

7 Algorithm design and problem-solving

Name: _____ Class: _____ Date: _____

Total: 33 marks

KEY WORDS decomposition • validation • verification • trace table • test data

Objective

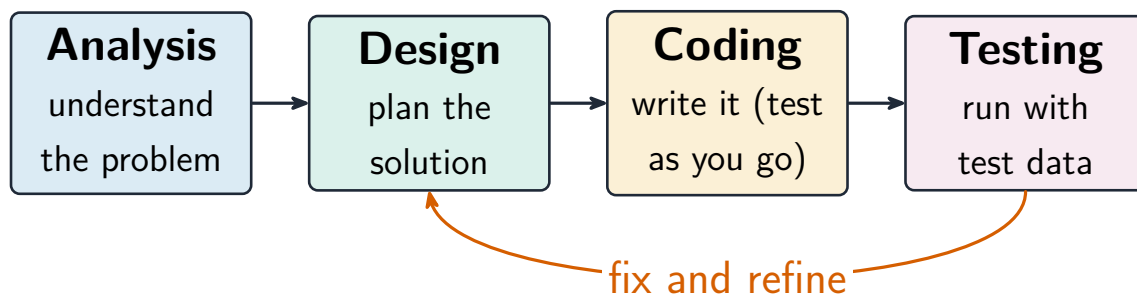
Build the skills to answer exam questions on **algorithm design and problem-solving** : the program development life cycle, decomposition, validation and verification, test data, trace tables and the standard methods of solution.

You must be able to:

- name the four stages of the program development life cycle and say what each one does
- split a problem into input, process and output
- choose a validation or verification check, and choose normal, abnormal, extreme and boundary data
- complete a trace table and spot a simple error in an algorithm

1 Key knowledge and worked examples

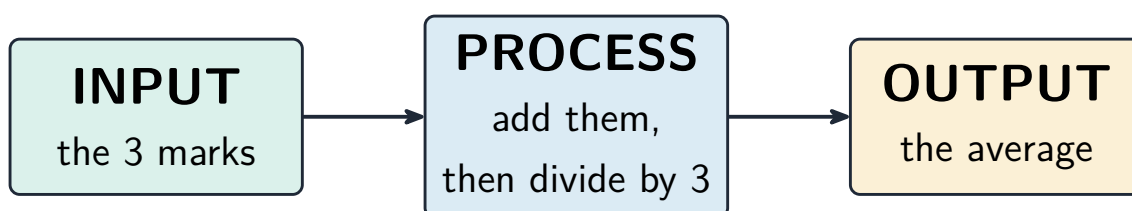
■ The life cycle and decomposition



Testing feeds back so the design can be fixed

Stage	What you do
analysis	find the problem and the requirements; use abstraction and decomposition
design	plan the solution with a structure diagram, a flowchart or pseudocode
coding	write the program and test small parts as you go
testing	run the program with test data

Abstraction keeps only the details that matter. **Decomposition** breaks one problem into smaller parts. A system is made of **sub-systems**, and those can be split again.



Every algorithm has an input, a process and an output

■ Checks on input, and test data

Validation asks “is this data sensible?”. It cannot prove the data is true.

Check	Question it asks
range check	is the value between the lowest and highest allowed?
length check	is the number of characters allowed?
type check	is it the right kind of data, such as a number?
presence check	was anything entered?
format check	does it match the pattern, such as dd/mm/yyyy?
check digit	does the extra digit agree with the rest of the number?

Verification asks “was this copied correctly?”. A **visual check** compares the typed data with the original. **Double entry** types the data twice and compares the two copies.

