buildHeap() Time Bound
Mike Maxim - 15211 Section G

February 15, 2002

Here is the proof about the time complexity of buildHeap() as discussed in recitation.

The first analysis involves making the claim that the farthest a node has to "percolate" is the height of the heap, or $\log n$ if n is the total amount of nodes in the heap. So for each node it travels a possible $\log n$ so the total time is $O(n \log n)$.

We refined that by noting that not every node actually goes $\log n$ distance during percolation. For any given level $0 \leq i \leq n$, in fact the node can travel a distance at most i. So the strategy is to get an expression for how long it takes to percolate a level i, and sum up this expression over all levels,or all $\log n$ of them. So first we need to discover how many nodes are on level i. In a normal BST, the levels are labelled with 0 being at the root and increasing the further down the tree. However here we have reversed that, and instead the root is at level $\log n$. So for a normal tree the number of nodes at a level i is 2^i, but in the reverse labelling we must divide the total nodes by 2^i to switch it around to end up with the expression $\frac{n}{2^i}$ as the total amount of nodes on level i. Again, this has nothing to do with the structure of the tree, just how we label the levels. So for each node on level i it travels at most a distance of i bringing the total cost of level i to $i(\frac{n}{2^i})$. Now we must sum up this expression from 0 to $\log n$ to get total cost of the entire operation.

$$T(n) = \sum_{i=0}^{\log n} i(\frac{n}{2^i}) \tag{1}$$

Now we can pull n out since it is not the summation variable

$$T(n) = n \sum_{i=0}^{\log n} \frac{i}{2^i} \tag{2}$$

Since we are looking for an upper bound, we sum from 0 to ∞

$$T(n) = n \sum_{i=0}^{\infty} \frac{i}{2^i} \tag{3}$$

The sum is equal to 2, however even if you cannot figure this out, it should be clear that it does converge to some number c without actually knowing what c is.

$$T(n) = 2n \tag{4}$$

So because of this we can say that buildHeap() runs in time proportional to $O(n)$.