

PAPER 2 REVISION GUIDE

Eduqas Computer Science 9-1

0

30% OF OVERALL MARK

Greenfoot, HTML, Assembly Language,
Algorithms

David Woods with some copy and
paste from www.greenfoot.org,
www.teach-ict.com, Mark Clarkson

Table of Contents

GREENFOOT

..... 2 HTML.....

15	ASSEMBLY LANGUAGE
19	SEARCHING ALGORITHM
MS	
	21
		SORTING
	ALGORITHMS
	25
	ALGORITHMS
	AND PSEUDOCODE
	
	32
		FLOWCHARTS.....
	
	35

GREENFOOT

Definitions

Class- stands for the general concept of something in Greenfoot e.g. The class of Wombat defines all Wombats

Object- We can create objects from a class. E.g. each separate Wombat will be an object (or instance)

Method- An operation that an object can do. I.e. a bit of code that makes the object work.

How methods work in the program

```
public void act()
{
    checkKeypress();
    move(5);
    lookForWorm();
    switchImage();
}
/**
 * Check whether a control key on the keyboard has been pressed.
 * If it has, react accordingly.
 */
public void checkKeypress()
{
    if (Greenfoot.isKeyDown("left"))
    {
        turn(-4);
    }
    if (Greenfoot.isKeyDown("right"))
    {
        turn(4);
    }
}
/**
 * Check whether we have stumbled upon a worm.
 * If we have, eat it. If not, do nothing. If we have
 * eaten eight worms, we win.
 */
public void lookForWorm()
{
    if ( isTouching(Worm.class) )
    {
        removeTouching(Worm.class);
        Greenfoot.playSound("slurp.wav");

        wormsEaten = wormsEaten + 1;

        if (wormsEaten == 8)
        {
            Greenfoot.playSound("fanfare.wav");
            Greenfoot.stop();
        }
    }
}
```

The **act()** method. This is what the object does when play is pressed. In this case a crab...

- Checks for key press
- Moves
- Looks for worm
- Switches image

The act method calls the other methods defined below it.

The methods are defined down here.

You can see checkKeypress
lookForWorm

The others will be below.

Cases for defining methods, variables
classes/objects

Objects and Classes- Capitalise e.g. Worm ,Athlete, Crab, Lobster, Wombat, Tree

Variables –camelCase e.g. totalScore, wormsEaten, a, number, myWorm, x,y etc

Methods – camelCase e.g. checkKeyPress, act, turnAtEdge

Defining Methods

Can be accessed by other objects	Does not return anything	Name of method	Parameters anything in the brackets provides additional information to the method
public	void	checkKeyPress	();

Can be accessed by other objects	Does not return anything	Name of method	Needs an integer called amount
public	void	addAmount	(int amount);

Can be accessed by other objects	Returns True or False	Name of method	No extras needed
public	boolean	hasWings	();

Can be accessed by other objects	Does not return anything	Name of method	3 integers called x y z
public	void	compare	(int x,int y,int z);

Can not be accessed by other objects	Returns an integer	Name of method	No extras needed
private	int	showScore	();

3

Defining Variables

When defining variables define the type

Eg. **int age;** integer called age

age=12; the value 12 has been assigned to age

You can do it in one line

```
int age =12;
```

```
boolean isHungry =True;
```

```
int score;
```

```
int year=2014;
```

Variables can be the result of other variables

```
int a=12
```

```
int b=2
```

```
int sum =a+b;
```

The type of a variable can be a class/object

e.g. `Crab myCrab;`
`myCrab = newCrab();`

Or in one line

Type of variable is a class –Crab in this case. Note	Name of	Assign an object
variable	the capital letter	Crab() to the variable
Crab	myCrab	= newCrab();

4

Local Variables and instance (global) variables

If you define variable in a method then that is a local variable. It disappears outside the method. It belongs to the method.

You can define an instance variable which you declare inside the class but outside the method. Instance variables belong to the class and can be used again and again. Use the keyword `private` to define them.

```
public class Camel extends Actor
{
    private GreenfootImage image1;
    private GreenfootImage image2;
    private int age;
    /**
```

Instance variables. Defined in the class but not in the method. They use the keyword `private`.
Global variables

```
    * Act - do whatever the Camel wants to
do. This method is called whenever
    * the 'Act' or 'Run' button gets pressed
in the environment.
```

```
    */
```

```
public void act()
```

```
{
    boolean is Alive;
    int n;
}
```

```
}
```

Local and Global

Local variables.

Variables Advantages

Local variable only exist in the function or method. They disappear when the method is not used. Global variables exist throughout the whole program. It is best to use local variables wherever possible. Global variables waste memory because they use up a space in RAM from the beginning to the end of the program. It is more difficult to trace a problem in programs which use global variables.

Code – Movement

move(int distance); e.g. **move(5);** -moves in the way the object is facing bigger the number, faster the movement

turn(int angle); e.g. **turn(180);** -full turn

setLocation(x,y); go to a co-ordinate e.g.

1. **setLocation(120,200)** goes to this co-ordinate.

