How do we determine the time analysis for an algorithm that calculates the height at each node? There are two algorithms to do this. The first visits each node but once and thus its time analysis is clearly $O(n)$. The second one, uses the height function at each node. In this section, we will analyze the second algorithm. At each node $V$ visited, the height is calculated. This means visiting each node in $V$’s subtree.

Assume that the tree is full and has 127 nodes. The number of nodes visited is $127 + 2 \times 63 + 4 \times 31 + 8 \times 15 + 16 \times 7 + 32 \times 3 + 64 \times 1$, where 2, 4, 8, etc. represent the number of subtrees at a given level starting with the first generation of children. This sum equals $127 + 126 + 124 + 120 + 112 + 96 + 64$ which is the same as $N + (N - 1) + (N - 3) + (N - 7) + (N - 15) + (N - 31) + (N - 63)$, where $N$ represents the number of nodes. This is what we’d expect since at each level in order to determine the number of nodes at that level and below, you must subtract the number of nodes above it from $N$. The number of times $N$ appears in this sum is the height of the tree minus one, or $\log N - 1$. So with a minimum loss of precision, we will represent this by $\log N$. The sum becomes $N\log N - (1 + 3 + 7 + 15 + 31 + 63)$.

Generalizing this result to a full tree of $N$ nodes, we find that the number of nodes visited is $N\log N - \sum_{k=0}^{\log N} (2^k - 1)$.

But $\sum_{k=0}^{\log N} (2^k) = 2^{\log N + 1} - 1$ and $\sum_{k=0}^{\log N} (-1) = -\log N$. Also $2^{\log N} = N$. So the analysis yields a complexity of $(N + 1)\log N - 2N$ or $O(N\log N)$. 