9 Algorithm Design and Problem-solving – Part 2

Name: _____ Class: _____ Date: _____

Vocabulary 词汇

Write each word three times. 把每个单词抄写三遍。

English	中文	Write it 3 times (English)		
flowchart	流程图			
trace table	追踪表			
Stepwise refinement	逐步求精			
linear search	线性查找			

Recall

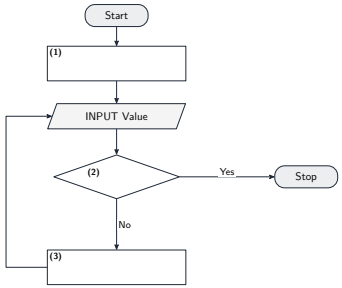
In **Part 1** you met computational thinking (decomposition, abstraction), the three constructs (sequence, selection, iteration), and pseudocode. Part 2 adds drawing and checking flowcharts, refining a design step by step, and a standard searching algorithm.

New ideas

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

1 Flowcharts and trace tables

A **flowchart** 流程图 draws an algorithm with standard shapes: a rounded box = start/stop, a parallelogram = input/output, a rectangle = a process, a diamond = a decision, and arrows = the flow.



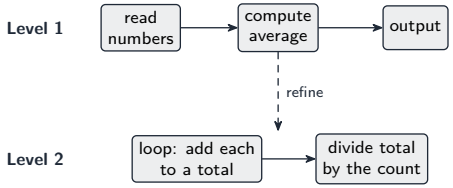
To check an algorithm, make a **trace table** 追踪表 — a column for each variable — and fill in the values row by row as you "run" the steps by hand.

Worked example. Tracing `total ← 0; FOR i ← 1 TO 3; total ← total + i`: the rows give `total` = 0, 1, 3, 6, so the final total is 6.

2 Stepwise refinement

Stepwise refinement 逐步求精 starts with a high-level outline and expands each step until it can be coded:

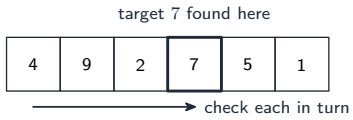
- *Level 1:* read the numbers → compute the average → output it;
- *Level 2:* the input loop that adds each number to a running total, then dividing by the count.



Worked example. "Make a sandwich" refines to "get bread → add filling → close it"; then "add filling" refines further into "spread butter → add cheese".

3 A standard algorithm: linear search

A **linear search** 线性查找 checks each element in turn until it finds the target (or reaches the end). It is simple and works on any list, but is slow on a long list.




```
FOR i ← 1 TO n
    IF A[i] = Target THEN
        OUTPUT "Found at ", i
    ENDIF
NEXT i
```

Worked example. Searching `[4, 9, 2, 7]` for 7: check 4 (no), 9 (no), 2 (no), 7 (yes) — found at position 4.

Watch out: when you fill a trace table, follow the steps *exactly* as written — do not silently "fix" the algorithm in your head, or you will miss the very bug you are hunting.

Practice

9.1 State the flowchart shape used for (a) a decision, (b) an input or output. [2]

9.2 State one advantage of a linear search over a more complex search. [1]

9.3 Trace this code and give the final value of `x`: `x ← 0`; `FOR i ← 1 TO 4`; `x ← x + i`; `NEXT i`. [2]

9.4 Explain what stepwise refinement means. [2]

9.5 State how a linear search works, and when it is slow. [2]

9.6 Explain why a trace table is useful for finding a logic error that does *not* crash the program. [2]

9.7 Challenge. A linear search looks through a list of 1000 items. State the largest number of comparisons it might need, and explain when that happens. [2]