Comparing Time Complexity

Given 2 algorithms, which is faster?

Method 1: factor out common terms

n^2 and n^3: factor out n^2 on both sides to get $n^2 \in O(n^3)$

Method 2: take the log of both sides

2^n and n^2 => $\lg 2^n = n$ and $\lg n^2 = 2\lg n$ $\lg n \in O(n)$, so it must be that $n^2 \in O(2^n)$

Method 3: Try substituting very large value of n

$n^{3/2}$ and $n\lg n$ Factor out n on both sides and we get $n^{1/2}$ and $\lg n$

Take lg of both sides and we get $\lg n^{1/2} = ½(\lg n)$ and $\lg\lg n$

Substitute 2^{1024} for n and we get

(½ *1024) = 512 and $\lg\lg 2^{1024} = \lg 1024 = 10$

So, as n gets very large, $n\lg n \in O(n^{3/2})$

Handy Asymptotic Facts

a) If T(n) is a polynomial function of degree k, then $T(n) = O(n^k)$.

b) $\log^k n = (\log n)^k = O(n)$ Any poly-log func of n grows more slowly than any linear func of n.

c) $n^b = o(a^n)$ for any constants $a > 1$, $b > 0$ polys grow more slowly than expos

d) $n! = o(n^n)$ $1*2*3*...*n < n*n*...*n$

e) $n! = \omega(2^n)$ $1*2*3*...*n > 2*2*...*2$ (n times)

f) $\lg(n!) = \theta(n\lg n)$

g) Given an expression like O(n + m), the actual time is the larger of n and m.