# CS3230 Chapter 1 - Formulation & Tools for Algorithmic Analysis

Based on lectures by Chang Ee-Chien
Notes taken by Andrew Tan
AY18/19 Semester 1

## 1 Algorithms

An algorithm is a sequence of computational steps that transform a given input into an output.

Given a problem, an effective algorithm would be both correct and efficient in execution, in that it would compute the correct solution from given input instances, and minimizes the time and space needed for computation.

## 2 Asymptotic Notations

### 2.1 Big-O Notation

Let $f(n)$ and $g(n)$ be functions on $\mathbb{Z}_+$. We say that $f(n) \in O(g(n))$ if there exists positive constants $c$ and $n_0$ such that $\forall n > n_0, f(n) \leq cg(n)$.

$$\begin{aligned} O(g(n)) = \{f(n) : \exists c > 0 \text{ and } n_0 > 0 \text{ such that} \\ \forall n > n_0, f(n) \leq cg(n)\} \end{aligned}$$

### 2.2 Big-Omega Notation

Let $f(n)$ and $g(n)$ be functions on $\mathbb{Z}_+$. We say that $f(n) \in \Omega(g(n))$ if there exists positive constants $c$ and $n_0$ such that $\forall n > n_0, f(n) \geq cg(n)$.

$$\begin{aligned} \Omega(g(n)) = \{f(n) : \exists c > 0 \text{ and } n_0 > 0 \text{ such that} \\ \forall n > n_0, f(n) \geq cg(n)\} \end{aligned}$$

### 2.3 Big-Theta Notation

Let $f(n)$ and $g(n)$ be functions on $\mathbb{Z}_+$. We say that $f(n) \in \Theta(g(n))$ if $f(n) \in O(g(n))$ and $f(n) \in \Omega(g(n))O(f(n))$

$$\Theta(f(n)) \equiv O(f(n)) \cap \Omega(f(n))$$

### 2.4 Proving asymptotic bounds

**Lemma 1.**
Consider the two functions $f(n)$ and $g(n)$ where $\forall n \in \mathbb{Z}$, $f(n) > 0$ and $g(n) > 0$, and

$$\lim_{x\to\infty} f(n)/g(n) = c$$

then,
if $c = 0$, then $f(n) \in O(g(n)), f(n) \notin \Omega(g(n))$
if $c = \infty$, then $f(n) \in \Omega(g(n)), f(n) \notin O(g(n))$
if $0 < c < \infty$, then $f(n) \in \Theta(g(n))$

## 2.5 Elementary operations

In analysis, we only take the elementary operations into account. Elementary operations can consist of the following:

- comparison
- assignment
- arithmetic

When performing analysis, the running time of a given algorithm is calculated from the number of elementary operations taken.

## 2.6 Best, worst, and average case

The time and space required by an algorithm may vary between different instances.

### 2.6.1 Worst-case analysis

The performance of the algorithm based on the worst possible input instance.

### 2.6.2 Best-case analysis

The performance of the algorithm based on the best possible input instance.

### 2.6.3 Average-case analysis

the average performance of the algorithm calculated based on all the possible input instances.

# 3 Recurrence Equations

A recurrence equation defines a function, say T(n), recursively. It is solved if we express it explicitly.
To calculate the time complexity of a recursive function, we can apply the Master Theorem.

**Master Theorem**
Let $a \geq 1$ and $b > 1$ be constants, let $f(n)$ be a function, and let $T(n)$ be defined on the nonnegative integers by the recurrence,

$$T(n) = \begin{cases} 0, & \text{if } n = 1 \\ aT(n/b) + f(n), & \text{otherwise} \end{cases}$$

Then,

1. If $f(n) \in O(n^{\log_b(a-\epsilon)})$ for some constant $\epsilon > 0$ then $T(n) \in \Theta(n^{\log_b(a)})$
2. If $f(n) \in \Theta(n^{\log_b(a)})$ for some constant $\epsilon > 0$ then $T(n) \in \Theta(n^{\log_b(a)} \log(n))$
3. If $f(n) \in \Omega(n^{\log_b(a+\epsilon)})$ for some constant $\epsilon > 0$, and if $af(n/b) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large $n$, then $T(n) = \Theta(f(n))$