A-LEVEL COMPUTER SCIENCE • EXERCISE SHEET

19.1 Algorithms

Name: _____ Class: _____ Date: _____ **Total: 14 marks**

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:

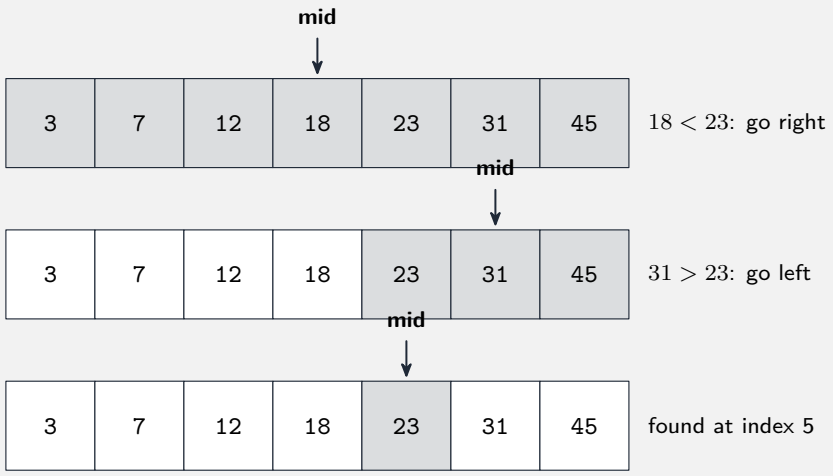
- trace a **linear** and a **binary** search
- trace a **bubble sort** and an **insertion sort**
- compare algorithms by **time** and **memory** (Big-O)

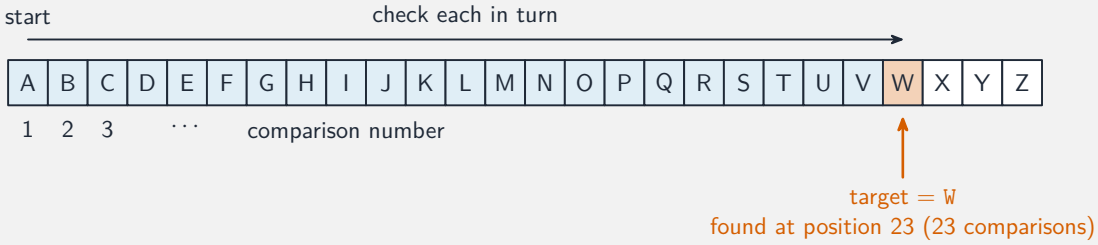
1 Worked examples

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]
    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]

4 Go further

You are now ready for the real exam questions on this subtopic. Open the **19.1 Algorithms** past-paper sheet in the Library, or try these in **Practice mode**:

- 9618/21 J25 —Q4 (searching and sorting)
- 9618/22 J25 —Q7 (a sort algorithm)
- 9618/33 J25 —Q10 (binary search)
- 9618/33 N24 —Q11 (algorithm efficiency)

Solutions

2.1 The data must be **sorted** (in order).

2.2 Linear search: **O(n)**; binary search: **O(log n)**.

2.3 Adjacent pairs are compared along the array and **swapped if out of order**, moving the largest remaining value to the end.

2.4 Any one: **time taken** (number of operations); **memory used**.

3.1 mid = index **4** (value 18, `<23` → go right); mid = index **6** (value 31, `>23` → go left); mid = index **5** (value 23 → found).

3.2 First pass: `5 2`→`[2,5,8,1]`; `5,8` ok; `8 1`→`[2,5,1,8]`. Answer: **`[2, 5, 1, 8]`**.

3.3 Advantages (any two): **far fewer comparisons** on large lists (O(log n) vs O(n)); much faster when searching repeatedly. Disadvantage: the data must be **sorted first** (a one-off cost).

3.4 Most elements are already in place, so the inner `WHILE` does **few or no shifts** — close to O(n).