2. **setLocation(getX(),getY()+4)** goes to current x location, current y location +4 i.e. moves up on the y-axis

5

Code –Random Numbers

Greenfoot.getRandomNumber(integer) will get a random number between 0 and the integer

Code- Edge detection

`isAtEdge()` returns True or False

Code – Selection (if statements)

If object hits an edge

turn 17°

if a random number is picked out of a 100 that is less than 10 (10% of the time) turn -45°

```
public void  
checkEdge ()
```

```
{
```

```
    if (isAtEdge ())
```

```
    {
```

```
        turn (17) ;
```

```
    }
```

```
    if (Greenfoot.getRandomNumber (100) < 10)
```

```
    { turn (-45) ;
```

```
    }
```

Code – Selection (if-else statements)

(condition)

```
{  
    statements;  
}  
else  
{  
    statements;  
}
```

Code- Sound

```
Greenfoot.playSound("slurp.wav");
```

Code-Key Detection

isKeyDown("left")

```
public void checkKeys()
{
    if (Greenfoot.isKeyDown("left"))
    {
        turn(-4);
    }
    if (Greenfoot.isKeyDown("right"))
    {
        turn(4);
    }
}
```

Code – Add Objects to world automatically

Adds two camels to the world

```
public MyWorld()
```

```

{
// Create a new world with 600x400 cells
with a c
super(600, 400, 1);
Camel myCamel= new Camel();
addObject(myCamel, 250,250);
Camel myCamel2=new Camel();
addObject(myCamel2, 100,100);
}

```

Super(600,400,1) –size and resolution of world

Type of variable is Camel class. Note	Name of variable		A new Camel instance (object) gets assigned to the variable myCamel
the capital letter			
Camel	myCamel	=	new Camel();

addObject(myCamel,200,200) the variable myCamel is added to the world (remember it contains an object)

7

Code-Collision

Detection

Boolean isTouching(Class cls)

Check whether this act is touching objects of the given class

Example:

```
if (isTouching(Bee.class))
{
    removeTouching(Bee.class);
}
```

List getIntersectingObjects(Class cls)

Return all the objects that intersect this object

Actor getOneIntersectingObject(Class cls)

Return an object that intersect this object

Example:

```
{
    Actor ant;
    ant=getOneIntersectingObject(Ant.class);
    if (ant != null)
    {
        World world;
        world =getWorld();
        world.removeObject(ant);
    }
}
```

See `getOneObjectAtOffset` for explanation

List getObjectsAtOffset(int dx, in dy, Class cls)

Return all objects that intersect the given location (relative to this object's location)

Actor getOneObjectAtOffset(int dx, in dy, Class cls)

Return all objects that intersect the given location (relative to this object's location)

Example:

Type of variable Actor . variable name worm

```
Actor worm;
worm = getOneObjectAtOffset(0, 0, Worm.class);
if (worm != null)
{
    World world;
    world = getWorld();
    world.removeObject(worm);
}
```

Place into the variable 'worm'. Any worm class that intersects with the object at 0,0 (probably top corner of your object)

This is like a double negative if there is **not nothing** in the variable worm i.e. if there is something or if worm is not empty

Type of variable World . variable name world

Put the current world into the variable called world

Remove whatever object is in the worm variable from the world (i.e. a worm)

Code-Removing Objects

Removing yourself!

To remove an object itself i.e. within the code for an object. E.g. a bacteria cell floats to the edge of a world and you want it to disappear.

```
getWorld().removeobject(this);
```

This method belongs to the world object so you can't just use **remove object(this)**; You have to get access to the world object with **getWorld()**

```
if (isAtEdge())  
{  
    getWorld().removeObject(this);  
}
```

This removes my Bee when it hits the edge

Removing someone else!

```
removeTouching(Class cls);
```

```
if (isTouching(Ant.class))  
{  
    removeTouching(Ant.class);  
}
```

Removing someone else 2!

```
public void checkCollision()  
{  
    Actor bacteria;  
    bacteria=getOneIntersectingObject(Bacteria.class);  
    if (bacteria!=null)  
    {  
        getWorld().removeObject(bacteria);  
    }  
}
```

- Variable called bacteria type is Actor
- Put a bacteria.Class (object) into the variable bacteria when it intersects
- If bacteria is not empty (double negative)
- Remove the bacteria (You need get world as this is from the World class)

Code –stop

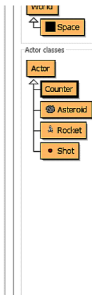
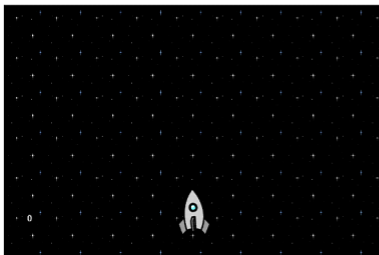
```
Greenfoot.stop();
```

9

Code –Add Text

```
import java.awt.Color; at the top (Underneath
import Greenfoot;)
setImage(new GreenfootImage(" Your Text",size,
Color.TEXTCOLOUR, Color.BGCOLOUR));
setImage(new GreenfootImage(" 0", 20,
Color.WHITE, Color.BLACK));
```

Code –Accessing One Object from Another



This asteroid game has a counter. You shoot asteroids and the score increases with every hit.

The screenshot shows a Java IDE window titled "tut-access-p1 - Java". The main editor displays the source code for the `Counter` class, which extends `Actor`. The code includes imports for `greenfoot.*` and `java.awt.Color`, a class comment, and the following methods:

```
import greenfoot.*; // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)
import java.awt.Color;

/**
 * Write a description of class Counter here.
 *
 * @author (your name)
 * @version (a version number or a date)
 */
public class Counter extends Actor
{
    private int totalCount = 0;

    public Counter()
    {
        setImage(new GreenfootImage("0", 20, Color.WHITE, Color.BLACK));
    }

    /**
     * Increase the total amount displayed on the counter, by a given amount.
     */
    public void bumpCount(int amount)
    {
        totalCount += amount;
        setImage(new GreenfootImage("" + totalCount, 20, Color.WHITE, Color.BLACK));
    }
}
```

At the bottom of the IDE, a status bar indicates "Class compiled - no syntax errors" and a "saved" button is visible. On the right side, a class hierarchy diagram shows "World classes" (World, Space) and "Actor classes" (Actor, Counter, Asteroid, Rocket, Shot). An arrow points from the `Counter` class in the diagram to the `bumpCount` method in the code editor.

