CSCE 221-200: Honors Data Structures and Algorithms
Assignment Cover Page
Spring 2021

Name:	Alexander Akomer
Email:	alexakomer@tamu.edu
Assignment:	PA 2
Grade (filled in by grader):	

Please list below all sources (people, books, webpages, etc) consulted regarding this assignment (use the back if necessary):

CSCE 221 Students	Other People	Printed Material	Web Material (give URL)	Other Sources
1.	1.	1.	1. https://www.geeksforgeeks.org/red-black-tree-set-1-introduction-2/	1.
2.	2.	2.	2.	2.
3.	3.	3.	3.	3.

Recall that TAMU Student Rules define academic misconduct to include acquiring answers from any unauthorized source, working with another person when not specifically permitted, observing the work of other students during any exam, providing answers when not specifically authorized to do so, informing any person of the contents of an exam prior to the exam, and failing to credit sources used. *Disciplinary actions range from grade penalty to expulsion.*

"On my honor, as an Aggie, I have neither given nor received unauthorized aid on this academic work. In particular, I certify that I have listed above all the sources that I consulted regarding this assignment, and that I have not received or given any assistance that is contrary to the letter or the spirit of the collaboration guidelines for this assignment."

Signature:	Alexander Akomer
Date:	03/15/2021

Alexander Akomer

PA 2 Report

1. *Introduction:*

This assignment is an early exercise in calculating efficiency of code. We were tasked with creating three different programs that different tree implementations – one following the design of a binary search tree, one following the design of an AVL tree, and the last that of a red-black tree. To compare the time efficiency of the three, 100,000 elements were inserted into each tree in different orders and removed from each tree in corresponding fashions (except for the red-black tree, which we weren't tasked with coding a remove function for). This was done to measure the time complexity of each structure's functions so that the advantages and disadvantages of each could be easily displayed. The results are as follows.

2. *Theoretical analysis:*

Seven different functions were required to be implemented across each of the three tree types. Of these seven functions, two will have no change in their time complexity regardless of how the tree is implemented. These two functions are size(), which will always be O(1), and display, which will always be O(n) as it must iterate through n nodes. The remaining five: findMin(), findMax(), insert(x), delete(x), and contains(x) are different across each implementation.

The binary search tree class theoretically has a worst-case in which a single chain of nodes exists (all entirely left children or entirely right children). In this case, the asymptotic analysis can be identified as Θ(n) for findMin(), findMax(), insert(x), delete(x), and contains(x).

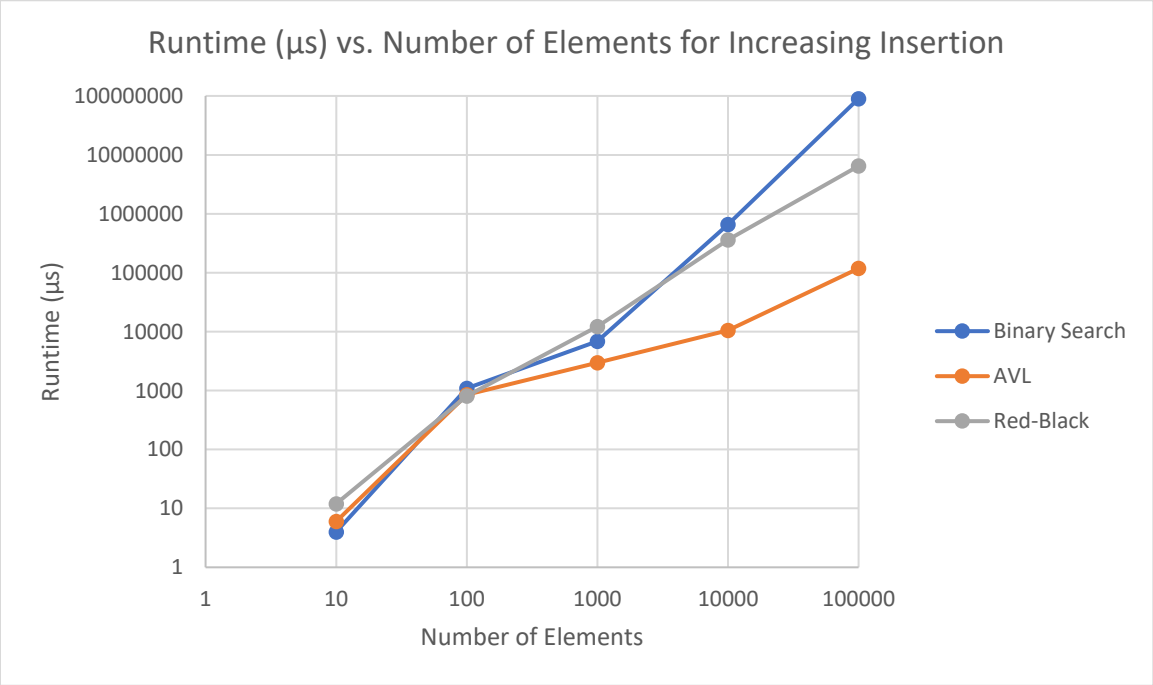
The AVL and red-black tree are similar in their balancing nature. Each will maintain a structure similar to that of an ideal tree – a structure in which subtrees are nearly perfectly balanced, allowing

