test1: insert unbalanced

**BINARY TREE**

| nodes | seconds |
|---|---|
| 16,000 | 1.04 |
| 32,000 | 3.9 |
| 64,000 | 15.97 |
| 128,000 | 73.38 |

test2: insert balanced

| nodes | micro seconds |
|---|---|
| 16,000 | 8091 |
| 32,000 | 17016 |
| 64,000 | 31059 |
| 128,000 | 55050 |

test3: remove unbalanced

| nodes | seconds |
|---|---|
| 16,000 | 0.78 |
| 32,000 | 2.68 |
| 64,000 | 7.96 |
| 128,000 | 15.51 |

test4: remove balanced

| nodes | micro seconds |
|---|---|
| 16,000 | 4114 |
| 32,000 | 8886 |
| 64,000 | 18256 |
| 128,000 | 34314 |

test5: find unbalanced

| nodes | seconds |
|---|---|
| 16,000 | 1.63 |
| 32,000 | 6 |
| 64,000 | 19.6 |
| 128,000 | 46.07 |

test6: find balanced

| nodes | micro seconds |
|---|---|
| 16,000 | 5273 |
| 32,000 | 11207 |
| 64,000 | 22670 |
| 128,000 | 42665 |

test1: insert unbalanced

**3-ARY TREE**

| nodes | seconds |
|---|---|
| 16,000 | 0.848 |

| | |
|---|---|
| 32,000 | 3.21 |
| 64,000 | 12.72 |
| 128,000 | 51.21 |

test2: insert balanced

| nodes | micro seconds |
|---|---|
| 16,000 | 6960 |
| 32,000 | 14617 |
| 64,000 | 28131 |
| 128,000 | 46335 |

test3: remove unbalanced

| nodes | seconds |
|---|---|
| 16,000 | |
| 32,000 | |
| 64,000 | |
| 128,000 | |

test4: remove balanced

| nodes | micro seconds |
|---|---|
| 16,000 | 2661 |
| 32,000 | 6078 |
| 64,000 | 13671 |
| 128,000 | 34976 |

test5: find unbalanced

| nodes | seconds |
|---|---|
| 16,000 | 1.21 |
| 32,000 | 4.46 |
| 64,000 | 14.39 |
| 128,000 | 33.08 |

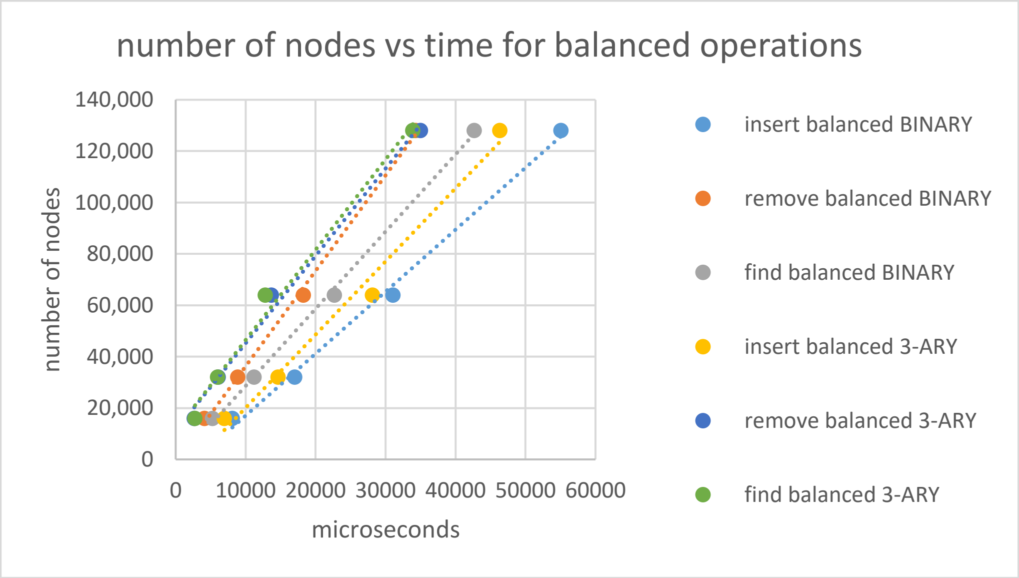
test6: find balanced

| nodes | micro seconds |
|---|---|
| 16,000 | 2756 |
| 32,000 | 5979 |
| 64,000 | 12781 |
| 128,000 | 33897 |

