

OCR Computer Science AS Level

2.3.1 Sorting Algorithms

Concise Notes

Specification:

- Standard algorithms
 - Bubble sort
 - Insertion sort

Sorting Algorithms

- Take a number of elements in **any order** and output them in a **logical order**
- This is usually **numerical** or **lexicographic** (phonebook style ordering)
- Most output elements in **ascending** order, but can typically be slightly **altered** or their output **reversed** in order to produce an output in descending order

Bubble Sort

- Makes **comparisons and swaps** between pairs of elements
- The largest element in the unsorted part of the input is said to “bubble” to the top of the data with each iteration of the algorithm
 - Starts at the first element in an array and compares it to the second
 - If they are in the wrong order, the algorithm **swaps** the pair
 - The process is then repeated for every adjacent pair of elements in the array, until the end of the array is reached
- This is one **pass** of the algorithm
- For an array with n elements, the algorithm will perform n passes through the data
- After n passes, the input is sorted and can be returned

A = Array of data

```
for i = 0 to A.length - 1:
    for j = 0 to A.length - 2:
        if A[j] > A[j+1]:
            swap A[j] and A[j+1]
return A
```

- Can be modified to improve efficiency
- A **flag** recording **whether a swap has occurred** is introduced
- If a full pass is made **without any swaps**, then the algorithm **terminates**
- With each pass, one **fewer element needs comparing** as the n largest elements are **in position** after the n^{th} pass
- Bubble sort is a **fairly slow** sorting algorithm, with a time complexity of $O(n^2)$

Insertion Sort

- Places elements [into a sorted sequence](#)
- In the i^{th} iteration of the algorithm the first i elements of the array are sorted
 - Warning: although the i elements are [sorted](#), they are not the i [smallest elements](#) in the input!
- Starts at the [second element](#) in the input, and compares it to the element [to its left](#)
- When compared, elements are [inserted into the correct position](#) in the sorted portion of the input to their left
- This continues until the last element is inserted into the correct position, resulting in a fully sorted array
- Has the same time complexity as bubble sort, $O(n^2)$

A = Array of data

```
for i = 1 to A.length - 1:
    elem = A[i]
    j = i - 1
    while j > 0 and A[j] > elem:
        A[j+1] = A[j]
        j = j - 1
    A[j+1] = elem
```