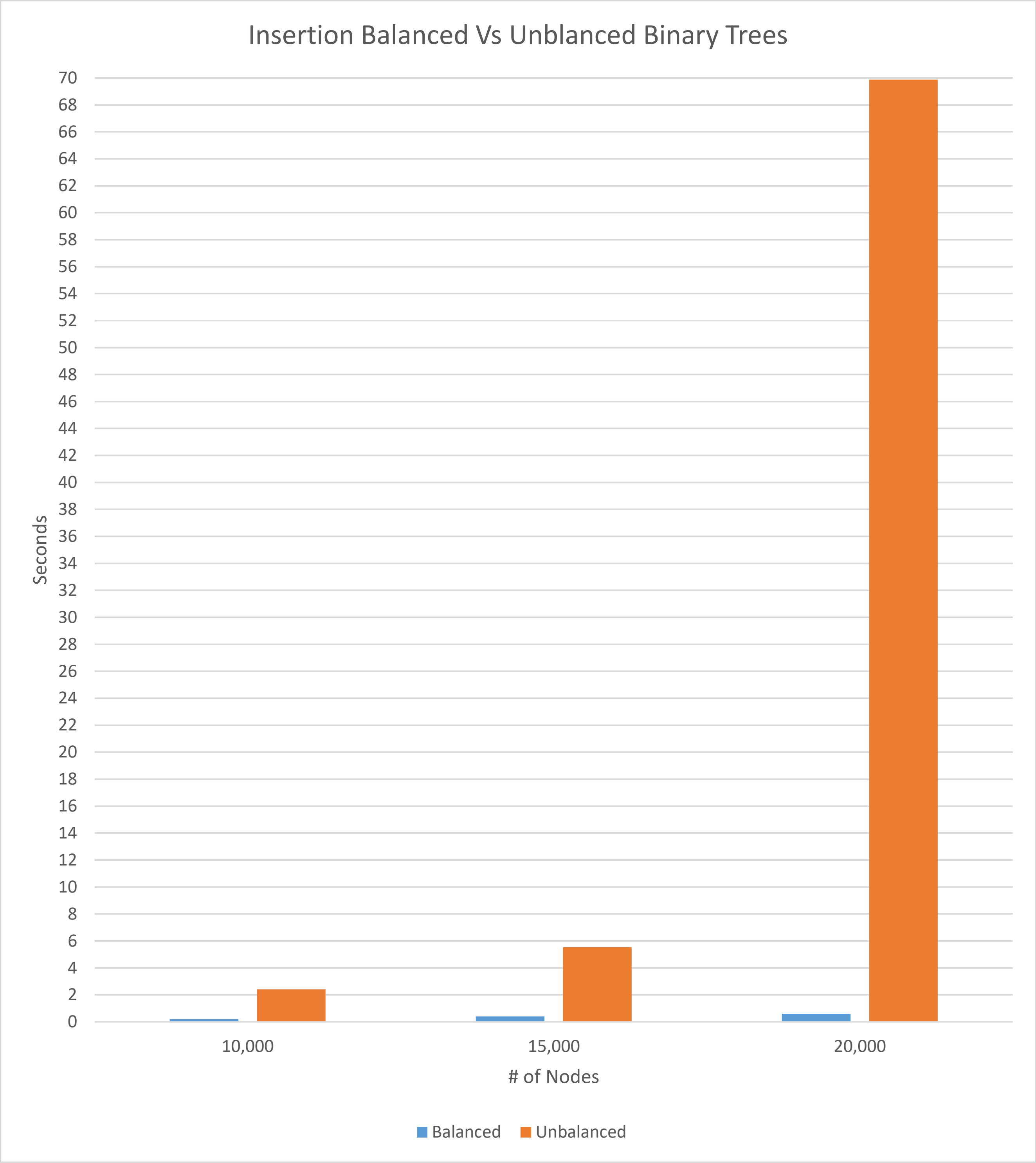
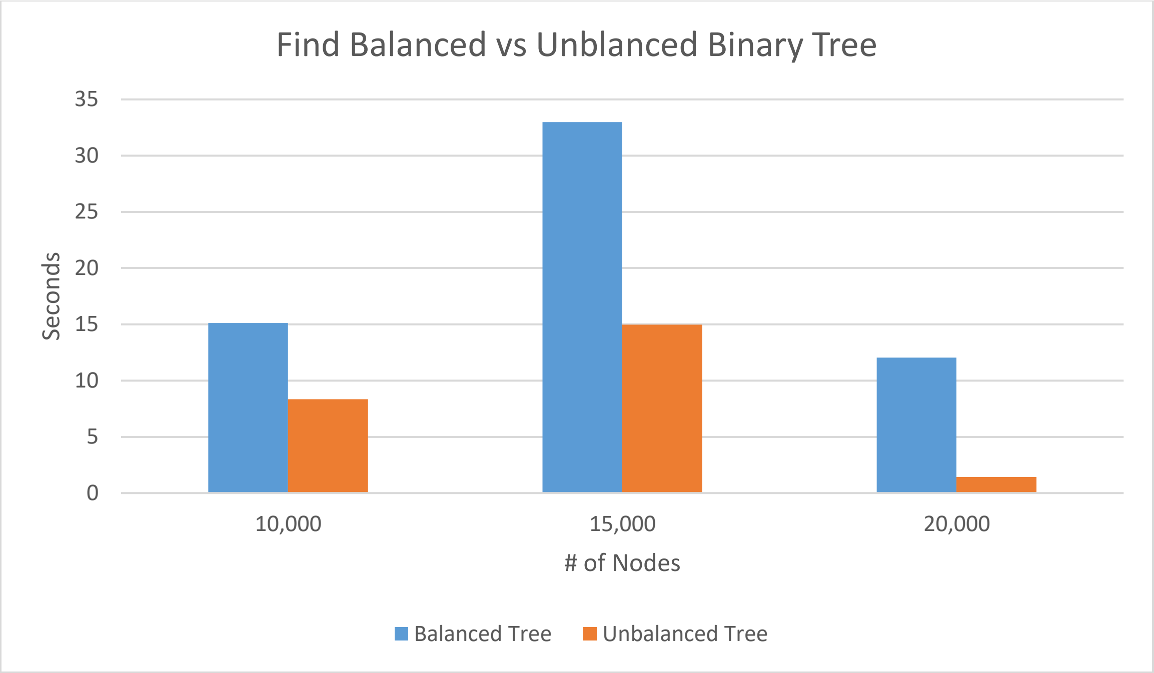
Treasa Roy