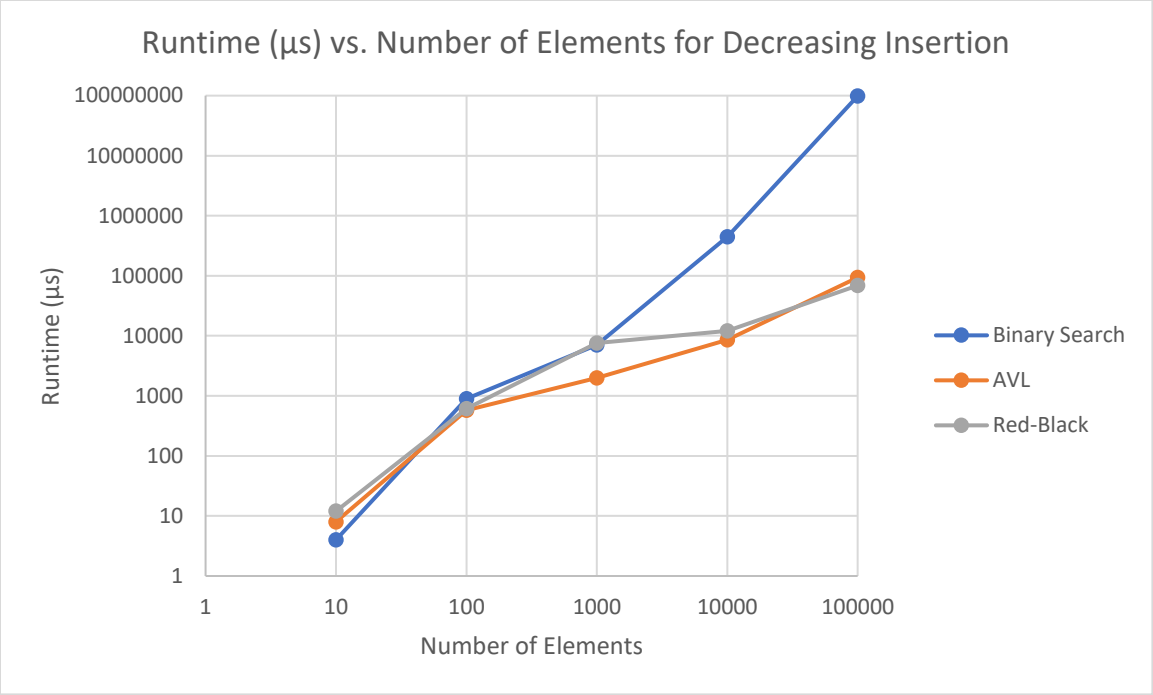
for Θ(log n) time for all five of the listed functions in each the AVL and red-black implementations of a tree.

