9 Algorithm Design and Problem-solving – Part 1

Name: _____ Class: _____ Date: _____

Vocabulary

Write each word three times.

English	**Write it 3 times** (English)		
Computational thinking			
decomposition			
abstraction			
algorithm			
sequence			
selection			
iteration			
flowchart			
pseudocode			
variable			

Recall

You have solved problems step by step before. Bring this with you:

- A clear set of steps can be followed by someone else to get the same result.
- You have read simple flowcharts.

At AS we learn to think like a computer scientist and to write algorithms carefully – the

exercises.lol exercises.com.cn

planning that comes before any code.

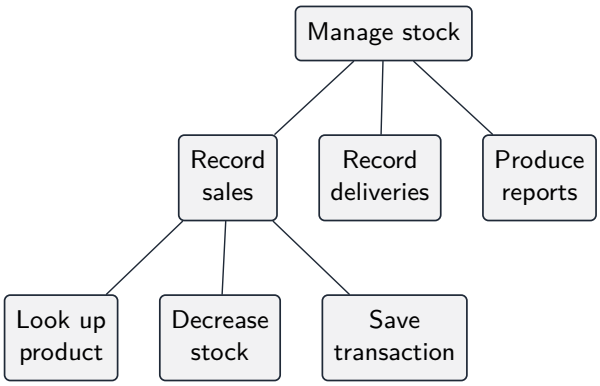
New ideas

Read these first. Each idea has a short worked example. Then do the Practice.

1 Computational thinking

Computational thinking means solving a problem in a clear, step-by-step way that a computer could follow. Two key skills:

- **decomposition** : break a big problem into smaller, easier parts;
- **abstraction** : keep only the details that matter and ignore the rest.

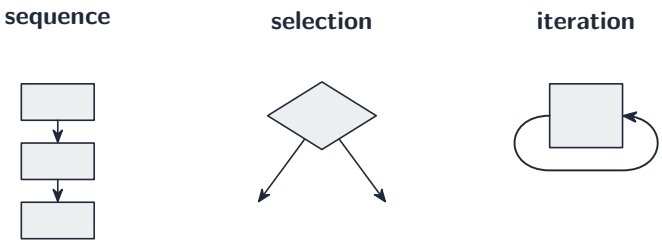


Worked example. To build a quiz app, decomposition splits it into "ask a question", "check the answer" and "keep the score"; abstraction ignores the screen colour and font, which do not affect how it works.

2 Algorithms and the three building blocks

An **algorithm** is a set of clear, ordered steps that always finishes. Every algorithm is built from just three structures:

- **sequence** : steps done in order;
- **selection** : choose between paths with `IF`;
- **iteration** : repeat steps with a loop.

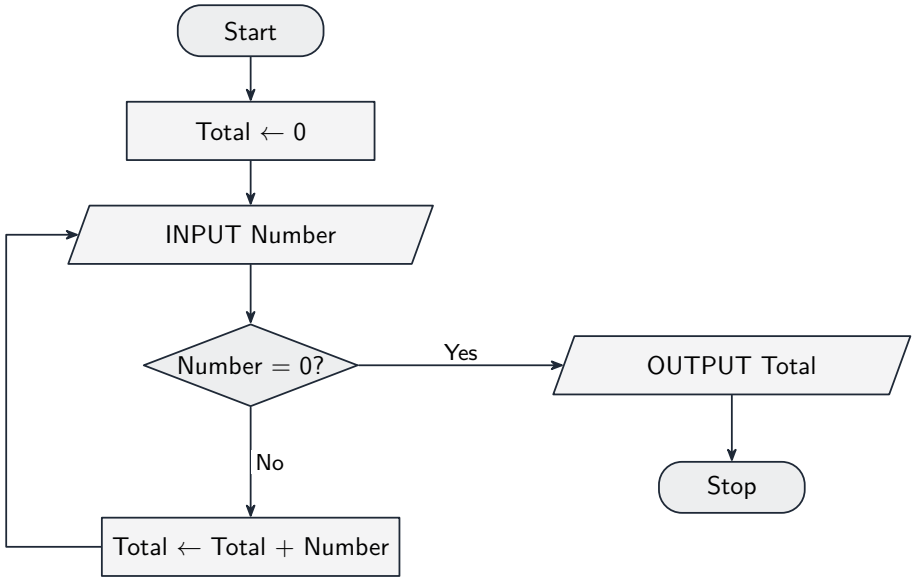


Worked example. "Keep asking for a password until it is correct" uses all three: read the input (sequence), check it (selection), and repeat while it is wrong (iteration).

Watch out: an algorithm must always *finish*. A loop needs a condition that eventually becomes false – or it runs forever.

■ 3 Writing it down

We plan an algorithm as a **flowchart** or in **pseudocode** (code-like English).



Pseudocode stores values in a **variable** :

```
INPUT age
IF age >= 18 THEN
    OUTPUT "adult"
ELSE
    OUTPUT "minor"
ENDIF
```

Worked example. If `age` is 20, the test `age >= 18` is true, so this outputs "adult".

Practice

9.1 Explain what decomposition means. [1]

9.2 Explain what abstraction means. [1]

9.3 State which building block an `IF … ELSE` statement is. [1]

9.4 Name the three building blocks of any algorithm. [3]

9.5 Write pseudocode that inputs two numbers and outputs their sum. [3]

9.6 Explain why breaking a large program into smaller parts (decomposition) makes it easier to write and test. [2]

9.7 Challenge. Write pseudocode that keeps inputting numbers until a 0 is entered, then outputs their total. [3]