Introduction:

The objective of this assignment is to implement a Binary Search Tree and a 2-4 tree is order to compare the running times of the to data structures. Both structures are implemented using a linked structure with BST nodes holding one key-value pair and 2-4 nodes holding 1, 2, or 3 key-value pairs.

Theoretical Analysis:

Both implementations utilize the search(k), insert(k,v),delete(k), size(), and empty() functions.

First I will analyze the Binary Search Tree's functions. The size() and empty() functions run in O(1) time due to just checking the size variable and either returning it or returning true if size is 0. The search(k) function will have an O(n) worst case when all of the items form a long linear chain, but will have an O(log n) runtime if the tree is relatively balanced. The search(k) function in this case will start at the root and will perform a binary search for the key, going to the left child if the key being searched for is less than the current nodes key and will go to the right child if the key is greater. If the key is equal the node holding it will be returned and if it is not found, the dummy node where it would fit is returned. The insert(k,v) function takes the same amount of time to run as the find function does. This is due to put using find in order to either update an existing node holding key k or to just insert a new node to the dummy node location found by find if no such key-value pair exists. The delete(k) function takes the same amount of time as find due to using find to locate the node that needs to be removed. It will take very slightly longer than put due to having to find the successor node if the node being removed is an internal node, but it doesn't take more time asymptotically. In this program the delete function would not work for large random inputs due to an issue finding the successor. I was unable to resolve this issue due to time limitations, but the issue has to do with the loop for finding the predecessor going too far.

The 2-4 tree functions are similar in function but vary in implementation and complexity. The size() and empty() functions serve the exact same purpose and run in the same time as in the Binary Search Tree. The search(k) function for a 2-4 tree is similar but will check each value stored in the node instead of just the one. This function runs in O(logn) time however because the height of the tree is always guaranteed to be logn. The delete(k) function also gets the benefit of running in O(logn) time as well due to the height properties of a 2-4 tree. This function finds the node looking to be deleted and removes it from the tree. Transformations are made to the tree afterwards to make sure that the height properties and number of children properties are held. The insert(k,v) function also runs in O(logn) time due to the height property. This function searches for the right location to insert the item and then makes the proper transformations to preserve the height of the tree. Due to time limitations, the delete function is not working for all cases, but does work in the random case. The issue I found but was unable to resolve was related to the node being found to delete being one too far down.

Experimental Setup:

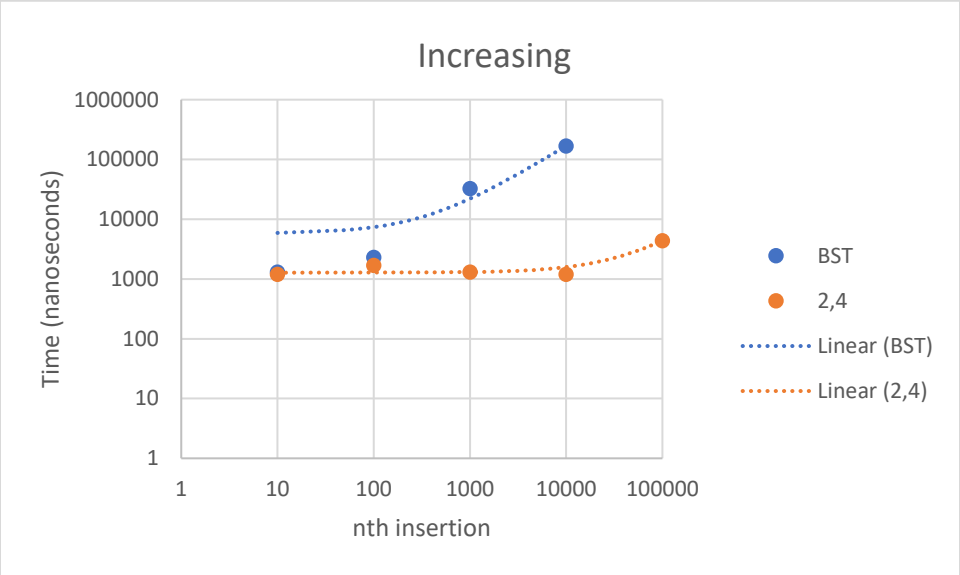
To perform the experiments if this assignment I used a Dell Inspiron 5577 with an Intel Core i7-7700HQ CPU @2.80 GHz. This laptop has 8 GB of RAM and a 64-bit operating system. For the compiler I

used Cygwin v3.0.7 and CMake v3.14.5 in combination with the CLion 2019.2.1 IDE. For this program I used C++14.