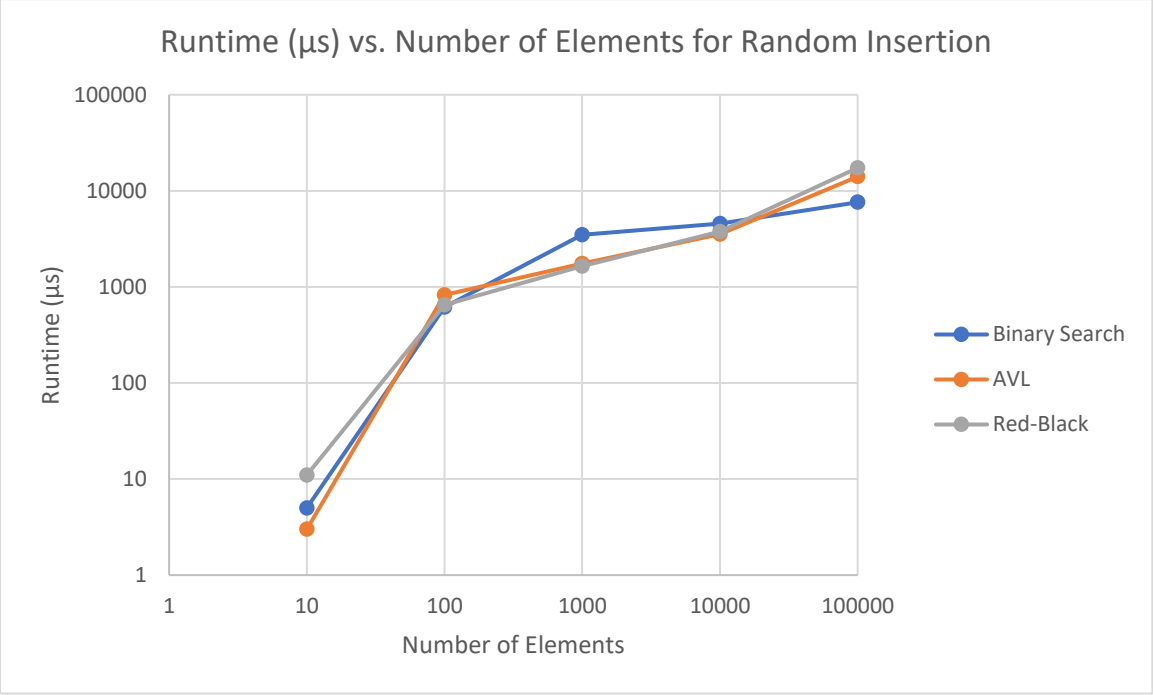
3. *Experimental Setup:*

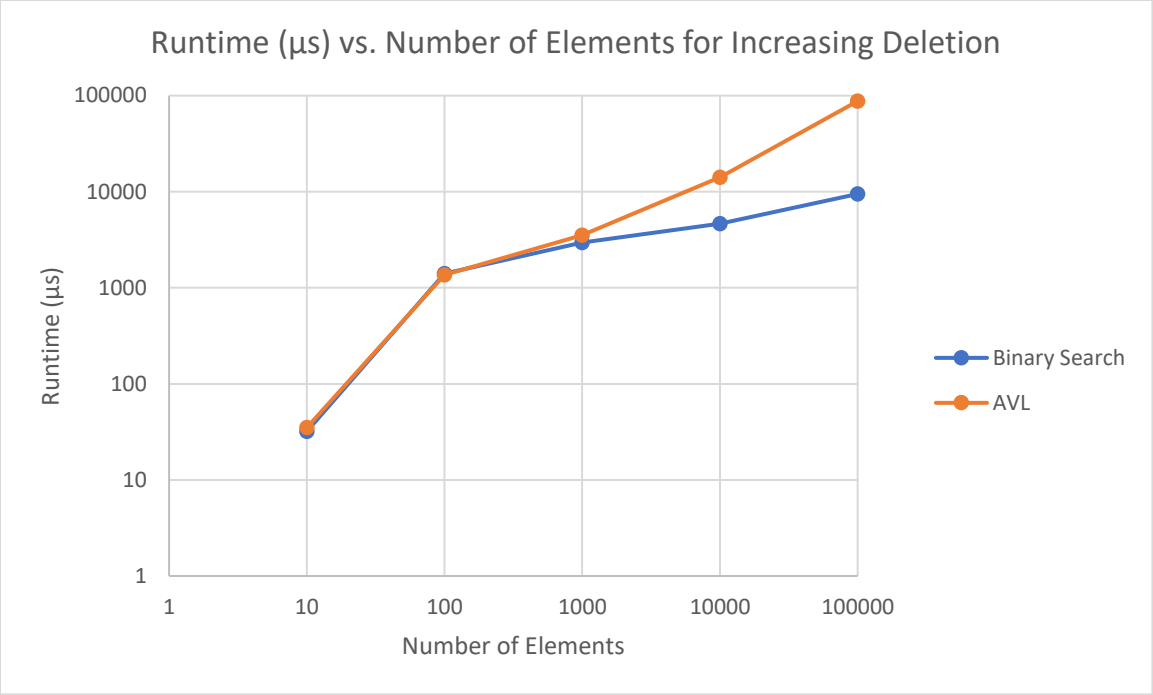
The machine the program was run on for my trials was a Surface Book 2 with 16 GB of RAM, an i7-8650U @ 1.90 GHz, and 109 GB / 475 GB free on the C: drive. My timing mechanism was the code attached on Canvas, which included the chrono package. This package allowed me to run a high-resolution clock and output the runtime in microseconds for each trial. Because the instructions only called for one trial of each function call for each data structure, that is the all of the data that was recorded, which may lead to faults in accuracy.