The method `bumpCount()` increases the counter score

10

It would make sense to call the method `bumpCount()` from the shot object. I.e. when the shot hits

the asteroid.

```
void hitAnAsteroid()  
{  
    bumpCount();  
}
```

THIS WILL NOT WORK

Cannot find symbol -
method bumpCount()

The solution to let objects interact with each other i.e. the shot object with the counter object is.....

Store a reference to the counter in the world, then retrieve it from the shot when we need to.

1.

Storing a reference to the counter in the world

This is the world

```
private Counter theCounter;  
  
public Space()  
{  
    super(600, 400, 1);  
    addObject(new Rocket(), 300, 200);  
    theCounter = new Counter();  
    addObject(theCounter, 5, 5);  
}
```

declare a global variable to store the reference to the Counter. Must be declared outside the code for space world **but inside the class**

assign global variable (the Counter) a counter in the world

Place it in the world

2.

Add a method to the world to retrieve the value of "theCounter" so it can be accessed by the the other object (shot in this case)

Still in the world

```
public Counter getCounter()  
{  
    return theCounter;  
}
```

This will allow the object to get access it will call the getCounter method on the world to obtain a reference to the counter.

11

3.

Now to call the bumpCount method on the counter reference from the object.

We are now in the shot class **not** the world

```
void hitAnAsteroid()  
{  
    Space spaceWorld = (Space) getWorld(); //  
    Counter counter = spaceWorld.getCounter(); //  
    counter.bumpCount(5);  
}
```

Get a reference to the world. Put it in a variable (spaceWorld in this case)

Get a reference to the counter. Put into to a variable called counter use getCounter() which you put in the world

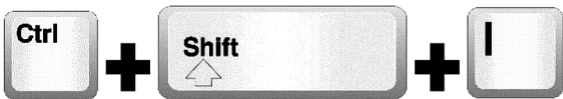
Now you can call the bumpCount method. Which will increase the counter. The counter will be increased by 5 in this method call

Greenfoot Helpful

Shortcuts

Indentation

Indents do not affect the running of the program but they make it easier to understand



Will sort out the indentation

Code completion

Start to type some code and press



```
lic Counter()
```

```
set(new GreenfootImage("0", 20, Color.WHITE, Color.BLACK));
```

```
Incr
```

```
lic
```

```
tot
```

```
set
```

```
void setImage(GreenfootImage)
```

```
void setLocation(int, int)
```

```
void setRotation(int)
```

greenfoot.Actor

```
void setImage(GreenfootImage)
```

Set the image for this actor to the specified image.

Parameters

image - The image.

see - #setImage(String)

12

Greenfoot –Common Errors

1. The classic error! ‘reached end of file while parsing’ You have missed the last }

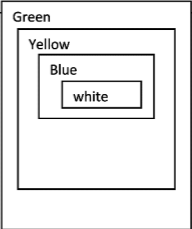

```
public class Counter extends Actor
{
    private int totalCount = 0;

    public Counter()
    {
        set(new GreenfootImage("0", 20, Co
    }

    /**
     * Increase the total amount displayed
     */
    public void bumpCount(int amount)
    {
        totalCount += amount;
        setImage(new GreenfootImage("" + t
    }
}
```

reached end of file while parsing

To solve this add the } at the end. Press ctrl+shift+i then check there is geen x with yellow then blue then white inside.



2. ; expected.

```
Counter()
set(new GreenfootImage("0", 20,
Color.WHITE, Color.BLACK))
```

;' expected

Increase the total amount displayed on the counter, by a given amount. The code in the white sections (in the methods) should have ; at the end of each line

3. Cannot find symbol method.....

```
Ground ground;  
ground = (Ground) getWorld();  
ground.removeObject (Gate);  
  
Greenfoot.plaSound("pop.wav");  
Counter co  
counter.bumpCounter (1);
```

cannot find symbol - method plaSound(java.lang.String)

The code has been spelled wrong or you have missed a dot.

14

HTML

HyperText Markup Language displays and formats content on a webpage

Tags

```
<html>... </html>
```

```
<head>...</head>
```

```
<body>...</body>
```

```
<html>
  <head>
    <title>website title</title>
  </head>
  <body>
    content of website ...
  </body>
</html>
```

```
<h?>...</h?> -heading
```

```
<html>
<body>

<h1>This is heading 1</h1>
<h2>This is heading 2</h2>
<h3>This is heading 3</h3>
<h4>This is heading 4</h4>
<h5>This is heading 5</h5>
<h6>This is heading 6</h6>

</body>
</html>
```

This is heading 1

This is heading 2

This is heading 3

This is heading 4

This is heading 5

This is heading 6

<p>...</p>

```
<!DOCTYPE html>
<html>
<body>
```

```
<p>This is a paragraph.</p>
<p>This is a paragraph.</p>
<p>This is a paragraph.</p>
```

```
</body>
</html>
```

This is a paragraph.

