Algorithm Design

IGCSE Computer Science ▪ Topic 7

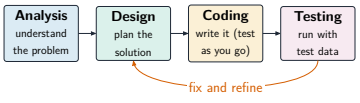
IGCSE Computer Science

What we will cover

1. The development life cycle
2. Design tools
3. Algorithms & IPO
4. Validation
5. Trace tables
6. Search & sort

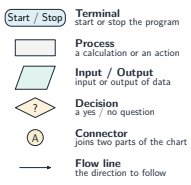
1 Designing

The development life cycle

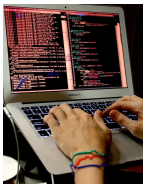


- **abstraction** 抽象 – keep only the important details and ignore the rest;
- **decomposition** 分解 – break a big problem into smaller, easier parts.

The program development life cycle



A program flowchart sets out the steps and decisions of a program during the design stage



Software is written by programmers, who follow the development life cycle

> A program flowchart sets out the steps and decisions of a program during the design stage

Decomposing a problem, and testing what you build

Sub-systems	Break the problem into **sub-systems** 子系统, each with its own inputs, **processing** 处理 and outputs – then solve them one at a time.
Test data	Choose **test data** 测试数据 deliberately: normal, boundary, abnormal – and know the expected result *before* running it.
Iterative testing	**Iterative testing** 迭代测试: test each module as it is written, fix, and test again, rather than waiting for the whole program.
Then the whole	Final testing runs the assembled program against the original requirements.

> A test with no expected result is not a test. Write the expected value first, or you will accept whatever appears.

Three design tools

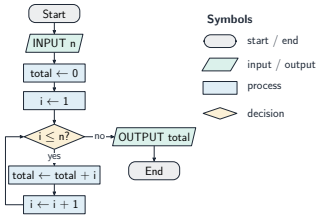
structure diagram
the parts of a system and how they fit

flowchart 流程图
boxes and arrows for the steps in order

pseudocode 伪代码
steps in simple, code-like English

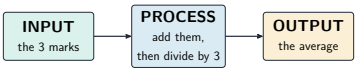
None is a real language – they plan the solution before any code is written.

A flowchart for the sum algorithm



A flowchart for the sum algorithm, using the standard symbols (start/end, input/output, process, decision)

Algorithms: input, process, output



An **algorithm** 算法 is an ordered set of steps that solves a problem. Every one splits into:

input → **process** → **output**

"Average of three marks": input the marks, process add and divide by 3, output the average. Name all three in an exam answer.

2 Checking

Validation checks

range 范围 – between a low and high value

length 长度 – allowed number of characters

type 类型 – the right kind (number, not letters)

presence 存在性 – something was entered

format 格式 – the right pattern (dd/mm/yyyy)

Validation checks data is *sensible*; verification (visual or double-entry) checks it was *copied right*.

The validation checks, by name

Presence check	A **presence check** 存在性检查: the field must not be left empty.
Length check	A **length check** 长度检查: the entry has the right number of characters – a password of at least eight, a code of exactly six.
Range and type	A range check keeps a number between limits; a type check rejects letters in a numeric field; a format check enforces a pattern.
Check digit	A **check digit** 校验码 is calculated from the other digits and appended, so a mistyped or swapped digit is caught at entry.

Validation checks that the data is *sensible*; verification checks it was *copied correctly*. They catch different mistakes.

Worked example – a trace table

Trace with input $n = 5$. What does it output?

```
total <- 0
FOR i <- 1 TO n
total <- total + i
NEXT i
OUTPUT total
```

i	total	OUT
1	1	
2	3	
3	6	
4	10	
5	15	15

It sums 1 to n: output 15. Fill *one row per pass* – never run the loop in your head.

Trace tables

count	total	output
1	5	
2	12	
3	20	20

A trace table records each variable's value as the program runs

3 Search and sort

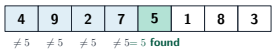
Linear search and bubble sort

- **linear search** 线性查找: check every item until found
- **bubble sort** 冒泡排序: compare neighbours, swap, repeat until no swaps

Linear search works on any list; bubble sort finishes when a whole pass makes no swaps.

Linear search and bubble sort, in detail

search for 5: check each item in turn (left to right)



linear search – check each in turn

compare the pair 5 and 2: $5 > 2$, so swap them

then repeat for each pair, until no swaps are needed

bubble sort – swap neighbours each pass

Linear search checks each item in turn from the start; bubble sort makes repeated passes, swapping neighbours that are the wrong way round.

Totalling and counting

- **totalling** 求和 – keep adding values to a running total (`total ← total + value`).
- **counting** 计数 – add 1 to a counter each time something happens (`count ← count + 1`).

Maximum, minimum and average

- to find the **maximum** 最大值: keep the largest value seen so far.
- to find the **minimum** 最小值: keep the smallest value seen so far.
- to find the **average** 平均值: divide the total by how many values there are.

Exam tips

- Learn the four life-cycle stages: analysis → design → coding → testing. **Abstraction** keeps only the important details; **decomposition** breaks a problem into smaller parts.
- **Validation** checks data is *sensible* (range, length, type, presence, format checks); **verification** checks it was *copied correctly* (a visual check or double entry).
- Learn the four test-data types: **normal** (accepted), **abnormal** (rejected), **extreme** (the largest/smallest still allowed), **boundary** (the values either side of a limit).
- To work out what an algorithm does, fill in a **trace table** – write down every variable's value at each step.
- Know the standard algorithms: **linear search** (check each item in turn) and **bubble sort** (swap side-by-side pairs until no swaps are needed).

4 Recap

Topic 7 – key takeaways

1. **Design** — abstraction + decomposition; 3 design tools
2. **IPO** — every algorithm: input, process, output
3. **Check** — validation = sensible; verification = copied right
4. **Trace** — one row per pass to see what code does

Check yourself

1. What are the three parts of every algorithm?
2. Range check or type check for "age 0–120"?
3. Validation or verification: double entry?
4. When does a bubble sort stop?

Answers **1.** input, process, output **2.** range **3.** verification **4.** when a pass makes no swaps