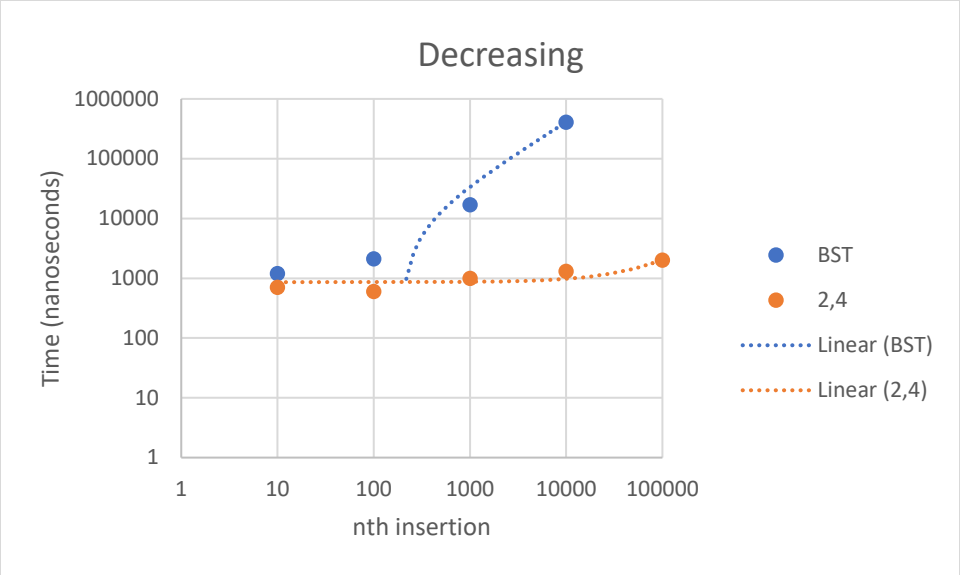
To measure the amount of time to sort n elements I used the high_resolution_clock from the C++ chrono library. I set it to nanoseconds in order to get accurate measurements on the lower test cases. I chose nanoseconds because milliseconds and coarser measurements did not give accurate results below 1000 elements.

The experiments were recorded only once for each case of increasing, decreasing, and random for both data structures. My computer would run into errors while trying to perform the insertion into the BST due to a lack of memory so I stopped at 10,000.

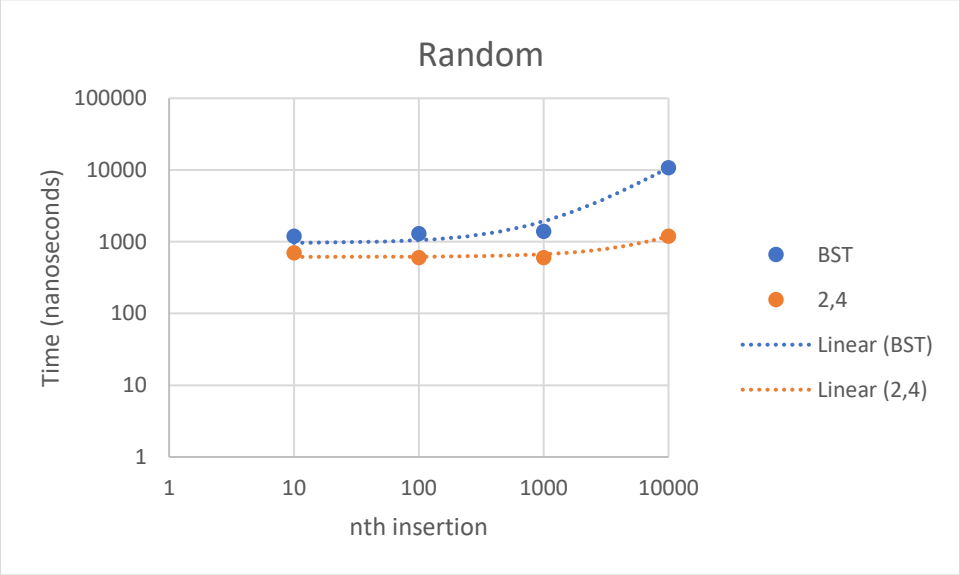
Experimental Results:



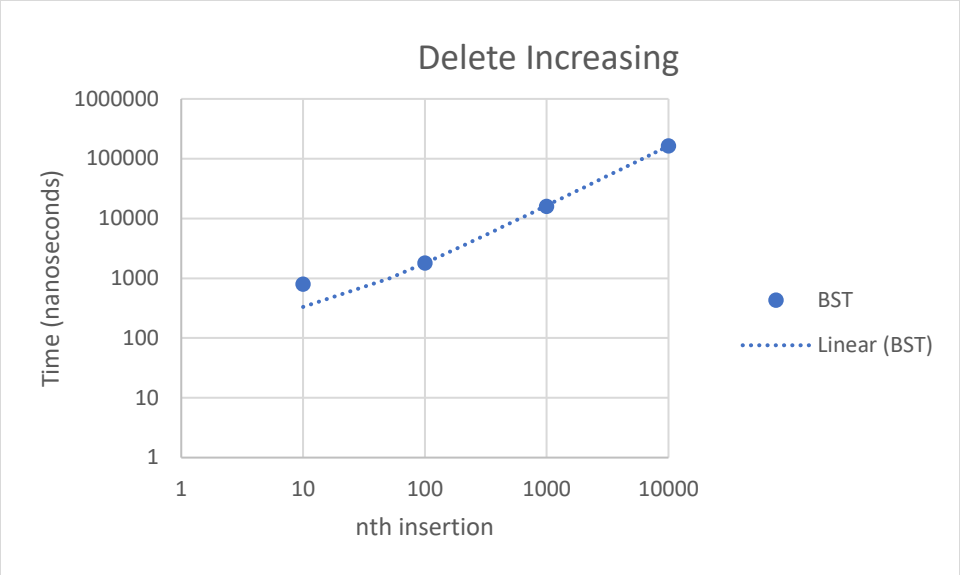
For the binary search tree, the insertion algorithm shows a runtime of about O(n) due to being essentially a long linear chain of nodes. The 2-4 tree on the other hand is running in O(logn) time due to the height being logn.