This is a paragraph.

This is a paragraph.

 ... -**bold**

<i> ... </i> -**italic**

<u> ... </u> -**underlined**

```
<p><b>This text is bold</b></p>
<p><i>This text is italic</i></p>
<p><u>This text is underlined</u></p>
```

This text is bold

This text is italic

This text is underlined

<blockquote> ... </blockquote>

```
<h1>About WWF</h1>
<p>Here is a quote from WWF's
website:</p>
```

```
<blockquote>
For 50 years, WWF has been protecting
the future of nature. The world's
leading conservation organization, WWF
works in 100 countries and is supported
by 1.2 million members in the United
States and close to 5 million globally.
</blockquote>
```

About WWF

Here is a quote from WWF's website:

For 50 years, WWF has been protecting the future of nature. The world's leading conservation organization, WWF works in 100 countries and is supported by 1.2 million members in the United States and close to 5 million globally.

` link text ` -anchor/link

```
<html>
<body>

<a href="http://www.w3schools.com/html/">Visit our HTML tutorial</a>

</body>
</html>
```

[Visit our HTML tutorial](#)

`<hr>` -horizontal rule

```
<html>
<body>

<h1>HTML</h1>
<p>HTML is a language for describing web pages.</p>

<hr>

<h1>CSS</h1>
<p>CSS defines how to display HTML elements.</p>

</body>
</html>
```

HTML

HTML is a language for describing web pages.

CSS

CSS defines how to display HTML elements.

` ... ` -unordered list

` ... ` -list item

```
<h4>An Unordered List:</h4>
<ul>
  <li>Coffee</li>
  <li>Tea</li>
  <li>Milk</li>
</ul>
```

An Unordered List:

- Coffee
- Tea
- Milk

 -Line break

```
<p>  
To break lines<br>in a text,<br>use the br element.  
</p>
```

To break lines
in a text,
use the br element.

17

<center> ...</center> -Centre text

```
<!DOCTYPE html>  
<html>  
<body>  
  
<p>This is some text.</p>  
<center>This text will be center-aligned.</center>  
<p>This is some text.</p>  
  
<p>The center element is not supported in HTML5. Use  
CSS instead.</p>
```

This is some text.

This text will be center-aligned.

This is some text.

The center element is not supported in HTML5. Use
CSS instead.

-image

```

```



18

This is the mnemonic language that can be used to program the CPU. It is often simulated with the Little Man Computer

Name	Description
HLT	Stop (Little Man has a rest).
ADD	Add the contents of the memory address to the Accumulator
SUB	Subtract the contents of the memory address from the Accumulator
STA	Store the value in the Accumulator in the memory address given.
LDA	Load the Accumulator with the contents of the memory address given
BRA	Branch - use the address given as the address of the next instruction
BRZ	Branch to the address given if the Accumulator is zero
BRP	Branch to the address given if the Accumulator is zero or positive
INP	Input take from Input
OUT	Output. copy to Output
DAT	Used to indicate a location that contains data.

Assembly Language Code

```

start LDA zero
STA sum
STA index
INP
BRZ end
STA value
loop LDA sum
ADD value
STA sum
LDA index
ADD count
STA index
SUB value
BRZ endLoop
BRA loop
endLoop LDA sum
OUT
BRA start
end HLT

sum DAT 0
index DAT 0
count DAT 1
value DAT 0
zero DAT 0

0 LDA 23
1 STA 19
2 STA 20
3 INP
4 BRZ 18
5 STA 22
6 LDA 19
7 ADD 22
8 STA 19
9 LDA 20
10 ADD 21
11 STA 20
12 SUB 22
13 BRZ 15
14 BRA 6
15 LDA 19
16 OUT
17 BRA 0
18 HLT

19 DAT 0
20 DAT 0
21 DAT 1
22 DAT 0
23 DAT 0
  
```

Little Man Computer

OUTPUT: 64

Program counter: 64

CPU:
 0 PROGRAM COUNTER
 5 INSTRUCTION REGISTER
 ADDRESS REGISTER: 23
 ACCUMULATOR: 0

RAM (Memory addresses 0-99):

0	1	2	3	4	5	6	7	8	9
523	319	320	901	718	322	519	122	319	520
10	11	12	13	14	15	16	17	18	19
121	320	222	715	606	519	902	600	0	0
20	21	22	23	24	25	2	27	28	29
0	1	0	0	0	0	0	0	0	0
0	31	32	33	34	35	36	37	38	39
0	0	0	0	0	0	0	0	0	0
40	41	42	43	44	4				
0	0	0	0	0	0	0	0	0	0
50	51	52	53	54	55	56	57	58	59
0	0	0	0	0	0	0	0	0	0
60	61	62	63	64	65	66	67	68	69
0	0	0	0	0	0	0	0	0	0
70	71						78	79	
0	0	0	0	0	0	0	0	0	0
80	81	82	83	84	85	86	87	88	89
0	0	0	0	0	0	0	0	0	0
90	91	92	9						
0	0	0	0	0	0	0	0	0	0

INPUT: 8

Annotations:
 - **Assembly language program**: Points to the assembly code.
 - **Data**: Points to the data section.
 - **Assembled program**: Points to RAM address 50.
 - **Memory addresses 50 - 59**: Points to RAM addresses 50-59.
 - **The current instruction explained**: Points to RAM address 23.
 - **5=LOAD into accumulator the contents of RAM address 23**: Explains the instruction at RAM address 5.

