

Exercise walk-through

# 5.1

## Algorithms, traces and correctness evidence

---

International Baccalaureate · IB Diploma · Computer Science · HL

# 5.1 Algorithms, traces and correctness evidence

---

Name: \_\_\_\_\_ Class: \_\_\_\_\_ Date: \_\_\_\_\_ **Total: 10 marks**

**KEY WORDS** algorithm • boundary test

# Objective

---

Algorithms, traces and correctness evidence. Supported HL concept preparation; original classroom practice, not an official IB paper or assessed learner work.

## 1 Key knowledge and worked examples

---

### ■ What would explain this observation?

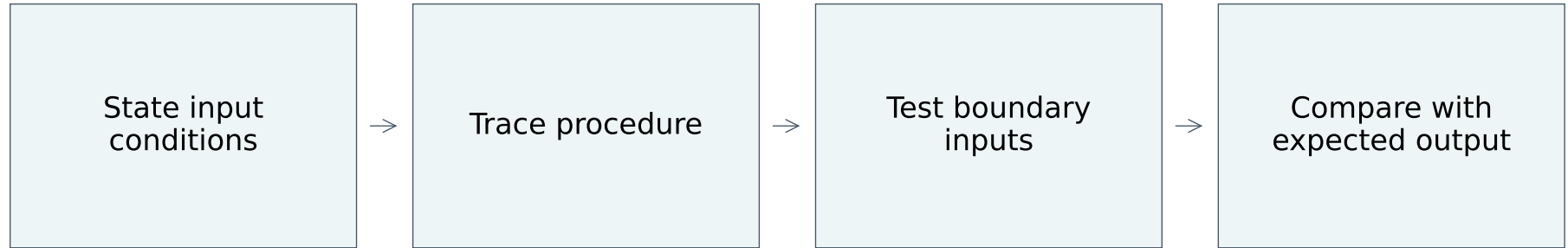
- An **algorithm** can work for one example and fail at a boundary. Testing should be designed from the specification, not only the happy path.
- Start with a prediction. State the quantities or features you would compare, then decide what evidence could distinguish two explanations.

---

## ■ Build the model

- An algorithm is a finite, unambiguous procedure for a task. A trace records state changes. A loop invariant describes a property preserved by each iteration and helps justify correctness.

- **algorithm**: A finite procedure solving a stated task; **boundary test** : A test at a limit of the allowed input range.



---

■ **Choose evidence that can test it**

- State input conditions and expected outputs. Use boundary cases, empty collections where allowed, duplicates and invalid values. Distinguish a wrong algorithm from a wrong implementation or an incomplete requirement.
- Trace a search over a small fictional sorted list. State the indexing convention. For binary search, update bounds so the remaining interval shrinks and reject unsorted input unless sorting is part of the task.

---

■ **Work from known quantities**

- State the known values and their units. Choose the relation because its assumptions

fit this case, then rearrange before substitution.

- Known: a linear search of an eight-item list can require eight comparisons when the sought item is last or absent. Doubling the list length doubles the worst-case comparison count under this model. Binary search reduces the interval by roughly half each step but requires a suitable ordered structure.

### **Example:**

A linear search scans all 14 items without a match. How many item comparisons occur? Use the same sequence: known quantities  $\rightarrow$  model  $\rightarrow$  relation  $\rightarrow$  substitution  $\rightarrow$  unit and interpretation.

---

■ Check the conclusion and its limits

- A successful sample test does not prove correctness for all valid inputs. Do not import Cambridge-specific pseudocode syntax into an IB course without a course source.
- Return to the original observation. Explain what the result supports, which conditions it assumes, and one way to test a competing explanation.

### **Warn:**

One successful sample test proves an algorithm correct for every valid input. This claim is false: A successful sample test does not prove correctness for all valid inputs. Do not import Cambridge-specific pseudocode syntax into an IB course without a course source.

### **Key:**

Algorithms, traces and correctness evidence: State input conditions and expected outputs. Use boundary cases, empty collections where allowed, duplicates and invalid values.

Distinguish a wrong algorithm from a wrong implementation or an incomplete requirement.

---

## ■ Runnable trace and boundary

```
def first_index(values, target):  
    for index, value in enumerate(values):  
        if value == target:  
            return index  
    return -1  
print(first_index([4, 7, 4], 4))  
print(first_index([4, 7, 4], 9))  
print(first_index([], 4))
```

Expected output:

```
0  
-1  
-1
```

The loop tests index 0 first, stops on a match, and returns  $-1$  only after exhausting the list. Duplicates therefore return the first index. An empty list executes no loop body.

## Exercises 2.1-2.3

---

**2.1** What must hold before applying ordinary binary search to a list? Choose one answer.

- **A** The list is ordered under the comparison rule
- **B** Every value is different
- **C** The list contains exactly ten items

[1]

---

**2.2** A linear search scans all 14 items without a match. How many item comparisons occur? Show the relation, substitution and units.

[2]

---

**2.3** Decide whether this claim is true and justify: One successful sample test proves an algorithm correct for every valid input.

[2]

## Exercises 2.1-2.3

---

**2.1** What must hold before applying ordinary binary search to a list? Choose one answer.

- **A** The list is ordered under the comparison rule
- **B** Every value is different
- **C** The list contains exactly ten items

[1]

**2.1 A** State input conditions and expected outputs. Use boundary cases, empty collections where allowed, duplicates and invalid values. Distinguish a wrong algorithm from a wrong implementation or an incomplete requirement.

---

**2.2** A linear search scans all 14 items without a match. How many item comparisons occur? Show the relation, substitution and units.

[2]

**2.2** One equality test per visited item; all 14 visited means 14 comparisons.

---

**2.3** Decide whether this claim is true and justify: One successful sample test proves an algorithm correct for every valid input.

[2]

**2.3** False. A successful sample test does not prove correctness for all valid inputs. Do not import Cambridge-specific pseudocode syntax into an IB course without a course source.

## Exercises 2.4·3.1

---

**2.4** A finite procedure solving a stated task: write the technical term. [1]

### 3 Exam-style questions

---

**3.1** Trace first-match linear search on  $[4,7,4]$  for 4 and 9. Give results and comparison counts, including an empty-list boundary. [4]

## Exercises 2.4-3.1

---

**2.4** A finite procedure solving a stated task: write the technical term. [1]

### 3 Exam-style questions

---

**2.4** algorithm. algorithm means A finite procedure solving a stated task.

---

**3.1** Trace first-match linear search on  $[4,7,4]$  for 4 and 9. Give results and comparison counts, including an empty-list boundary. [4]

**3.1** For 4 return index 0 after one comparison; for 9 return  $-1$  after three. Empty list returns  $-1$  after zero. Use zero-based indices and stop at the first match.