

# Knowledge Organiser 2.1: Searching and Sorting Algorithms

## 1. Binary Search

- The Algorithm**
- Calculate a mid-point in the data set.
  - Check if that is the item to be found.
  - If not...
    - If the item to be found is lower than the mid-point, repeat on the left half of the data set.
    - If the item to be found is greater than the mid-point, repeat on the right

## 2. Linear Search

- The Algorithm**
- Starting from the beginning of a data set, each item is checked in turn to see if it is the one being searched for

- Requirements / Efficiency**
- Doesn't require the data set to be in order.
  - Will work on any type of storage device

## 3. Bubble Sort

- The Algorithm**
- Sorts an unordered list of items.
  - It compares each item with the next one and swaps them if they are out of order.
  - The algorithm finishes when no more swaps need to be made.
  - In effect it "bubbles" up the largest (or smallest) item to the end of the list in successive passes.

- Efficiency**
- This is the most inefficient of the sorting algorithms but is very easy to implement.
  - This makes it a popular choice for very small data sets

x To remember the code for these algorithms

## 4. Insertion Sort

- The Algorithm**
- The insertion sort inserts each item into its correct position in a data set one at a time.

- Efficiency**
- It is a useful algorithm for small data sets.
  - It is particularly useful for inserting items into an already sorted list.
  - It is usually replaced by more efficient sorting algorithms for large data sets.

- Uses a divide and conquer method.
- Creates two or more identical sub-problems from the largest problem, solving them individually.
- Combines their solutions to solve the bigger program.

**Sorted** | **Unsorted**

The insertion sort algorithm uses two lists, one sorted and one unsorted.

Elements are gradually moved from the unsorted list to the correct position in the sorted list.

*Relatively efficient when used with small lists.*

**Pass 1**

The bubble sort algorithm works through a list, comparing pairs of values and swapping them if necessary.

It keeps on passing through the list comparing values and making swaps until the list is sorted.

*Easy to implement; however, it isn't very efficient.*

**Pass 2**

The merge sort algorithm works by splitting a list into individual elements and gradually merging them into larger and larger sorted lists until they are in one sorted list.

*Very efficient when used with both large and small lists.*