Buttons: ASSEMBLE CODE INTO RAM, RUN, STEP, RESET, LOAD, SAVE

Footer: ©OCSE.computing.org.uk

Example programs

```
    INP
    STA 99
    INP
    ADD 99
    OUT
    HLT
// Output the sum of two
numbers
```

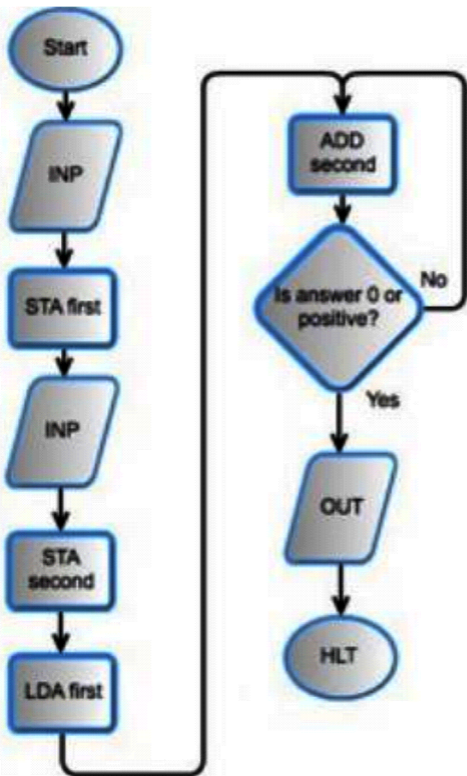


```
    INP
    STA FIRST
    INP
    ADD FIRST
    OUT
    INP
    SUB FIRST
    OUT
    HLT
FIRST  DAI
// Input three numbers.
// Output the sum of the
first two
// and the third minus the
first
```

Using BRA

By combining a BRA (break always) with a BRP or BRZ you can create a loop.

In this program we will take a negative number (e.g. -7) and keep adding a second number (e.g. 2) until it gets to 0 or a positive number:



Message Box:

INP

STA first

INP

STA second

LDA first

looptop ADD second

BRP done

BRA looptop

done OUT

HLT

first DAT

second DAT

SEARCHING ALGORITHMS

Linear searching

This is the simplest kind of searching. It is also called the linear search or sequential search. Searching starts with the first item and then moves to each item in turn until either a match is found or the search reaches the end of the data set with no match found. A criteria is set up before the search begins. e.g. "find the address of customer no 1344" This criteria will allow a possible match to be found within the records / items stored. If no match is found, then the process will return the appropriate message.

Serial searching algorithm

Set up the search criteria

Examine first item in the data set

If there is a match, end the procedure and return the result with 'match found'

If no match is found repeat

with the next item

If the last item is reached
and no match is found return
'match not found'.

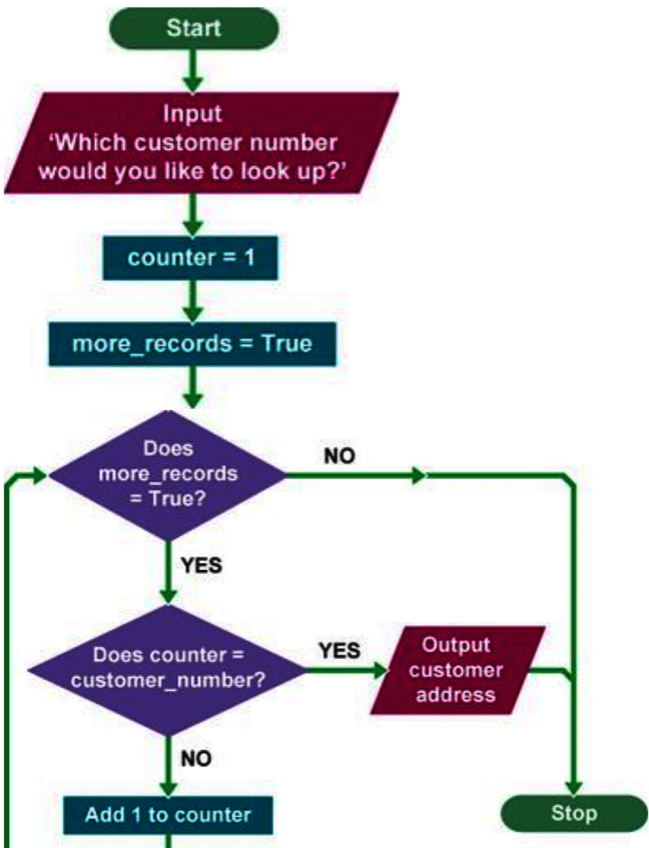
Advantages


Serial search is fairly simple to code. For example the pseudo-code below shows the algorithm in action

```
For i = 0 to 19
    check_for_match(i, list)
    if match_found return 'match found'
Next i
return 'no match found'
```

The subroutine starts with a loop going over a 20 element list / array. Each item is checked until either a match is found or the loop ends and the 'return 'no match found' is reached.

In Flowchart form





Good performance over small to medium lists. Computers are now very powerful and so checking potentially every element in the list for a match may not be an issue with lists of moderate length.

The list does not need to be in any order. Other algorithms only work because they assume that the list is ordered in a certain way. Serial searching makes no assumption at all about the list so it will work just as well with a randomly arranged list as an ordered list.

Not affected by insertions and deletions. Some algorithms assume the list is ordered in a certain way. So if an item is inserted or deleted, the computer will need to re-order the list before that algorithm can be applied. The overhead of doing this may actually mean that serial searching performs better than other methods.

