Note: Your TA may not get to all the problems. This is totally fine, the discussion worksheets are not designed to be finished in an hour. The discussion worksheet is also a resource you can use to practice, reinforce, and build upon concepts discussed in lecture, readings, and the homework.

1 Asymptotic Notation Intro

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.

Asymptotic Notation: 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 $\leq$, Ω is a lot like $\geq$, and Θ 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 $g(n) = n^2 - n$ $f(n) = O(g(n))$

(ii) $f(n) = n^2$ and $g(n) = n^2 + n$ Θ

(iii) $f(n) = 8n$ and $g(n) = n \log n$ O

(iv) $f(n) = 2^n$ and $g(n) = n^2$ Ω

(v) $f(n) = 3^n$ and $g(n) = 2^{2n}$ Θ

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

(i) $f(n) = 50$ $\Theta(1)$

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

(iii) $f(n) = n + \cdots + 3 + 2 + 1$ $\Theta(n^2)$

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

(v) $f(n) = n^{1.1} + n \log n$ $\Theta(n^{1.1})$

(vi) $f(n) = (g(n))^2$ where $g(n) = \sqrt{n} + 5$

$\underline{(\sqrt{n}+5)^2}$ $\underline{\Theta(n)}$

2 Asymptotics and Limits

If we would like to prove asymptotic relations instead of just using them, we can use limits.

Asymptotic Limit Rules: 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 $\mathcal{O}$, Θ, and Ω.

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

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

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

$f(n) = g(n) \rightarrow \lim_{n\to\infty} \frac{f(n)}{g(n)} = 1$

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

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

(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 $f(n) = \mathcal{O}(g(n))$.

$g(n) = (n)$

$$f(n) = \begin{cases} n & n \text{ is even} \\ n+1 & n \text{ is odd.} \end{cases}$$

$f(x) = x(\sin x + 1)$

$g(x) = x$

3 Runtime and Correctness of Mergesort

In general, this class is about design, correctness, and performance of algorithms. Consider the following algorithm called *Mergesort*, which you should hopefully recognize from 61B:

```
function MERGE($A[1,\dots,n]$, $B[1,\dots,m]$)
    $i, j \leftarrow 1$
    $C \leftarrow$ empty array of length $n+m$
    while $i \le n$ or $j \le m$ do
        if $i \le n$ and ($j > m$ or $A[i] < B[j]$) then
            $C[i+j-1] \leftarrow A[i]$
            $i \leftarrow i+1$
        else
            $C[i+j-1] \leftarrow B[j]$
            $j \leftarrow j+1$
    return $C$
function MERGESORT($A[1,\dots,n]$)
    if $n \le 1$ then return $A$
    $mid \leftarrow \lfloor n/2 \rfloor$
    $L \leftarrow$ MERGESORT($A[1,\dots,mid]$)
    $R \leftarrow$ MERGESORT($A[mid+1,\dots,n]$)
    return MERGE($L, R$)
```

Recall that MERGESORT takes an arbitrary array and returns a sorted copy of that array. It turns out that MERGESORT is asymptotically optimal at performing this task (however, other sorts like Quicksort are often used in practice).

(a) Let $T(n)$ represent the number of operations MERGESORT performs given a array of length n. Find a base case and recurrence for $T(n)$, use asymptotic notation.

$$T(n) = \begin{cases} 1 & n = 1 \\ 2T(\frac{n}{2}) + \underline{O(n)} & n > 1 \end{cases}$$

(b) Solve this recurrence. What asymptotic runtime do you get?

$a = 2 \quad b = 2 \quad d = 1$

$\underline{\log_b a = 1} \qquad \underline{n \log n}.$

(c) Consider the correctness of MERGE. What is the desired property of C once MERGE completes? What is required of the arguments to MERGE for this to happen?

A[1.. n] is sorted $a_1 \le a_2 \le a_3 \le \dots a_n$.

A, B are sorted . .

(d) Using the property you found for MERGE, show that MERGESORT is correct.

C is sorted

Use Induction to proof.

bottom up