Table 1: Increasing Order Of Growth

Type	Big-Oh	Asymptotic Order
Logarithmic:	$O(logn)$	$\log n$
Poly Logarithmic:	$O((logn)^2), O((logn^3), ...$	$(logn)^2 \leq (logn)^3 \leq (logn)^4 \leq ...$
Fractional Power:	$O(n^c) for 0 < c < 1$	$n^{0.1} \leq n^{0.2} \leq n^{0.3} \leq ...$
Linear:	$O(n)$	n
$nlogn$ time:	$O(nlogn)$	$nlogn \leq n(logn)^2 \leq n(logn)^3 \leq ...$
Polynomial Time:	$O(n^a) for a > 1$	$n^2 \leq n^3 \leq ...$
Exponential Time:	$O(2^n)$	$1.5^n \leq 2^n \leq 3^n ...$

Definitions

Big O Notation: Will give us an upper bound for function f(n) when n is very large/asymptotically. This is used extensively to describe the worst case scenario for the number of operations used by an algorithm. The notation used in the definition is: O(g(n)) which is read "big-oh of g of n"

Definition:

Let $f(n) : N \to R+$. For a given function $g(n)$, we say that $f(n)$ is $O(g(n))$ if there are constants C and k such that:
$f(n) \leq Cg(n)$ for all $n > k$

$log_2 n \leq n$ for all $n \geq 1$

This inequality is true for all logarithms for any base b > 0

For a base $b > 1$ and any exponent $a > 0$ we have that $log_b n \leq n^a$ for large enough n (for $n \geq k$ for some constant k)

Big Omega Notation Ω: The notation Big-ω is defined similar to that of Big-O but for a lower bound.

Definition:

Let $f(n)$ and $g(n)$ be functions on the natural numbers to the positive real numbers. We say that $f(n)$ is $\Omega(g(n))$ if there is a constant C such that $f(n) \geq Cg(n)$ whenever $n > k$

Big Theta Notation θ: Big-θ Notation is used to specify the function f(n) that is sandwiched between multiples of g(n). If being asked for Theta one must prove for Big O AND Big Omega **Definition:**

Let $f(n)$, and $g(n)$ be functions from the natural numbers to the positive reals. If $f(n)$ is $O(g(n))$ and $f(n)$ is $\Omega(g(n))$ then we say that $f(n)$ is of the order of $g(n)$ and use the notation $\theta(g(n))$

Formulas

Summation Formula:

$$\sum_{k=1}^{n} k = \frac{1}{2n}(n+1) = \theta(n^2)$$

$$\sum_{k=1}^{n} 10k = \frac{10n(n+1)}{2} = \theta(n^2)$$

Finding Summation Lower Bounds:

Example: $\sum_{k=1}^{n} k^2 * n^{22}$

Swap k=1 for half the list $\frac{n}{2}$ in this case k=1 > n is now $\frac{n}{2}$ > n. $= \sum_{k=n/2}^{n} (\frac{n}{2})^2 * n^{22}$

Further Processed: $\sum_{k=n/2}^{n} \frac{n^2}{4} * n^{22}$

Further Process the remainders which equal to: $\sum_{k=n/2}^{n} \frac{n^{24}}{4}$

Multiply The Initial Swapped Lower Summation To the newly calculated values: $\sum_{k=n/2}^{n} \frac{n^{24}}{4} * \frac{n}{2} = \frac{n^{25}}{8}$

LOG N Formula: Log N : $50 * Logn^3 <=> 150 * logn$

Master Method Formula: $T(n) = aT\frac{n}{b} + f(n)$
Master Method Requirements:
1. $a \geq 1$ (Has to be at least One, which means it Must recurse at least once)
2. $b > 1$ Has to be at least one (Otherwise T(n) = ∞) 3. a and b are O(1)
4. $f(n) > oforn > n_o$
Master Method processing f(n) goes to the root of the tree and branch out as many times as necessary
each time you branch out, you branch out as T(n/b).
each time you branch out per level $f(n) > a * f\frac{n}{b} > a^2 * f\frac{n}{b^2} > a^i * f\frac{n}{b^i}$ height of tree determined only by information inside master method formula $\frac{n}{b}$ if height is $\frac{n}{b}$ then we are going to have $h = Log_b n$ levels. How many times can you take N and Divide by B. how many leafs? look at level formula above $a^i * f(\frac{n}{b^i})$ process Number of leaves with $a^h = a^{logbn} = n^{log_b a}$
***SWAP A and N ***
Per Leaf processing = $\Theta(_n log_b a)$ per leaf
each leaf takes constant time to solve.