Disadvantages

May be too slow over large lists. Computers take a finite amount of time to search each item, So naturally, the longer the list, the longer it will take to search using the serial method. The worst case being no match found and every item had to be checked.

This speed disadvantage is why other search methods have been developed.

22

Binary search

This is a fast method of searching for an item in a sorted / ordered list.

Sometimes, you may be doing a binary search without realising it.

Example

You want to find Samuel Jones in the local telephone

book. Would you start at page 1 and then go on from there, page by page? Unlikely.

You don't do this because you know an important fact about telephone books - the entries are in alphabetic order. So what you do is make a guess - J is about halfway down the alphabet and so you open the telephone book around half way. The page you see has names starting with N. So you know J will be in the first half of the book. Next you open a page about halfway down the first half the page has 'H'. So now Jones must be in the upper half of this section. You are carrying out a 'Binary search' algorithm. Notice that after only two guesses you are getting much closer to the answer. If you were carrying out a serial search, you would still be at page 2.

Binary Search algorithm

```
Set the highest location in the list to be searched as N
Set the lowest location in the list to be searched as L
Examine the item at location  $(N - L) / 2$  (i.e. halfway)
Is it a match?
if Yes End search.
```

No

Is item less than criteria ?

If Yes, Set lower limit L to item + 1 (Force the next search to use the upper half)

If No, Set upper limit N to item - 1 (Force the next search to use the lower half)

Is lower limit = upper limit, if yes end search (no match found)

Repeat from step 3 with the new upper and lower bounds.

Is Binary searching better than linear searching?

It depends.

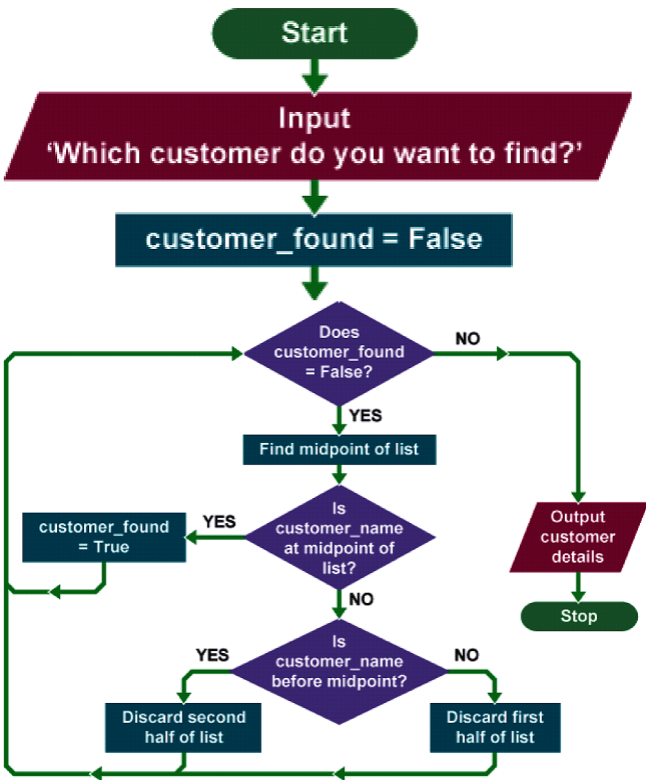
1. If the list is large and changing often, with items constantly being added or deleted, then the time it takes to constantly re-order the list to allow for a binary search might be longer than a simple serial search in the first place.
2. If the list is large and static e.g. telephone number database, then a binary search is very fast compared to linear search. (in maths terms it takes $2\log_2(n)$ for a binary search over n items)
3. If the list is small then it might be simpler to just use a linear search

4. If the list is random, then linear is the only way
5. If the list is skewed so that the most often searched items are placed at the beginning, then on average, a linear search might be better.

23

```
OUTPUT "Which customer do you want to  
find?"  
INPUT user inputs John Smith  
STORE the user's input in the customer  
name variable customer found = False
```

```
(we need to create a flag that  
identifies if the customer is found)  
WHILE customer found = False:  
    Find midpoint of list  
    IF customer name = record at  
midpoint of list THEN customer found =  
True  
    ELSE IF customer comes before  
the midpoint THEN  
    throw away the second half of the list  
    ELSE  
    throw away the first part of the list  
OUTPUT customer details
```



SORTING ALGORITHMS

Bubble Sort

A bubble sort is a very simple algorithm used to sort a list of numerical data into ascending or descending order.

The algorithm works its way through the list, making comparisons between a pair of adjacent items. Any items found to be in the wrong order are then exchanged. It keeps doing this over and over until all items in the list are eventually sorted into the correct order.

Step by step example

The following list of data (9, 23, 2, 5, 34, 56) needs to be put into ascending order using a bubble sort.

Original set

Step 1

No swap needed

9 23 2 5 34 56

9 23 2 5 34 56

9 23 2 5 34 56

Step 1: Compare the first two items in the list, 9 and 23.

As 9 is smaller than 23 they are in the correct order (ascending) so no action needs to be taken.

Step 2

9 23 2 5 34 56

Swap

9 2 23 5 34 56

Step 2: Move forward by one position and compare the next two numbers in the list - numbers 23 and 2

23 is larger than 2 so the bubble sort will swap the position of those two items.

Step 3

9 2 23 5 34 56

Swap

9 2 5 23 34 56

Step 3: Move forward by one position and compare the next two numbers in the list - numbers 23 and 5. 23 is greater than 5 so they are swapped.

