1 Asymptotic Notation

In this class, we care a lot about the runtime of algorithms. However, we don’t care too much about concrete performance on small input sizes (most algorithms do well on small inputs). Instead we want to compare the *long-term (asymptotic)* growth of the runtimes. To this effect, we define the following notation for two functions $f(n), g(n) \geq 0$:

- $f(n) = \mathcal{O}(g(n))$ if there exists a $c > 0$ where after large enough $n$, $f(n) \leq c \cdot g(n)$. ‘Asymptotically, $f$ grows at most as much as $g$’.
- $f(n) = \Omega(g(n))$ if $g(n) = \mathcal{O}(f(n))$. ‘Asymptotically, $f$ grows at least as much as $g$’.
- $f(n) = \Theta(g(n))$ if $f(n) = \mathcal{O}(g(n))$ and $g(n) = \mathcal{O}(f(n))$. ‘Asymptotically, $f$ and $g$ grow the same’.

If we compare this to the order on the numbers, $\mathcal{O}$ is a lot like $\leq$, $\Omega$ is a lot like $\geq$, and $\Theta$ is a lot like $=$ (except all are with regard to asymptotic behavior).

(a) For each pair of functions $f(n)$ and $g(n)$, state whether $f(n) = \mathcal{O}(g(n))$, $f(n) = \Omega(g(n))$, or $f(n) = \Theta(g(n))$. For example, for $f(n) = n^2$ and $g(n) = 2n^2 - n + 3$, write $f(n) = \Theta(g(n))$.

(i) $f(n) = n$ and $g(n) = n^2 - n$
**Solution:** $f(n) = \mathcal{O}(g(n))$

(ii) $f(n) = n^2$ and $g(n) = n^2 + n$
**Solution:** $f(n) = \Theta(g(n))$

(iii) $f(n) = 8n$ and $g(n) = n \log n$
**Solution:** $f(n) = \mathcal{O}(g(n))$

(iv) $f(n) = 2^n$ and $g(n) = n^2$
**Solution:** $f(n) = \Omega(g(n))$

(v) $f(n) = 3^n$ and $g(n) = 2^{2n}$
**Solution:** $f(n) = \mathcal{O}(g(n))$

(b) For each of the following, state the order of growth using $\Theta$ notation, e.g. $f(n) = \Theta(n)$.

(i) $f(n) = 50$
**Solution:** $f(n) = \Theta(1)$

(ii) $f(n) = n^2 - 2n + 3$
**Solution:** $f(n) = \Theta(n^2)$

(iii) $f(n) = n + \cdots + 3 + 2 + 1$
**Solution:** $f(n) = \frac{n(n+1)}{2} = \Theta(n^2)$

(iv) $f(n) = n^{100} + 1.01n$
**Solution:** $f(n) = \Theta(1.01^n)$

(v) $f(n) = n^{1.1} + n \log n$
**Solution:** $f(n) = \Theta(n^{1.1})$
(vi) \( f(n) = (g(n))^2 \) where \( g(n) = \sqrt{n} + 5 \)

Solution: \( f(n) = \Theta(n) \)

Solution: In general, we can observe the following properties of \( \mathcal{O}/\Theta \) from this:

- If \( d > c \), \( n^c = \mathcal{O}(n^d) \), but \( n^c \neq \Omega(n^d) \) (this is sort of saying that \( n^d \) grows strictly faster than \( n^c \)).
- \( \mathcal{O} \) only cares about “highest-growing” terms. For example, \( n^2 + n = \Omega(n^2) \).
- \( \mathcal{O} \) does not care about leading constants. For example \( 50n = \Theta(n) \).
- Any exponential with base \( > 1 \) grows more than any polynomial.
- The base of the exponential matters. For example, \( 3^n = \mathcal{O}(4^n) \).
- If \( d > c \), \( n^c \log n = \mathcal{O}(n^d) \).

2 Asymptotics and Limits

If we would like to prove asymptotic relations instead of just using them, we can use limits. Specifically, if \( f(n), g(n) \geq 0 \):

- If \( \lim_{n \to \infty} \frac{f(n)}{g(n)} = 0 \), then \( f(n) = \mathcal{O}(g(n)) \).
- If \( \lim_{n \to \infty} \frac{f(n)}{g(n)} = c \), for some \( c > 0 \), then \( f(n) = \Theta(g(n)) \).
- If \( \lim_{n \to \infty} \frac{f(n)}{g(n)} = \infty \), then \( f(n) = \Omega(g(n)) \).

Note that these are all sufficient conditions involving limits, and are not true definitions of \( \mathcal{O} \), \( \Theta \), and \( \Omega \).

(a) Prove that \( n^3 = \mathcal{O}(n^4) \).

Solution:

\[
\lim_{n \to \infty} \frac{n^3}{n^4} = \lim_{n \to \infty} \frac{1}{n} = 0
\]

So \( f(n) = \mathcal{O}(g(n)) \)

(b) Find an \( f(n), g(n) \geq 0 \) such that \( f(n) = \mathcal{O}(g(n)) \), yet \( \lim_{n \to \infty} \frac{f(n)}{g(n)} \neq 0 \).

Solution: Let \( f(n) = 3n \) and \( g(n) = 5n \). Then \( \lim_{n \to \infty} \frac{f(n)}{g(n)} = \frac{3}{5} \), meaning that \( f(n) = \Theta(g(n)) \). However, it’s still true in this case that \( f(n) = \mathcal{O}(g(n)) \) (just by the definition of \( \Theta \)).
(c) Prove that for any $c > 0$, we have $\log n = O(n^c)$.

