# Recitation 6: Asymptotics & Analysis

This week we will be studying asymptotics and how to find the asymptotic runtimes of given algorithms. The way we do this is by writing down a ***recurrence relation*** which relates a larger instance of a problem to a smaller instance of that problem. We can then unroll this recurrence relation down to its base case, and count the amount of unrolling that we need to do.

## 0 Review: Big-O, Big-$\Omega$, and Big-$\Theta$

O, $\Omega$, and $\Theta$ are tools that we can use to compare functions to each other, in the same way that we use $\leq$, $\geq$, and $=$ to compare numbers to each other.

Let's recall some definitions:

**Definition 0.1.** Let $f, g : \mathbb{N} \to \mathbb{R}$ be sequences of real numbers. We say that $f(n) \in O(g(n))$ if there is a constant $c$ and a threshold natural number $N$ such that for all $n \geq N$, we have

$$f(n) \leq c \cdot g(n)$$

**Definition 0.2.** Let $f, g : \mathbb{N} \to \mathbb{R}$ be sequences of real numbers. We say that $f(n) \in \Omega(g(n))$ if there is a constant $c$ and a threshold natural number $N$ such that for all $n \geq N$, we have

$$f(n) \geq c \cdot g(n)$$

**Definition 0.3.** Let $f, g : \mathbb{N} \to \mathbb{R}$ be sequences of real numbers. We say that $f(n) \in \Theta(g(n))$ if it holds that both $f \in O(g)$ and $f \in \Omega(g)$.

Recall that $f \in O(g)$ is an asymptotic way of saying "$f \leq g$, up to constant factor". Likewise, $f \in \Omega(g)$ is an asymptotic way of saying that "$f \geq g$, up to constant factor". Finally, $f \in \Theta(g)$ is a way of saying that $f$ and $g$ are within a constant factor of being equal to each other.

**Exercise 0.4.** Prove that $f \in O(g)$ if and only if $g \in \Omega(f)$

**Exercise 0.5.** Prove that $f \in \Theta(g)$ if and only if $g \in \Theta(f)$

# 1 Ordering Runtimes of Algorithms

The runtime of an algorithm is usually going to be a function of its input size, which we will call $n$. So $f(n)$ and $g(n)$ will usually denote the amount of time that it takes some algorithm to terminate, when given an input of length $n$. We want to be able to understand which runtimes are better than others, and what the threshold of "better" is.

For example, suppose we have two algorithms Alg1 and Alg2 which both complete a given task. If Alg1 has a runtime of $f(n) = 1000n^2$ and Alg2 has a runtime of $g(n) = n^3$, then we might want to be able to figure out which one is faster.

We should observe that Alg2 will be faster on smaller inputs, and Alg1 will be faster on larger inputs. The threshold here is $N = 1000$, after which Alg1 becomes faster than Alg2. So we say that $f \in O(g)$ because for all $n > 1000$, $f(n) \leq g(n)$.

## 1.1 Exercise

Order the following functions in increasing growth rate, using Big-O and Big-Theta notation. Use Big-O only when the inequality is strict.
(e.g., $n \in O(n^2)$, $n^2 \in \Theta(n^2 + 1)$, $n^2 + 1 \in O(n^3)$)

$$n^3 + 8n^2, \quad 2n^2\sqrt{n}, \quad 6n^3 - n^2, \quad n^2 + 300n$$

## 1.2 Exercise

Order the following functions in increasing growth rate, using Big-O and Big-Theta notation. Use Big-O only when the inequality is strict.

$$n \ln(3n), \quad 2^{2n}, \quad n^3 + 7, \quad n^3 + 2^{5n}$$

**Note:** The following results from calculus may be useful:

**(Limit Comparison Test)** If the limit $L := \lim_{x\to\infty} \frac{f(x)}{g(x)}$ exists, then:

- if $0 < L < \infty$, then $f(n) \in \Theta(g(n))$
- if $L = 0$, then $f(n) \in O(g(n))$, but $f(n) \notin \Theta(g(n))$
- if $L = \infty$, then $g(n) \in O(f(n))$, but $g(n) \notin \Theta(f(n))$

**(L'Hôpital's rule)** Suppose $g$ and $f$ are both differentiable functions, with either

a. $\lim_{x\to\infty} f(x) = 0$ and $\lim_{x\to\infty} g(x) = 0$; or

b. $\lim_{x\to\infty} f(x) = \pm\infty$ and $\lim_{x\to\infty} g(x) = \pm\infty$

If $\lim_{x\to\infty} \frac{f(x)}{g(x)}$ exists, then $\lim_{x\to\infty} \frac{f(x)}{g(x)} = \lim_{x\to\infty} \frac{f'(x)}{g'(x)}$.

## 2 Analyzing Code

Now that we know how to compare runtimes for various algorithms, we need to actually be able to figure out what those runtimes are, for various algorithms. For the following algorithms, write down a recurrence relation which relates the runtime on an input of size $n$ to the runtime on an input of size smaller than $n$. You do not need to solve the recurrence relation, but you should do so if you can.

For example, the recurrence relation for Linear Search (i.e. looking through a list for a specific element, one-by-one) is

$$f(n) = f(n-1) + 1$$

meaning that it takes Linear Search one more operation to process a list of length $n$ than it takes to process a list of length $n-1$. Unrolling tells us that $f(n) = n$.

### 2.1 Exercise

Give a recurrence relation for the following algorithm which finds the maximum element of a list:

**Algorithm 1** MAX1

```
1: procedure MAX1(Integer Array A)
2:     Len = |A|
3:
4:     if Len = 1 then return A[1]
5:     else if Len = 0 then return −∞
6:
7:     m1 ← MAX1(A[1 : L/2])
8:     m2 ← MAX1(A[L/2 + 1 : L])
9: return m1
```

### 2.2 Exercise

Give a recurrence relation for the following algorithm which finds the maximum element of a list:

**Algorithm 2** MAX2

```
1: procedure MAX2(Integer Array A)
2:     L = |A|
3:
4:     if L = 1 then return A[1]
5:     else if L = 0 then return −∞
6:
7:     m ← MAX2(A[1 : L − 3])
8:
9:     for i ← L; i ≥ 1; i ← i − 1 do
10:        if m ≥ A[i] then m ← A[i]
       return m
```

$$\log_b(mn) = \log_b(m) + \log_b(n), \qquad \log_b\left(\frac{m}{n}\right) = \log_b(m) - \log_b(n)$$

$$\log_b(n^p) = p\log_b(n), \qquad \log_b(n) = \frac{log_p(n)}{log_p(b)}$$

a. $\log_3(x^2y^3) - \log_2(\sqrt{z})$

b. $\log_3\left(\frac{(t+5)}{(t^4)}\right)$

c. $\log_2(2^x) + \log_4(\frac{x}{5y})$

Assume $a = X^n$, $b = X^m$

$a^{\log b} = (X^n)^{\log X^m}$

$= (X^n)^{m \log x}$

$= X^{n \cdot m \log x}$

$= (X^m)^{n \log x}$

$= (X^m)^{\log x^n}$

$= b^{\log a}.$