Introduction to Algorithms

Informally, an algorithm is any well-defined computational procedure that takes some value, or set of values, as input and produces some value, or set of values, as output in a finite amount of time. An algorithm is thus a sequence of computational steps that transform the input into the output.

Let us consider the problem of preparing an omelet. To prepare, we follow these steps:

```
1) Get the frying pan.
2) Get the oil.
        a. Do we have oil?
                i. If yes, put it in the pan.
                ii. If no, do we want to buy oil?
                        1. If yes, then go out and buy.
                        2. If no, we can terminate.
3) Turn on the stove, etc...
```

An algorithm for a computational problem is correct if every problem instance provided as input halts or finishes its computing in finite time and outputs the correct solution to the problem instance.

There are two main criteria for judging the merits of algorithms:

- Correctness (does the algorithm solve the problem in a finite number of steps?)
- Efficiency (how much resources in terms of memory and time does it take to execute?)

Definition: A finite set of statements that guarantees an optimal solution in a finite interval of time

Examples of real-life algorithms:

i) The internet allows people all over the world to easily and quickly access a lot of information. Smart algorithms help websites manage this huge amount of data by finding the best paths for the data to travel. They also use search engines to quickly find pages that contain specific information.

ii) You need to compress a large file containing text to occupy less space. Algorithms such as Huffman coding allow you to store with fewer bits.

Algorithm Specification:

- Input - Every algorithm must take zero or more number of input values from the external.
- Output - Every algorithm must produce an output as a result.
- Definiteness - Every statement/instruction in an algorithm must be unambiguous (only one interpretation)
- Finiteness - The algorithm must produce results within a finite number of steps for all cases.
- Effectiveness - Every instruction must be basic enough to be carried out, and it must also be feasible.

To analyze algorithms, we need to predict the amount of resources required:

- Memory: how much space is needed?
- Computational time: how fast the algorithm runs

Running Time: The running time of an algorithm refers to the amount of time it takes for the algorithm to complete its execution, given an input.

The running time grows with the size of the input. The input size refers to the number of elements in the input, such as the size of an array, the number of elements in a matrix, the number of vertices, or the number of edges in a graph.

In algorithm analysis, "how long" refers to the number of steps (primitive operations), not real time. These steps are called primitive operations, and they include: Basic math: +, -, *, /; Comparison: if $x > y$; Loops: while, for; Assignments: $x = 5$

If computers were infinitely fast and had infinite memory, any correct method for solving a problem would do. You would probably want your implementation to be within the bounds of good software engineering practice (for example, your implementation should be well-designed and documented), but you would most often use whichever method was the easiest to implement

Different algorithms that solve the same problem often differ dramatically in their efficiency (not only due to differences between hardware and software). For example, two sorting algorithms: insertion sort takes time roughly equal to c_1n^2 to sort n items whereas merge sort takes $c_2n\log n$. When n is very large, the time difference between the algorithms is significant. There are more than 100 million web searches every half hour, and more than 100 million emails sent every minute, hence it is important to use an efficient algorithm.