Notes Week 6

- Read Chapters 22, Chapter 9 for Week 7

Rules for Big O Notation (Upper Bounds)

- **Transitivity**: If $f = O(g)$ and $g = O(h)$ then $f = O(h)$
- **Sums**: If $f_1 = O(g_1)$ and $f_2 = O(g_2)$ then $f_1(n) + f_2(n) = O(max(f, g))$
- **Products**: If $f_1 = O(g_1)$ and $f_2 = O(g_2)$ then $f_1 \times f_2 = O(g_1 \times g_2)$
- If the largest term is a polynomial of degree k, then the whole thing is $\Theta(n^k)$
- $log^k n = O(n) \rightarrow$ a log raised to any constant power is still just $O(n)$
- $\lim n \rightarrow \infty \frac{f(n)}{g(n)} \rightarrow$ If the limit is:
  - $\mathbf{0} \rightarrow f = o(g) \rightarrow g(n) > f(n) \rightarrow g$ dominates f
  - $c \neq 0 \rightarrow f = \Theta(g(n)) \rightarrow g(n) = \Theta(f(n))$
  - $\infty \rightarrow g(n) = o(f(n))$
- Example: $f(n) = \frac{n}{log(n), g(n) = n^{\frac{1}{2}} log^2 n}$

Rules for Big Ω Notation (Lower Bounding)

- **Comparison-based Solution** $\omega(nlogn) \rightarrow$ it's been proven that comparison-based sorting cannot be faster
- Some problems are $\Omega(2^n) \rightarrow$ Super slow

Quick Maths

- $b^{log_b a} = a$
- $a^{log_b n} = n^{log_b a}$
- Any exponential function dominates any polynomial
  - Exponential function $\rightarrow 2^n \rightarrow$ variable in the exponent
  - Polynomial function $\rightarrow n^{234} \rightarrow$ exponent is a constant
- $\sum(ca_k + b_k) = c\sum(a_k) + \sum(b_k)$

Recurrences - Recurrence Relations

- A **Recurrence** is used when dealing with recursion.
- Merge Sort:

Case	**Formula**	**Big O**
base case	$T(n) = \Theta(1)$	$\Theta(1)$
not base case	$2T(\frac{n}{2}) + \Theta(n)$	?

- How does $2T(\frac{n}{2})$ explain the recurrence of merge sort?
  - Each division splits the array in half
  - There are two pieces in each call, one for each half
  - So the array is split in half, and each is operated on recursively
- So what is the O(merge-sort)
  - $T(n) = 2T(\frac{n}{2}) + n \rightarrow$ Merge Sort Recurrence
  - Solve using iteration method: "Unfold the recurrence"
    1. If there's a constant, write out that constant with square brackets empty to indicate something needs done
    $$2[\quad] + n \tag{1}$$
    2. Plug the $\frac{n}{2}$ back into the original
    $$2[2T(\frac{n}{4}) + \frac{n}{2}] + n \tag{2}$$
    3. Multiply it out
    $$4T(\frac{n}{4}) + 2n$$
    4. Take the previous iteration (2) and repeat
    $$4[2T(\frac{n}{8}) + \frac{n}{4}] + 2n \tag{3}$$
    5. Simplify
    $$8T(\frac{n}{8}) + 3n$$
    6. Repeat the process until you can spot the general case. The next iteration would be
    $$16T(\frac{n}{16}) + 4n \tag{4}$$
    7. Write down the general case in terms of k
    $$2^k(\frac{n}{2^k}) + kn$$
    8. How many times does the recurrence occur? $k = log_2 n$
    9. Plug in k
    $$2^{log_2 n}T(\frac{n}{2^{log_2 n}}) + nlog_2 n$$
    10. Simplify
    $$nT(1) + nlog_2 n = O(nlog_2 n)$$