECTED

Zig Zag Education

INSPECTION COPY

COPYRIGHT PROTECTED

Zig Zag Education

6

Polynomial Complexity

The rate at which time complexity increases as the number of items gets larger.

O(n^k)

k is a constant value.

This graph illustrates polynomial complexity.

Bubble sort is an example of an algorithm with polynomial complexity.

Time to Complete

COPYRIGHT PROTECTED

Zig Zag Education

INSPECTION COPY

COPYRIGHT PROTECTED

Zig Zag Education

Exponential Complexity

The time complexity increases exponentially as the number of items gets larger.

$$O(k^n)$$

This graph illustrates exponential complexity.



The **travelling salesman problem** has exponential complexity. It involves a salesman wanting to find the shortest route that allows him to visit every city only once before returning to the starting point.

INSPECTION COPY

COPYRIGHT
PROTECTED



Order of Complexity

This is the order of complexity from best to worst.

Complexity	Big O Notation
Constant	$O(1)$
Logarithmic	$O(\log(n))$
Linear	$O(n)$
Polynomial	$O(n^k)$
Exponential	$O(k^n)$

COPYRIGHT
PROTECTED



INSPECTION COPY

Examples

Algorithm	Big O Notation
Linear Search	$O(n)$
Bubble Sort	$O(n^2)$
Binary Search	$O(\log(n))$
Binary Tree Search	$O(\log(n))$
Merge Sort	$O(n \log(n))$

INSPECTION COPY

COPYRIGHT
PROTECTED

