

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 $g(n) = n^2 - n$
(ii) $f(n) = n^2$ and $g(n) = n^2 + n$
(iii) $f(n) = 8n$ and $g(n) = n \log n$
(iv) $f(n) = 2^n$ and $g(n) = n^2$
(v) $f(n) = 3^n$ and $g(n) = 2^{2n}$

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

(i) $f(n) = 50$
(ii) $f(n) = n^2 - 2n + 3$
(iii) $f(n) = n + \cdots + 3 + 2 + 1$
(iv) $f(n) = n^{100} + 1.01n$
(v) $f(n) = n^{1.1} + n \log n$
(vi) $f(n) = (g(n))^2$ where $g(n) = \sqrt{n} + 5$
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 \( n^3 = \mathcal{O}(n^4) \).

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

(c) Prove that for any \( c > 0 \), we have \( \log n = \mathcal{O}(n^c) \).

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

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

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

```plaintext
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 $\text{mod}$) as one operation. Starting with $\text{EUCLID-GCD}(a, b)$ where $a < b$, what will the arguments be after two recursive calls?

(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, $n' \leq \frac{3}{2}n$. 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 $c' \leq \frac{3}{2}c$, you are finished.

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

\begin{verbatim}
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)
\end{verbatim}

(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 \( b \).

(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 \( a \leq b = k \).

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