*Hint:* Use L'Hôpital’s rule: If $\lim_{n \to \infty} f(n) = \lim_{n \to \infty} g(n) = \infty$, then $\lim_{n \to \infty} \frac{f(n)}{g(n)} = \lim_{n \to \infty} \frac{f'(n)}{g'(n)}$ (if the RHS exists)

*Solution:* By L'Hôpital’s rule, 

$$\lim_{n \to \infty} \frac{\log n}{n^c} = \lim_{n \to \infty} \frac{n^{-1}}{cn^{c-1}} = \lim_{n \to \infty} \frac{1}{cn^{c}} = 0$$

Therefore, $\log n = O(n^c)$.

(d) Find an $f(n), g(n) \geq 0$ such that $f(n) = O(g(n))$, yet $\lim_{n \to \infty} \frac{f(n)}{g(n)}$ does not exist. In this case, you would be unable to use limits to prove $f(n) = O(g(n))$.

*Solution:* Let $f(x) = x(\sin x + 1)$ and $g(x) = x$. As $\sin x + 1 \leq 2$, we have that $f(x) \leq 2 \cdot g(x)$ for $x \geq 0$, so $f(x) = O(g(x))$.

However, if we attempt to evaluate the limit, $\lim_{x \to \infty} \frac{x(\sin x + 1)}{x} = \lim_{x \to \infty} \sin x + 1$, which does not exist (sin oscillates forever).

3 Euclid GCD

*Euclid's algorithm* is an algorithm that computes the greatest common divisor between two integers $a, b \geq 0$:

```java
function EUCLID-GCD(a, b)
    if a > b then
        return EUCLID-GCD(b, a)
    else if a = 0 then
        return b
    else
        return EUCLID-GCD(b mod a, a)
```

As an introduction to the types of things we’ll be doing in CS 170, let’s analyze this algorithm’s runtime and correctness.

(a) Let $n = a + b$. We will try to write this algorithm’s runtime as $O(f(n))$ for some function $f$, counting every integer operation (including `mod`) as one operation. Starting with EUCLID-GCD($a, b$) where $a < b$, what will the arguments be after two recursive calls?

*Solution:*

If we start with EUCLID-GCD($a, b$), after one recursive call we have EUCLID-GCD($b \ mod \ a, a$). After two recursive calls, we have EUCLID-GCD($a \ mod \ (b \ mod \ a), b \ mod \ a$).

(b) Assume still that $a < b$, and let $n'$ be the sum of $a$ and $b$ after two recursive calls. Consider two cases: $2a \leq b$ and $2a > b$. In each case, show that after two recursive calls, $n' \leq 2n$. What runtime do we end up with, in the form $O(f(n))$?
Hint: For each case, show that \( n \geq c \) and \( n' \leq c' \) for some \( c, c' \). If \( c' \leq \frac{2}{3}c \), you are finished.

Hint: Use the properties that: (1) \( b \mod a \leq a \), (2) If \( b \geq a \), then \( b \mod a \leq b - a \).

Solution:

Case 1: Let \( 2a \leq b \). Then \( n = a + b \geq 3a \).

As \( b \mod a \leq a \), we have \( a \mod (b \mod a) \leq (b \mod a) \leq a \) as well.

So \( n' \leq 2a \).

Case 2: Let \( 2a > b \). Then \( n = a + b \geq \frac{3}{2}b \).

As \( b \mod a \leq b - a \leq \frac{1}{2}b \), we have \( a \mod (b \mod a) \leq (b \mod a) \leq \frac{1}{2}b \) as well.

So \( n' \leq b \).

We have shown that after every two calls, \( n' \leq \frac{3}{2}n \). This kind of fractional behavior in input size leads to a \( O(\log n) \) runtime.

Notice that the base of the log isn’t actually important! Because \( \log_{d} x = \frac{\log_{c} x}{\log_{c} d} \), all bases of log are related by a constant, which \( O \) ignores.

(c) Let’s look at correctness. We can prove that this algorithm returns the correct answer using a proof by induction. Note that the result of the algorithm does not depend on if \( a > b \) or \( a \leq b \), so let’s focus on the case that \( a \leq b \), and the other case follows.

As our base case, let \( a = 0 \). Argue that \( \text{EUCLID-GCD}(0, b) \) returns the correct answer for all \( b \).

Solution: For all integers \( d, d \mid 0 \). Then the GCD is the largest integer \( d \) such that \( d \mid b \), which is \( b \). So the \( \text{EUCLID-GCD} \) acts correctly in this case.

(d) Now for the inductive step: Assume that \( \text{EUCLID-GCD}(a, b) \) computes the right answer for all \( a \leq b \leq k - 1 \). Show that \( \text{EUCLID-GCD}(a, b) \) computes the right answer for all \( a \leq b = k \).

Hint: Use the property that \( d \mid a \) and \( d \mid b \iff d \mid a \) and \( d \mid (b \mod a) \).

Solution:

By the property in the hint, the set of common divisors of \( \{a, b\} \) is the same as the set of common divisors of \( \{b \mod a, a\} \), so the greatest of these common divisors must also be the same.

If you are interested in why the property is true, think about the fact \( b = \ell \cdot a + (b \mod a) \) for some \( \ell \). How does this show \( d \mid a \) and \( d \mid b \iff d \mid a \) and \( d \mid (b \mod a) \)?

It remains to be shown that \( \text{EUCLID-GCD}(b \mod a, a) \) is calculated correctly. There are two cases:
**Case 1:** If $a = b$, then $b \mod a = 0$. By the base case, $\text{EUCLID-GCD}(0, a)$ is calculated correctly.

**Case 2:** If $a < b$, then $a \leq k - 1$. By the inductive hypothesis, $\text{EUCLID-GCD}(b \mod a, a)$ is calculated correctly.