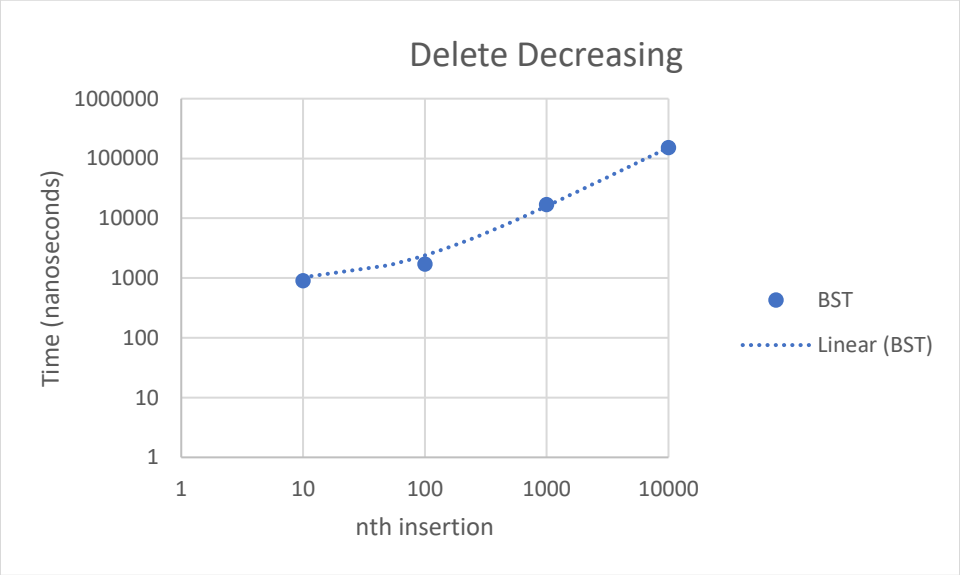
For the binary search tree, the insertion algorithm shows a runtime of about O(n) due to being essentially a long linear chain of nodes. The 2-4 tree on the other hand is running in O(logn) time due to the height being logn.



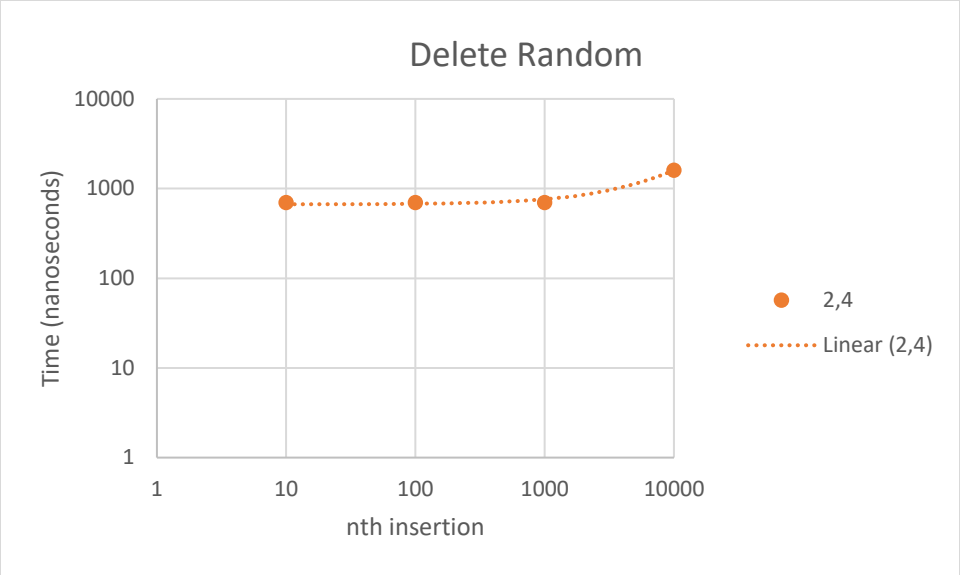
For both algorithms in this case the running time is about O(logn). Since the tree is more balanced with this input, the binary search tree can have a height of about logn. This lets it run in a similar, although still slightly longer, time to a 2-4 tree.



Since I was unable to complete the deletion algorithm for all cases, the 2-4 tree results are not shown. The binary search tree is shown to be running in O(n) time due to being a long chain of nodes.



Since I was unable to complete the deletion algorithm for all cases, the 2-4 tree results are not shown. The binary search tree is shown to be running in O(n) time due to being a long chain of nodes.



Since the deletion algorithm for the binary search tree was unable to be fixed, only the 2-4 trees runtime is shown. The 2-4 tree is shown to run in O(logn) time due to having the height always being logn.