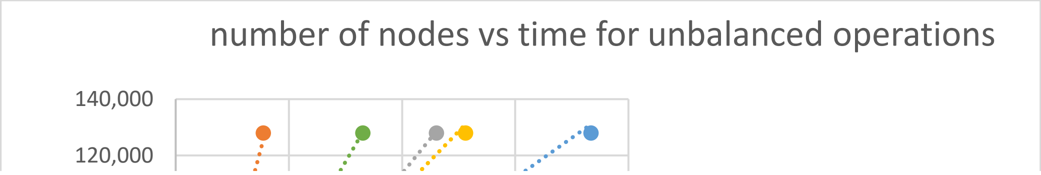
When we started testing, we wanted to use the same tests that we ran on the binary search tree in order
to get the most accurate comparison. We tested the 3-ary search tree with 16,000 nodes, 32,000 nodes, 64,0
and 128,000 nodes. For each node amount, we made both an unbalanced and a balanced tree. On each tree v
the time it took to do remove, find, and insert a certain amount of times.
We expected the 3-ary tree to be faster for multiple reasons. The first is that these nodes have two values rat
so there is less traversal necessary. Second, each time we move down the tree, much more elements are elim
we move down the bianry tree. It is about 2/3 eliminated rather than the binary's 1/2 eliminated.

Our results show that our predictions were correct. The 3-ary tree was faster in each operation, but not by a h
However, the difference is still noticeable and more drastic as time goes on. This is because log3 grows slowe
3-ary tree uses log3.

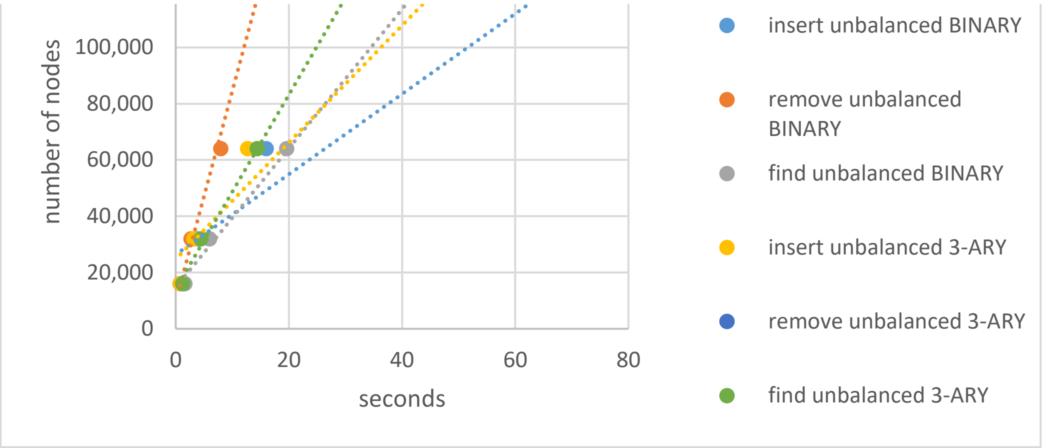
NOTE: We did not include any data for removing nodes from the 3-ary tree because the test cases took so lon
works, but because our implementation relies so heavily on recursion, the runtime for remove is huge for larg
If you run our implementation with less nodes, the test cases will not take so long.



number of nodes vs time for balanced operations



number of nodes vs time for unbalanced operations



00 nodes,
ve measured

her than one,
inated than when

ıuge amount.
r than log2, and the

ıg. Our implementation
ȝe amounts of nodes.