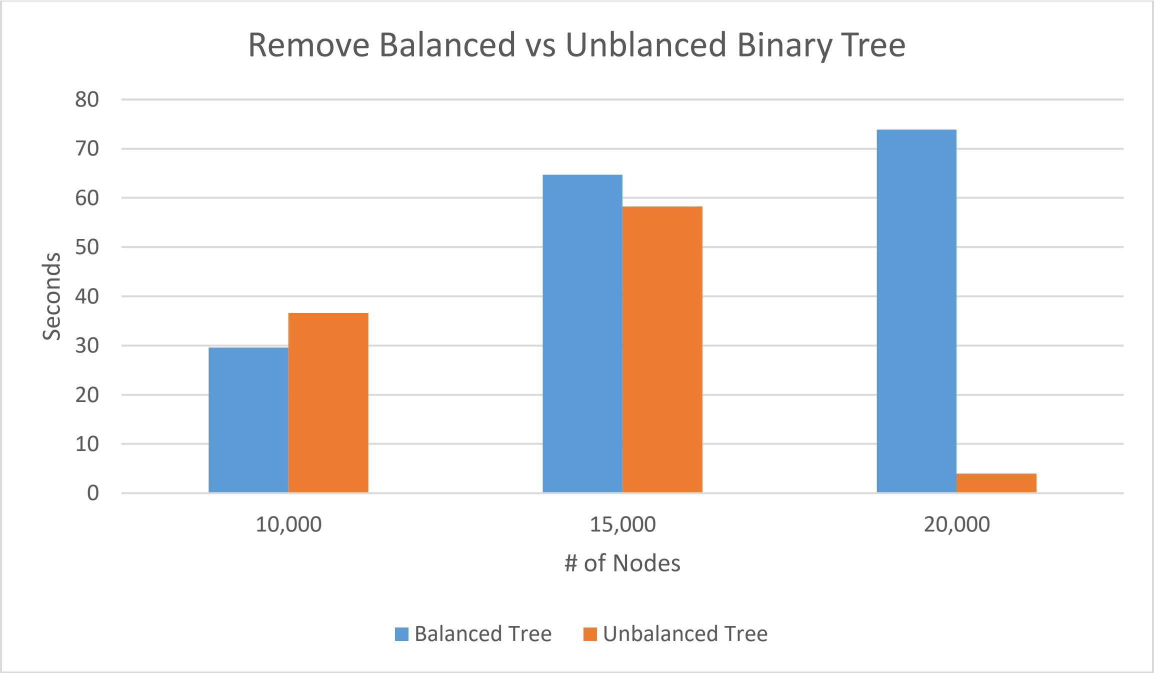
Kristjana Popovski

Lab 6 Report

The graphs we created for balanced binary tree are displayed below:




Find Balanced vs Unblanced Binary Tree
35
30
25
20
15
10
5
0
Seconds
10,000
15,000
20,000
# of Nodes
■ Balanced Tree
■ Unbalanced Tree


Remove Balanced vs Unblanced Binary Tree
80
70
60
50
40
30
20
10
0
Seconds
10,000
15,000
20,000
# of Nodes
■ Balanced Tree
■ Unbalanced Tree

Program 2 Report

a) **Problem We're Trying to Solve:**
We're trying to figure out if it takes more time to implement the insert, remove, and find