Defining Ω Θ O

1. $f(n) = O(g(n))$ iff $\exists c > 0$ st $f(n) \leq cg(n), \forall n \geq n_0$
2. $f(n) = \Omega(g(n))$ iff $\exists c > 0$ st $f(n) \geq cg(n), \forall n \geq n_0$
3. $f(n) = \Theta(g(n))$ iff $\exists c_1, c_2 > 0$ st $c_1 g(n) \leq f(n) \leq c_2 g(n) \forall n \geq n_0$ AKA asymptotically the same
4. $f(n) = o(g(n))$ iff $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$ (Use L'Hopital's Rule to achieve either 0 or ∞.)
5. $f(n) = \omega(g(n))$ iff $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$ (Use L'Hopital's Rule to achieve either 0 or ∞.)

Four rules for determining Asymptotic Notations:

1. Multiplicative constants can be omitted: $14n^2$ becomes n^2.
2. n^a dominates n^b if $a > b$: for instance, n^2 dominates n.
3. Any exponential dominates any polynomial: 3^n dominates n^5 (it even dominates 2^n).
4. Likewise, any polynomial dominates any logarithm: n dominates $(logn)^3$. This also means, for example, that n^2 dominates $nlogn$.

Limits, sums and Sterling's Formula

1. Sterling basically proves that $\log n! \approx n \log n$
2. $\lim_{n\to\infty} \frac{n^k}{2^n} = 0$ which implies $2^n > n^k$

Master's Theorem: Applies only to $T(n) = aT(\frac{n}{b}) + O(n^d)$

1. $T(n) = O(n^d)$ if $d > \log_b a$
2. $T(n) = O(n^d \log n)$ if $d = \log_b a$
3. $T(n) = O(n^{\log_b a})$ if $d < log_b a$

Sorting: Binary Search, Mergesort, Quicksort (SPLIT), Heapsort

1. Mergesort - Runtimes: O(n lg n) (Avg n Worst)
   [5, 1, 2, 4, 7, 9] (original)
   [5, 1, 2] [̂4, 7, 9] (splitting)
   [5, 1], [2] [̂4, 7], [9] (splitting)
   [1, 2, 5] [̂4, 7, 9] (merging)
   [1, 2, 4, 5, 7, 9] (merging)
2. Quicksort - Runtimes: O(n lg n) (Avg), O (n^2) (Worst)
   [4, 2, 3, 1]
   [4 (piv), 2 (¡), 3 (¡), 1 (¡)]
   [2, 3, 1, 4]
   [2 (piv), 3 (¿), 1 (¡), 4 (¿)]
   [2 (piv), 1 (¡), 3 (¿), 4 (¿)]
   [1, 2 (piv), 3 (¿), 4 (¿)]
   [1, 2, 3 (piv), 4 (¿)]
   [1, 2, 3, 4] Recurrence Relation:
   $T(n) = \frac{1}{n} \sum_{i=1}^{n-1} (T(i) + T(n-i)) + (n-1)$
   For worst case: $T(n) = O(n) + T(n-1) = O(n) + T(n-1)$ **For average case:** $T(n) = O(n) + 2T(\frac{n}{2})$

3. Heapsort - O(n lg n) (Avg n Worst)
   Recursive Algorithm: Take array A at index i
   BuildHeap(A,i)
   FixHeap(A,2)
   Non-recursive:
   For $i = \frac{n}{2}$ down to 1
   FixHeap(A,i)
   Basically, this means to build a binary tree with the array elements. Then, build a max heap. Pop the root off and put it in the nth space of the array. Make the next highest number the root and repeat until the array is sorted.
   The above satisfies the relation T(n) $\leq$ 2T($\frac{n}{2}$) + $O(log(n)) = O(n)$.

4. Binary Search-Assuming $n = 2^k - 1$:
   $k - \frac{1}{2}$ (avg)
   $O(\log n + 1)$(worst)

Polynomials and evaluation, Horner method and related stuff

1. Get matrix into augmented matrix format $Ax = b$ where A is the computed matrix and x represents the n coefficients of the n degree polynomial and b is the y values of y ordered pairs.

2. Horner's Method basically makes for recursive multiplication of polynomials. For example, a polynomial of degree 8 would take 8 multiplications but it could be done more efficiently. If we know what x^2 is, just multiply x^2 3 more times. This would save 2 more multiplications.

Integer Multiplication

1. For x and y Decompose x using a = high ceiling(n/2) bits, b = low floor(n/2) bits and y similarly w/ c and d. Form $w1 = a + b$, $w2 = c + d$, $u = w1 * w2$, $v = ac$, $w = bd$. Solve $xy = 2^n(v) + (2^{\frac{n}{2}} * (u - v - w)) + w$

Useful summation formulas

$\sum_{i=0}^{n} i = \frac{n*(n+1)}{2} = \Theta(n^2)$ $\quad$ $\sum_{i=0}^{n} i^2 = \frac{n*(n+1)(2n+1)}{6} = \Theta(n^3)$
$\sum_{i=0}^{n} i^3 = (\frac{n*(n+1)}{2})^2 = \Theta(n^4)$ $\quad$ $\sum_{i=0}^{n} x^i = \frac{x^{n+1}-1}{x-1}$
$\sum_{i=0}^{n} 2^i = \frac{2^{k+1}-1}{2-1} = 2^{k+1} - 1$ $\quad$ $\sum_{i=1}^{n} i * 2^i = (1 * 2) + (2 * 2^2) + (3 * 2^3) + ... + (k * 2^k)$

Divide and Conquer Advice

Generalize to T(n) = aT($\frac{n}{b}$) + D(n) + C(n): a = num subproblems n/b = size subproblems D(n) = time to divide, C(n) = time to combine.

OTHER EXAMPLES

1. Asymptotics! $f(n) = n!$ $g(n) = 2^n$

$$\text{When } c = 1 \text{ and } n = 200,\ f(n) \leq cg(n)$$
$$\text{Therefore, } f(n) = \Omega(g(n))$$

2. Recurrence relation! Use the fact that $\sum_{i=1}^{m} i2^i = (m-1)2^{m+1} + 2$
   $T(n) = 2T(\sqrt{n}) + \log\log n$
   Let $S(k) = T(2^k)$
   $S(k) = 2S(\frac{k}{2}) + logk$
   $S(2^m) = 2S(2^{m-1}) + m$
   $2^m S(1) + \sum_{i=1}^{m} 2^i - \sum_{i=1}^{m} i2^i$
   $2^m + m(2^{m+1} - 2) - (m-1)2^{m-1} + 2$
   $\Theta(\log n)$
   IT'S WEDNESDAY MY DUDES