Performance

1. My Workload
    a. I designed a workload that causes half the tree to lock while on the other half a simple query is being made
        i. It does this by starting with a balanced tree and querying on the left side constantly
        ii. Then on the right side it removes a node one down which requires it to traverse all the way down 5000 nodes to find what to swap it with, effectively locking this half of the tree
    b. This lets the query happen fast during fine grain locking when just part of the tree needs to be locked
    c. Then under RW lock it will allow querying to happen simultaneously instead of being blocked
    d. Then under coarse it is fully linear, and essentially single threaded
2. Input files
    a. To run the test I ran
        i. time ./server_fine <input
        ii. time ./server_rw <input
        iii. time./server_coarse<input
    b. input is just a simple test file piping commands into the server
    c. It calls tree to initialize the tree
    d. Then runs remove and q6000
        i. Remove removes the right side of the tree super inefficiently
        ii. q6000 querys the left side 6000 times
3. Results
    a. Coarse: 5.755
    b. RW: 5.691
    c. Fine: 6.404
        i. Here fine was less efficient than RW and coarse due to the additional overhead of having to lock and unlock while traversing the array
        ii. This is quite a significant amount of overhead, and is only worthwhile if each node operation takes much more operating time than locking & unlocking the node does, which is not the case at all
        iii. Because of this I was unable to find an instance of fine being more effective
4. Why locking matters more on multiprocessor machines
    a. When a machine is a single processor, there is no efficiency gains from locking effectively, all that matters is avoiding deadlock. Regardless of how effective the locks are, only one database operation can be run at a time
    b. When on a multiprocessor machine each processor can access the data simultaneously if locked properly, thus it can be more efficient. If the locking strategy is too simple, such as coarse grained locking here, then only one processor access the data at a time, just like in a single processor system.