function for the balanced 3-ary tree or unbalanced 3-ary tree. In the end, we will compare the binary tree vs the 3-ary tree.

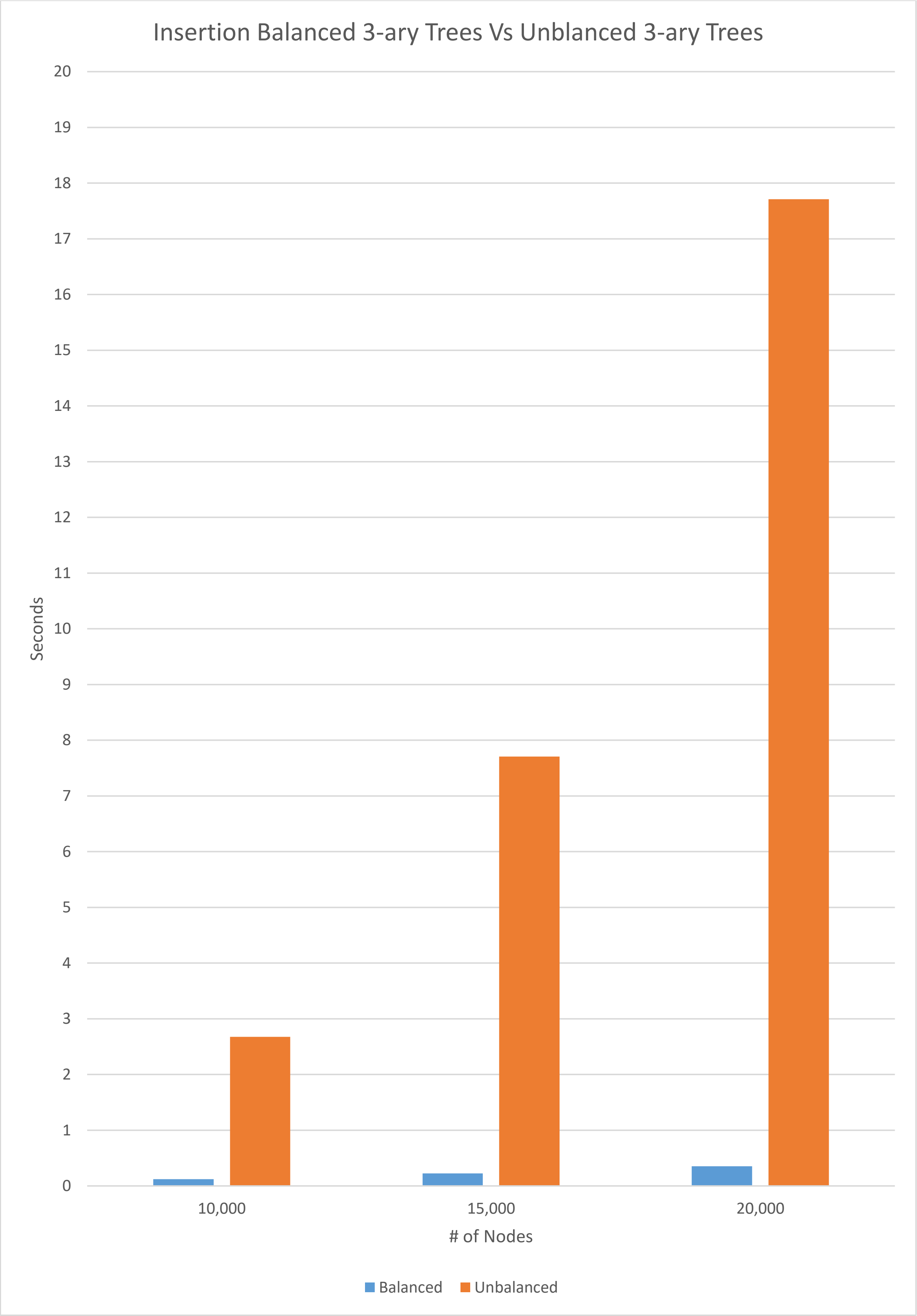
b) **Tests we ran:**
We're comparing 3-ary trees that have 10000, 15000, and 20000 nodes. We're finding how long it takes to implement the insert, remove, and find functions for balanced and unbalanced 3-ary trees for these 3 values. We created the balanced tree by inserting random numbers and the unbalanced tree by inserting numbers in order starting from 0 to either 10000, 20000, or 30000 nodes.

