Algorithm Analysis Treatise

Mike Maxim

September 14, 2002

Big O Notation.

1.) $O(n^2)$
2.) $O(n)$
3.) $O(n^2)$
4.) $O(n^2)$
4.) $f(n) = O(g(n))$ and $g(n) = \Omega(f(n))$

Running Time Analysis.

1.) Recurrence: $T(1) = 1$ $T_n = T_{n/2} + O(1)$
Solution (Backsubstitute for pattern): $T_n = T_{n/4} + 2 => T_n = T_{n/2^k} + k$
Let $k = \log_2 n$ gives: $T_n = T_1 + \log_2 n = 1 + \log_2 n$ which is $O(\log n)$

2.) Recurrence: $T(1) = 1$, $T_n = 2T_{n/2} + O(n)$
Solution (Backsubstitute for pattern): $T_n = 4T_{n/4} + 2n => T_n = 2^k T_{n/2^k} + kn$
Let $k = \log_2 n$ gives: $T_n = nT_1 + n\log_2 n = n + n\log_2 n$ which is $O(n \log n)$

Induction.

1.) Base Case: T consists of a single root. $2^{0+1} - 1 = 1$.
IS: Let T be a perfect binary tree of some height $H > 0$. Consider the left and right subtrees of H and let them be called L and R. Note L and R are two perfect binary trees of height $H - 1$. Because of this, we can apply the IH to both L and R and add them together with the root to get the total node count in T. Doing this we get:

$$Nodes(T) = Nodes(L) + Nodes(R) + 1 = (2^{(H-1)+1} - 1) + (2^{(H-1)+1} - 1) + 1$$
$$Nodes(T) = 2(2^H - 1) + 1$$
$$Nodes(T) = 2^{H+1} - 1$$

So now by induction the proof is complete.