Step 4

9 2 5 23 34 56

No swap needed

9 2 5 23 34 56

Step 4: Move forward by one position and compare the next two numbers in the list - numbers 23 and 34
23 is not greater than 34 so do NOT swap their position.

Step 5

No swap needed

© teach-ict.com

9 2 5 23 34 56

9 2 5 23 34 56

End of 1st Pass

Step 5: Move forward by one position and compare the next two numbers in the list - numbers 34 and 56
34 is not greater than 56 so do not swap.

26

This is the end of the first pass.

You can see from the final image above that the data set is not quite sorted in ascending order so the process is repeated once again, starting at the beginning of the list.

The algorithm is complete when it finishes a pass

without having to perform any swaps.

Bubble Sort pseudocode

The pseudocode below is for the ascending order algorithm

```
data_set = [9,2,5,23,34,56]
last_exam_position = data
set.length - 2
swap = true
WHILE swap == true
    swap = false
    FOR i = 0 to last_exam_position
        IF data_set[i] > data_set[i
+1] THEN
            temp = data_set[i+1]
            data_set[i+1] = data_set[i]
            data_set[i] = temp
            swap = true
        END IF
    NEXT i
END WHILE
```

```
PRINT "List is now in ascending
order."
```

And for completeness sake, a small adjustment to the pseudocode will sort the list in *descending* order, the only line that changes is the one in red shown below, where the 'greater than' becomes 'less than'

```
data_set = [9,2,5,23,34,56]
last_exam_position = data_set.length - 2
swap = true

WHILE swap == true
  swap = false
  FOR i = 0 to last_exam_position
    IF data set[i] < data set[i +1]
THEN
    -
    -
temp = data_set[i+1]
data_set[i+1] = data_set[i]
data_set[i] = temp // the swap is
complete

    swap = true
  END IF
NEXT i
```

```
END WHILE
```

```
PRINT "List is now in descending order."
```

. Bubble Sort pros and cons

27

Advantages

Simple to write the code for.

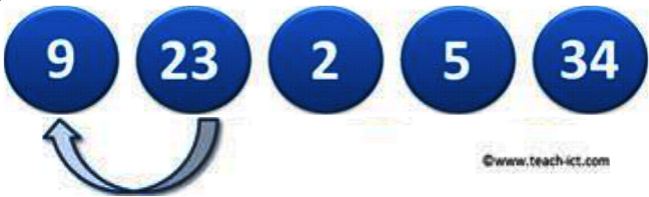
Simple to understand.

The data is sorted in the same memory location that it is held, so you don't need much extra memory to run the algorithm.

Disadvantages

One of the slowest ways to sort a list. For example, if the list becomes ten times larger than before, it takes almost a hundred times longer to sort. So this method of sorting is very sensitive to the length of the list.

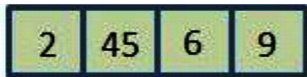




Merge Sort

Although bubble and insertion sorts work well on small lists of data, they are inefficient at sorting much larger lists.

The merge sort was developed to handle the sorting of large lists. It does this by breaking them down into multiple smaller lists, quickly sorting them, and then merging them back together into one larger list i.e. it is faster to sort these two lists then merge them back together



15	12	7	8
----	----	---	---

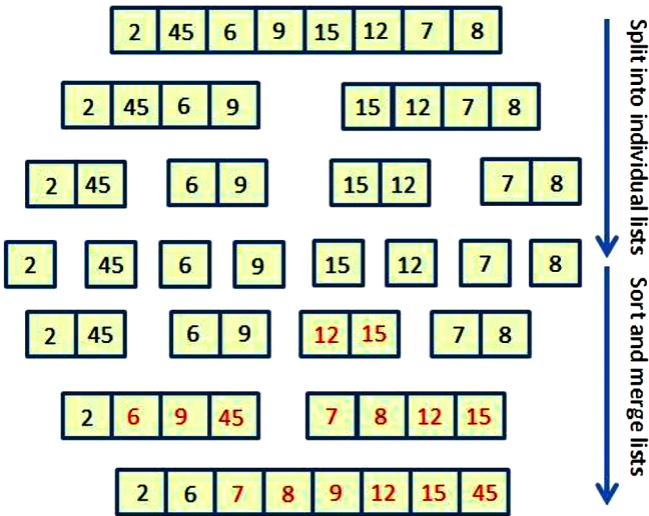
than to sort the list in its entirety.

Merge sort is an example of a '**divide-and-conquer**' algorithm because it splits down a larger problem into a number of smaller ones which are then solved. Each solution is then combined in some way to solve the larger problem.

Let's take a quick look at an example of a merge sort and then on the next page we will break it down into its different stages to help you understand what is happening, and why:

28

UNSORTED LIST



© teach-ict.com

SORTED LIST

The diagram above showing the first stage of the merge sort is to keep on splitting the lists until they are only 1 item long. Then each list is first sorted and then re-combined to form a fully sorted final list. We start off with a list of unordered numbers which we want to sort in ascending order:

2	45	6	9	15	12	7	8
---	----	---	---	----	----	---	---

©www.teach-ict.com

Step 1: Divide the above list into two smaller lists:

2	45	6	9
---	----	---	---

15	12	7	8
----	----	---	---

©www.teach-ict.com

Had there been 9 numbers then we would have a list of 4 items and a list of 5 items.