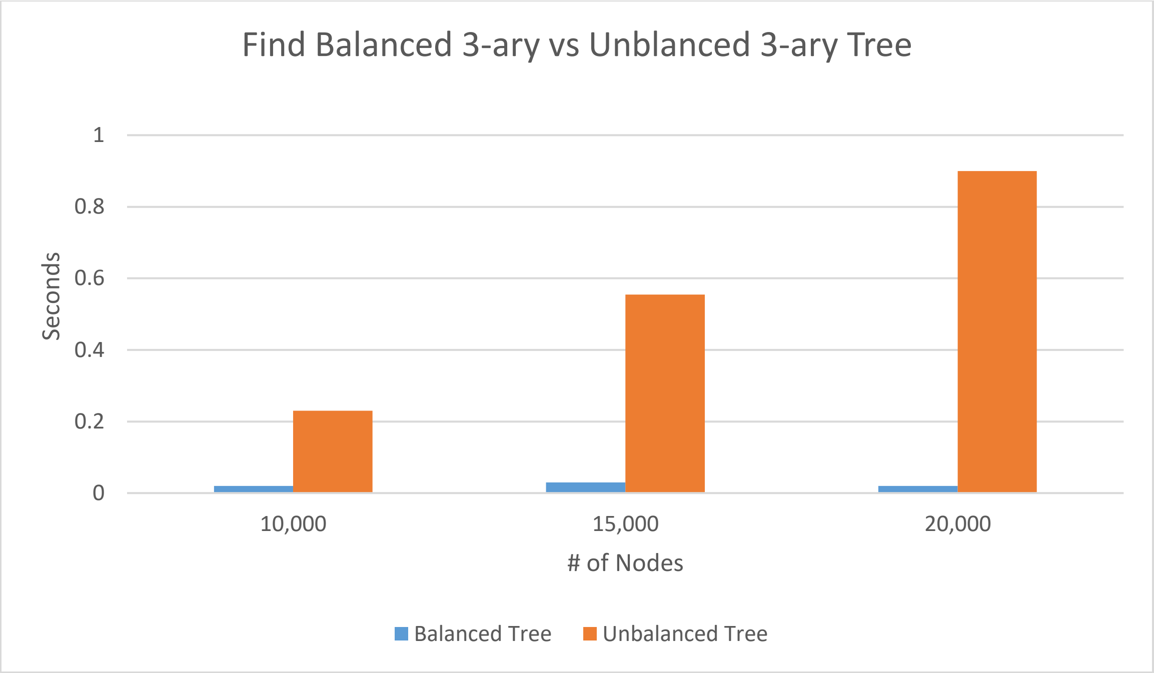
We found 5,000 for 10,000 nodes.
We found 7,500 for 15,000 nodes.
We found 10,000 for 20,000 nodes.

We removed 4,425 for 10,000 nodes.
We removed 5,000 for 15,000 nodes.
We removed 7,000 for 20,000 nodes.

