Efficiency and Algorithmic Analysis

# Exercise 1. Tutorial

1. An algorithm should be regarded as a technology in the same way as hardware, as it has an input, performs an operation and provides an output
2. We should not optimise early as it is easier to make a correct program faster, but it is more difficult to make a fast program correct.
3. - Asymptotic: Large Values

* Complexity: Performance

1. A heuristic is a robust estimate of performance of an algorithm. This can be at design-time. It is independent of implementation details.
2. Worst case efficiency is useful to a system designer as it lets the system be designed to minimize any hiccups during real-world performance, such as a large time delay.
3. In Big O notation, a league represents a different growth rate of O(n)
4. Big O disregards all but the fastest growing term as algorithms are likely to have to work with a large input set, such as sorting 1,000,000 numbers. The fastest growing term will be the most influential term that will change the growth rate the most.

# Exercise 2. Assessing Efficiency

1. Insertion sort has a Big O complexity of O(n^2) (n-squared) as there is one nested loop. Therefore, n \* n.
2. The best-case efficiency for bogosort is O(1), where the input is already sorted. Also, the randomly generated numbers could be generated in the correct order on the first try. This is of no use to the system designer, as either of these outcomes are very unlikely. This chance further decreases the bigger the input.

# Exercise 3. Comparing Efficiencies

1. Merge sort performs better than insertion sort at large input sizes, and merge sort is in a league of O(nlog2n) in comparison to O(n2) of insertion sort.
2. It might be better to implement insertion sort if input sizes are guaranteed to be small as it could be quicker, due to no overhead from creating stack frames due to recursion, and it also has a space complexity of O(1), meaning it just uses one variable to sort.
3. Merge sorting an array where n = 177147 results in the stack being 177147 \* 17.43 (18) = 3,188,646 units deep.
4. For an input size n = 1,000,000 the stack would be 1,000,000 \* 19.93 (20) = 20,000,000