**Build Heap – Proving O(Build Heap) = n:**

|  |
| --- |
| **Part 1 – Developing a Mathematical Equation for Build Heap:**     * The work done by the heapify function in the add depends on the level of the node that it is being called on. If that node is the root, then the complexity will be log n to fix it. If the node is at level 1 (i.e. one level above the leafs) then it will take only one step to fix it in the heapify. If the node that needs to be re-arranged is halfway up the tree, it will take approximately ½ log n. So heapify has a average and worst case complexity of log n. The complexity completely depends on the level of the node that it calls heapify on.      * We have established that the complexity of heapify depends on the level of the node that is out of order. Let say this level is j (where j = 0 in the best case when this is a leaf node, j = log n = h in the worst case, and j = ½ h in the average case). The number of steps of heapify given level j = j. * The number of nodes at level j is 2h-j. So when j = 0 (i.e. the leaf) there is 23-0 = 8 nodes. When j = 3 (i.e. the root) there is 23-3 = 1 node. * Since we know the complexity of heapify at level j (it is j steps) and we know the number of nodes at a level (it is 2h-j) then the complexity to call heapify on the entire level j is . You can see the right side of the tree diagram above and see proof of this. * The time it takes to call heapify add on ALL the nodes in the tree is the summation of all the levels which is     **Note:** This is the complexity of build heap which is literally adding all nodes to the tree. |

|  |
| --- |
| **Part 2 – Simplifying the Mathematical Equation for Build Heap:**  We know that O(Build Heap) =   * This is mathematically equivalent to * Since h is a constant we can factor out the 2h to get * See the appendix on how we can prove that = 2. This means that <= 2. This means we can substitute for 2 in the O(Build Heap) function since two is an upperbound of * We can now simplify to O(Build Heap) = 2 × 2h = 2h + 1. * Since n = 2h + 1 – 1 for trees, then n + 1 = 2h + 1. * O(Build Heap) = n + 1 = n if we substitute the last part. |

|  |
| --- |
| **Appendix:**  We are going to prove that = 2.   * We start off with the geometric series equation * If you take the derivative of both sides, you’ll get * If you multiply both sides by x, you’ll get * Now substitute x (which is a coefficient in the geometric series and not a variable) to 2. This yields = 2. * Finally so substituting yields, = 2. |