Step 2: Divide these lists into smaller, equal sized lists (dealing with an odd number if necessary):

2	45
---	----

6	9
---	---

15

12

7

8

As a human, we can look at this pattern and know that there are only two items left, meaning we could theoretically stop here and begin to sort and merge the data.

However, it is difficult for a computer to "look ahead" like this. It works out to be more efficient to simply let the computer run to the end, splitting lists down to 1 or 0 items. So we continue splitting the list until there is only one item per list.

Step 3: Continue to split each list until there is only one item per list (or zero in some cases for uneven lists):

2

45

6

9

15

12

7

8

Step 4: Now that the list has been split as far as possible, we can begin to merge and sort the data items:



two adjacent lists are paired back together and they are sorted (for this example, in ascending order). The ones that changed are the 12,15 pair

Step 5: Two more adjacent lists are merged together and the items within each list are sorted, the red numbers changed position:



7

8

12

15

©www.teach-ict.com

Step 6: This process of merging and sorting lists continues until all of the individual lists are merged together and just one list remains. Within this list, all of the data items will be sorted into the correct order:

2

6

7

8

9

12

15

45

©www.teach-ict.com

Merge sort pros and cons

With a 'divide-and-conquer' algorithm, there is a lot of repeating the same steps in a similar way. In this case the algorithm says 'Keep on dividing the list' and 'keep on merging and sorting'.

The statement 'keep on ...' is a very common and powerful idea in computer programming. There is a word for it: '**recursion**'.

A *recursive procedure* is one that calls itself with slightly different arguments until a stop condition is met.

In order to code the merge sort algorithm efficiently, a recursive procedure is used that keeps on splitting the list and another one is used that keeps on merging and sorting the lists. The pseudocode for this is quite complicated and is unlikely to be asked for in an exam - but here it is.

Advantages

It is the fastest of three types of sort (bubble, insert, merge)

It is the best option to use for long lists of data (more than 1000 long)

Disadvantages

More complicated to code compared to bubble and insert

It may use twice the memory size of the list - depending on the way it is coded. This becomes important if the list is millions of items long.

ALGORITHMS AND PSEUDOCODE

"Code that resembles a programming language but that uses a less strict syntax to express an algorithm and is independent of any real programming language."

Below are some examples of how the given pseudocode can be implemented using Python.

Variable names and types

Pseudocode	Python
Note- you declare what data type the variable is <code>myVariable</code> <code>myVariable is integer</code> <code>name is string</code> <code>found is Boolean</code> <code>myArray[99]</code>	In Python you do not have declare what data type the variable is In Python we call an array a list

Sequence

Pseudocode	Python
<code>name is string</code> <code>input "Your name:"</code> <code>age is integer</code> <code>input "Your age:"</code>	<code>name = input("Your name: ")</code> <code>age = int(input("Your age: "))</code> <code>print(name, "in dog years you are", age * 7)</code>

```
output name, " in dog years you  
are"  
output age * 7
```

Assignment

Pseudocode	Python
set age=17	age = 17
set city = "Manchester"	city = "Manchester"
set names = ["Bob", "Baz", "Ann"]	names = ["Bob", "Baz", "Ann"]

32

Selection

Pseudocode	Python
<pre>if bobMood == "happy" set bobEmotion = ":)" endif</pre>	<pre>if bobMood == "happy": bobEmotion = ":)"</pre>
<pre>if lives > 0 output 'Carry on!' else output 'Game over.' endif</pre>	<pre>if lives > 0: print("Carry on!") else: print("Game over.")</pre>

Iteration (repetition)

Pseudocode	Python
<pre>for i =1 To 10 do output i *2 next i endfor</pre>	<pre>for i in range(1, 11): print(i*2)</pre>
<pre>count is integer set count=10 while count <>5 repeat set count =count-1 output "Countdown is" count</pre>	<pre>count = 10 while count != 5: count = count - 1 print("Countdown is ", count)</pre>
<pre>x is integer set x =0 repeat until x =10 do output x set x= x+1</pre>	<pre>x=0 while x <10: print(x) x=x+1</pre>
<pre>set myArray=[2, 4, 6, 8] for i in myArray do output myarray[i] endfor</pre>	<pre>myArray=[2, 4, 6, 8] for i in myArray print(myArray[i])</pre>

33

Operators

Pseudocode	Meaning	Python
>	Greater than	>
<	Less than	<
<=	Less than or equal to	<=
>=	Greater than or equal to	>=
<>	Not equal to	!=
= =	The same as	= =

AND	Both statements must be true for the argument as a whole to be true.	and
OR	Only one of the statements needs to be true for the argument as a whole to be true.	or
NOT	The opposite of	not
XOR	The argument is false if both statements are true. The argument is false if both statements are false. Otherwise the statement is true.	
DIV	Integer division Finds the quotient or the 'whole number of times' a divisor can be divided into a number. $11 \text{ DIV } 2 \rightarrow 5$ (2 divides into 11 a whole number of 5 times)	$11 // 2$
MOD	Modulo division Finds the remainder when a divisor is divided into a number. $11 \text{ MOD } 3 \rightarrow 2$ (11 divide by 3 gives a remainder of 2)	$11 \% 3$

34

FLOWCHARTS

Algorithms represented using a flowchart will use the following convention:

Start / Stop procedure

Decision box

Input / Output

Operation

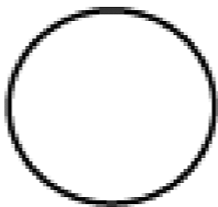
Connector

Store / Subroutine call

Flow of control

(Arrowhead indicates
direction of flow)





35