Problems

In computer science, a *problem* refers to a general class of problems where in mathematics it might refer to a specific example. Thus our solutions are algorithms which are more generally applicable to a class of problems. Some examples of problem classes include

- Sorting problems, where an instance of a problem is a sequence of items.
- Graph colouring, where an instance is a graph.
- Equation solving problems, where an instance is a system of equations.

An *algorithm* is a finite sequence of instructions with

- No ambiguity and each step precisely defined
-

Euclid's Algorithm

With the observation that
`gcd(m, n)== gcd(max(m, n)- min(m, n), min(m, n))` we have a recursive relation to find the greatest common denominator of two values.

```
int gcd(int m, int n) {
    if (m == n)
        return m;
    else
        return gcd(max(m, n) - min(m, n), min(m, n));
}
```

However, for large numbers this algorithm requires a very large number of iterations. Luckily, we have an operator that allows for easy repeated subtractions. The modulo operator allows us to instead use the algorithm `gcd(m, n)= gcd(n, n mod m)`.

```
int gcd(int m, int n) {
    if (n == 0)
        return m;
    else
        return gcd(n, n % m);
}
```

Although these two algorithms accomplish the same task, one performs it with much greater efficiency. Rather than perform this algorithm recursively we can do it iteratively, saving us the overhead of a function call.

```
int gcd(int m, int n) {
    while (n) {
        int r = m % n;
        m = n;
        n = r;
    }
    return m;
}
```

Data Structures

A linear data structure is a data structure where elements are ordered in some conceptual line. These include arrays, lists and variants like stacks and queues.

- Arrays have the benefits of being very fast to read from and write to individual cells. The downsides are that they are contiguous and of fixed size.
- Linked lists have the benefits of rapid extension and reduction in size. Selection of a specific element however requires traversal of the list and is quite slow.

Stacks and queues are abstract data types as opposed to the more concrete examples of arrays and lists. These are abstract as they tend to have more basic structures at a low level, with their special operations implemented as useful abstractions on top. Where a stack is last in first out, a queue is last in

first out. A stack could be implented efficiently as an array by keeping an index for the top element, incrementing for insertion and decrementing to pop. This wouldn't be as efficient for a queue due to the need to remove the first rather than the last element.

Growth Rates

When considering the performance of an algorithm, we look at how the time and space used by the algorithm grow as the input size increases. We generally look at the worst case scenario, though we can also look at the worst case scenario. When doing this, we ignore the details of the hardware and compilation and focus on the mathematical performance of the algorithm.

We assume that data is represented in fixed length words. Additionally we assume that fundamental operations take constant time. These include arithmetic, memory access, comparison and more. If an operation costs some amount c and it is called $g(n)$ times, then generally the cost of an algorithm is roughly $c \cdot g(n)$ for sufficiently large n. Identifying the basic operation and the number of times called is the fundamental part of the procedure.

Amortised takes into account the context an algorithm is run in and calculates the cost over a large number of runs. This is useful for algorithms that have a very expensive operation that might be called only once each time they are run.