FizzBuzz Big O Problem 7

October 11, 2017

Question

If a Binary search tree is not balanced, how long will it take (worst case) to find an element in it?

Explanation and Algorithm

When thinking about the Big O of your algorithm, write a formula for your run time and then drop the constants. For example, if you have an algorithm with two for loops, the run time is $0(n)$, not $O(2n)$. In addition to this, it is also important to remember to drop the non dominant terms. For example, $O(N^2 + N)$ should simplify to $O(N^2)$. The Big O times in order from most efficient to least efficient are the following: $O(1)$, $O(logn)$, $O(nlogn)$, $O(n^2)$, $O(2^n)$, $O(n!)$.

Hints

1. Think about how many times an operation is conducted in this problem. What are the main operations?
2. An explanation of $O(n)$: This means an algorithm is being run in constant time.
3. An explanation of $O(logn)$: If n number of operations get divided at each run time.
4. An explanation of $O(n^2)$: If n times n operations are performed.
5. An explanation of $O(2^n)$: If the number of operations increases exponentially.

Run time analysis

In our worst case scenario, the height of the tree is the number of nodes. This means that our worst case run time for an unbalanced tree is O(n). N stands for the number of nodes in the tree. In the worst case, our maximum time to find an element is the depth of the tree. Our tree will have a depth of n.