For a value that must be from 0 to 120:

Type	Meaning	Example
normal	accepted, and not on the edge	25
abnormal	rejected, wrong kind or clearly not allowed	-4 or cat
extreme	smallest and largest values still accepted	0 and 120
boundary	an accepted edge and the rejected value next to it	0 and -1; 120 and 121

■ Trace tables and two standard methods

A **trace table** writes each variable after every step. One row per pass. Do not keep the values only in your head.

count	total	output
1	5	
2	12	
3	20	20

Each column is one variable; each row is one step

Worked example. Inputs are 2, then 5, then 8.

```
Count ← 0
Total ← 0
FOR I ← 1 TO 3
    INPUT Value
    Total ← Total + Value
    Count ← Count + 1
NEXT I
OUTPUT Total / Count
```

I	Value	Total	Count	Output
1	2	2	1	
2	5	7	2	
3	8	15	3	5

The purpose is to find the average of three inputs. Total starts at 0 so the first value is not added to an old total.

A **linear search** checks each item from the start until it finds the target or reaches the end. A **bubble sort** compares each side-by-side pair and swaps the pair when the values are in the wrong order, then repeats until no swap is needed.

2 Practice

2.1 Name the four stages of the program development life cycle, in order.

[4]

2.2 State the difference between validation and verification. [2]

2.3 Name the validation check for each rule. [4]

- a password must contain at least 8 characters
- an age must be from 11 to 18
- a name must not be left blank
- a mark must be a number, not letters

2.4 A range check accepts whole numbers from 10 to 50. Give one normal value, the two extreme values, and one rejected boundary value. [3]

2.5 State what a trace table records, and why it is used. [2]

2.6 A program finds the largest of three input numbers. State the input, the process and the output. [3]

3 Exam-style questions

3.1 Keeping only the details that matter, and ignoring the rest, is: [1]

- ☐ **A** verification
B abstraction
C a presence check
D a bubble sort

3.2 A program stores a percentage from 0 to 100.

(a) Name the validation check that rejects 150. [1]

(b) Name the verification method that types the percentage twice and compares the copies. [1]

(c) Give one normal value, one abnormal value, one extreme value, and one boundary pair (an accepted edge and the rejected value beside it). [4]

3.3 Complete the trace table for this algorithm. The inputs are 3, then 9, then 4. [4]

```
Max ← 0
FOR I ← 1 TO 3
    INPUT Number
    IF Number > Max
        THEN
            Max ← Number
    ENDIF
NEXT I
OUTPUT Max
```

I	Number	Max	Output
---	--------	-----	--------

3.4 The algorithm in question 3.3 sets **Max** to 0 before the loop.

(a) Explain why the output is wrong when every input is negative. [2]

(b) State a correct value to store in **Max** before the comparisons, and which list position it comes from. [2]

Solutions

Each answer shows the steps, then the **result**.

2.1

- **analysis**
- **design**
- **coding**
- **testing**

2.2

- validation checks that data is sensible
- verification checks that data was copied or entered correctly

2.3

- at least 8 characters: **length check**
- age from 11 to 18: **range check**
- name not blank: **presence check**
- mark must be a number: **type check**

2.4

- one normal value strictly inside 10 to 50, such as **30**
- extreme values **10** and **50**
- one rejected boundary value, **9** or **51**

2.5

- it records the value of each variable after each step
- it is used to follow an algorithm and find what it does, or to find an error

2.6

- input: the three numbers
- process: keep the largest value seen so far
- output: that largest value

3.1 B

- abstraction keeps only the important details
- A is a copy check, C is a validation check, and D is a sort

3.2

(a)

- **range check**

(b)

- **double entry**

(c)

- normal: a value such as **40**
- abnormal: a non-number such as **cat**, or a value such as **-5**
- extreme: **0** or **100**
- boundary: an accepted edge together with the rejected neighbour, such as **0 and -1** or **100 and 101**

3.3

- after input 3, **Max** is **3**
- after input 9, **Max** is **9**
- after input 4, **Max** stays **9**
- the output is **9**

I	Number	Max	Output
1	3	3	
2	9	9	
3	4	9	9

3.4

(a)

- 0 is greater than every negative input, so the condition **Number** > **Max** is never true
- **Max** stays 0, which was not one of the inputs

(b)

- store the **first number** in **Max** before comparing the rest
- that value is **List[1]**, or the first input