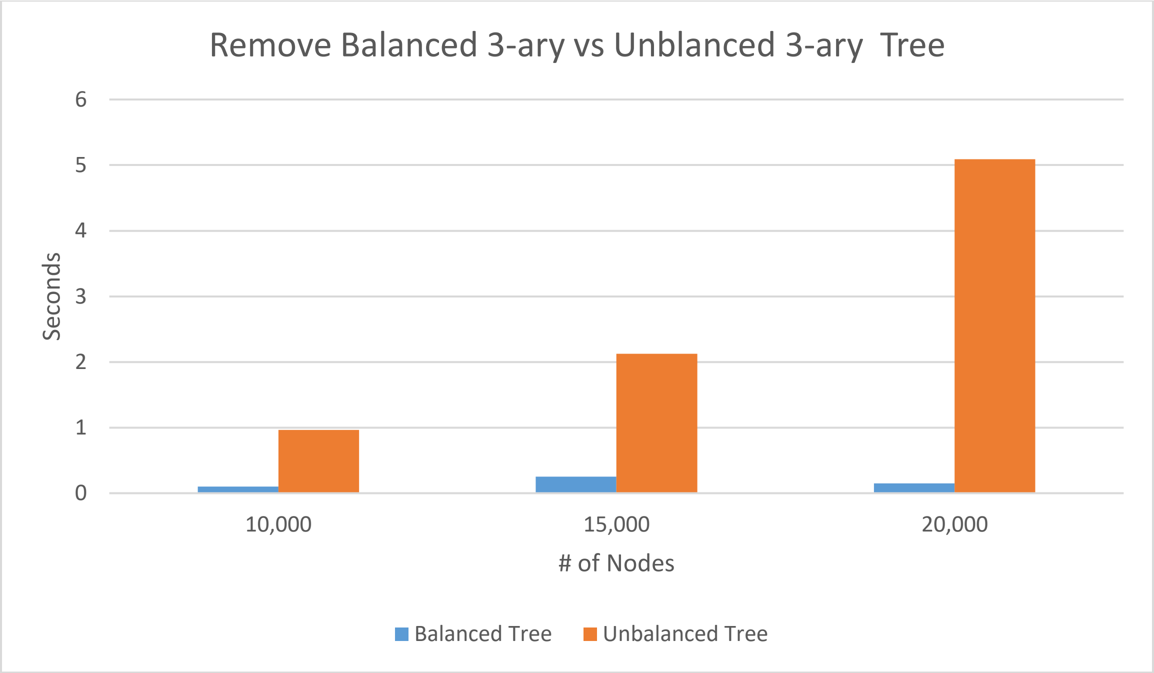
c) **Draw conclusions:**
We concluded implementing the insert, remove, and find for the balanced 3-ary tree takes much less time than an unbalanced 3-ary tree. For remove and find we noticed that the 20,000 nodes took less time to execute than the 10,000 or 15,000 nodes. This could be because of the operating system.

The graphs we created are displayed below:


Insertion Balanced 3-ary Trees Vs Unblanced 3-ary Trees
20
19
18
17
16
15
14
13
12
11
10
9
8
7
6
5
4
3
2
1
0
Seconds
10,000
15,000
20,000
# of Nodes
Balanced
Unbalanced



Find Balanced 3-ary vs Unblanced 3-ary Tree



Remove Balanced 3-ary vs Unblanced 3-ary Tree

Binary vs 3-ary tree:

In general, the run time performance for inserting into a balanced tree is much better than inserting into an unbalanced tree. This is because a balanced tree has a shorter height and the height of the trees dictates the number of operations that the tree must execute.

Theoretically the runtime complexity for a balanced tree is nlogn and an unbalanced tree is n. A RTC for nlogn is going to be much faster than n which our data supports.

We noticed that the insertion, find, and remove for the 3-ary trees took significantly less time to execute than the insertion, find, and remove for the binary tree. When comparing the binary tree to the 3-ary tree, the height for the 3-ary tree is less but there will be more work required for each node. There could be a significant difference in the time because of the height difference.

In theory, the RTC for a binary tree is log(2n) and is log(3n) for 3-nary tree. Therefore, the data should have been somewhat similar but our data does not reflect this theory. The reason it was not similar could have been because of the way we implemented the functions. We probably created it in a more efficient way.