19.1 Algorithms

Name: ______________ Class: __________ Date: ____________ **Total: 31 marks**

KEY WORDS bubble sort 冒泡排序 • binary search 二分查找 • insertion sort 插入排序 • linear search 线性查找 • array 数组

Objective

Build the skills to answer exam questions on **algorithms** — linear and binary search, bubble and insertion sort, and comparing algorithms.

You must be able to:

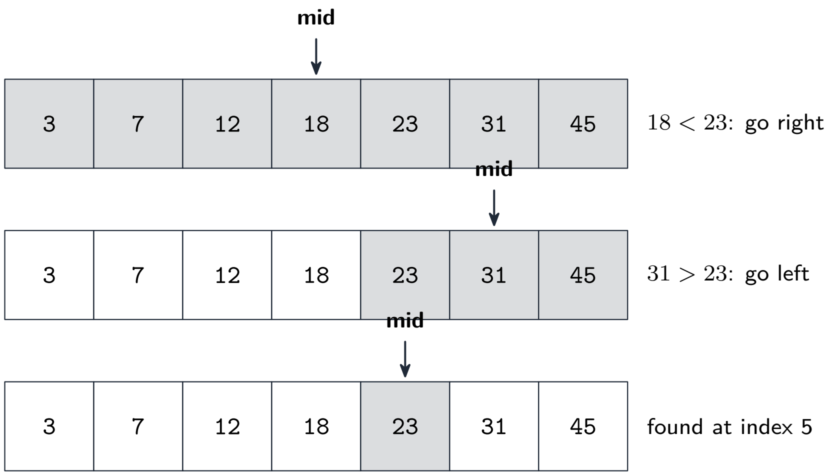
- trace a **linear** and a **binary** search
- trace a **bubble sort** and an **insertion sort**
- compare algorithms by **time** and **memory** (Big-O)

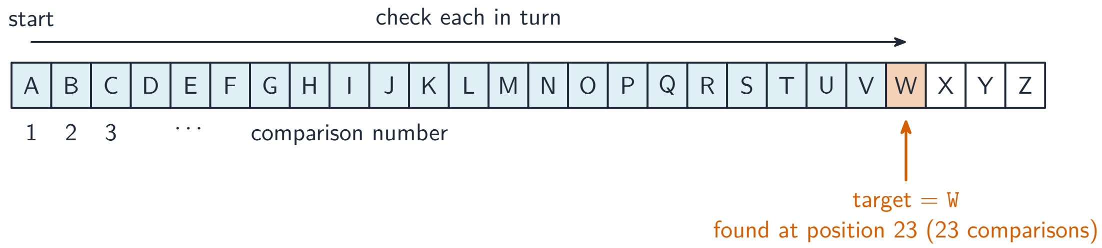
1 Worked examples

Study these first. Each one shows the method for a question type used later — follow the steps and you can do the Practice and Exam-style questions yourself.

Searching

A **linear search** checks each element in turn — works on **any** list, worst case **O(n)**. A **binary search** needs the data **sorted**: check the middle, then keep the half that could contain the target, halving the range each step — worst case **O(log n)**:





Linear search compared with binary search

Bubble sort

Repeatedly walk the array, **swapping** adjacent out-of-order pairs, so the largest "bubbles" to the end each pass:

```
FOR pass ← 1 TO n - 1
    FOR i ← 1 TO n - pass
        IF A[i] > A[i+1] THEN
            swap A[i], A[i+1]
        ENDIF
    NEXT i
NEXT pass
```

Insertion sort

Build a sorted prefix; insert each new element by **shifting larger ones right**:

```
FOR i ← 2 TO n
    key ← A[i]
    j ← i - 1
    WHILE j >= 1 AND A[j] > key DO
        A[j+1] ← A[j]; j ← j - 1
    ENDWHILE
    A[j+1] ← key
NEXT i
```

Both are $\mathbf{O(n^2)}$ worst case; both are $\mathbf{O(n)}$ on already-sorted data (with an early exit / no shifts). Compare algorithms by **time taken** and **memory used**.

2 Practice

Now apply the methods above.

2.1 State the **precondition** that a binary search needs. [1]

2.2 State the **worst-case** time complexity of a **linear** search and a **binary** search. [2]

2.3 In a **bubble sort**, describe what happens during **one pass**. [1]

2.4 State **one** criterion used to compare two algorithms. [1]

3 Exam-style questions

3.1 Trace a **binary search** for `23` in `[3, 7, 12, 18, 23, 31, 45]` (indices 1–7). State the **index** examined at each step. [3]

3.2 The array `[5, 2, 8, 1]` is sorted with a **bubble sort**. Show the array **after the first complete pass**. [2]

3.3 State **two** advantages of a binary search over a linear search, and **one** disadvantage. [3]

3.4 State why an **insertion sort** is efficient on a list that is **already nearly sorted**. [1]

3.5 The algorithm below is intended to find the position of the first negative value in an array `Data` of 10 integers, or to output a message if there is none.

```
Found ← FALSE
Index ← 1
FOR Index ← 1 TO 10
    IF Data[Index] < 0
      THEN
        Found ← TRUE
        Position ← Index
    ENDIF
NEXT Index
IF Found = TRUE
  THEN OUTPUT Position
  ELSE OUTPUT "None found"
ENDIF
```

The array holds: `8, 3, -6, 11, -2, 7, 4, 9, 1, 5`.

(a) **Complete** the trace table by dry running the algorithm for the first **five** iterations. [4]

Index	**`Data[Index]`**	**Found**	**Position**

(b) **State** the value the algorithm outputs, and **explain** why it is **not** the position of the *first* negative value. [3]

(c) The outer loop structure is **not** the most appropriate one for this task. **Explain** why, and **name** the loop you would use instead. [3]

(d) **Write** corrected pseudocode using that loop, so the algorithm stops as soon as it finds the first negative value. [4]

(e) **Calculate** how many comparisons your corrected version makes on this data, and how many the original makes. **Describe** what this shows about the two loop choices when the target is near the start. [3]