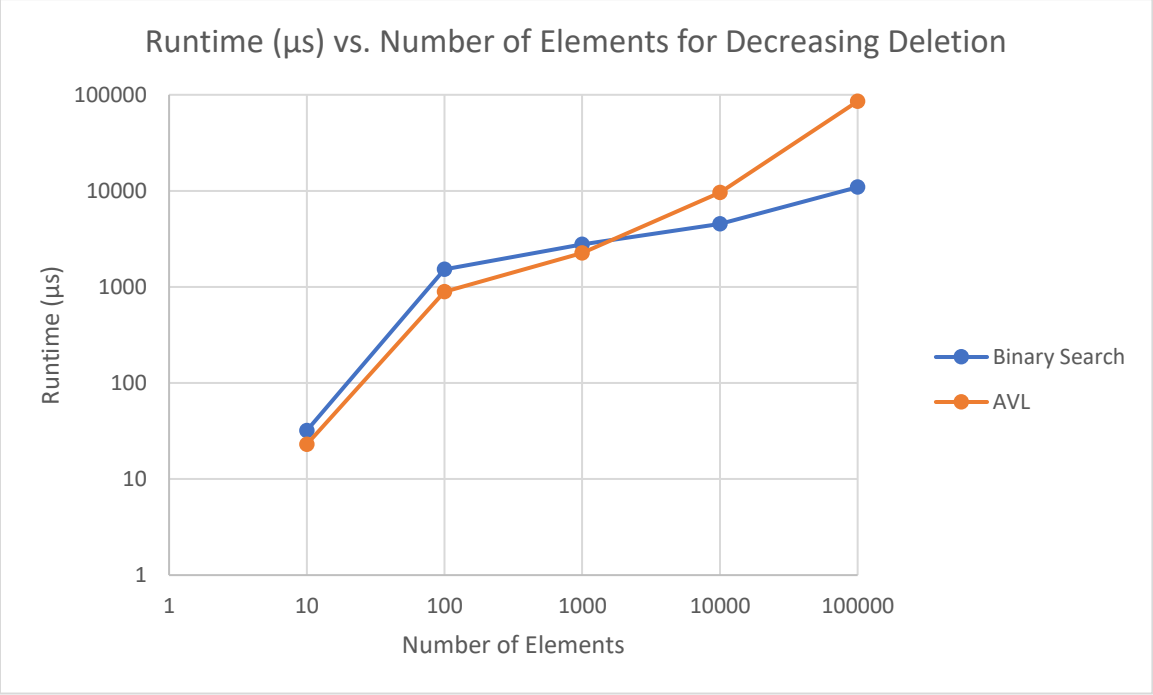
4. *Experimental Results:*

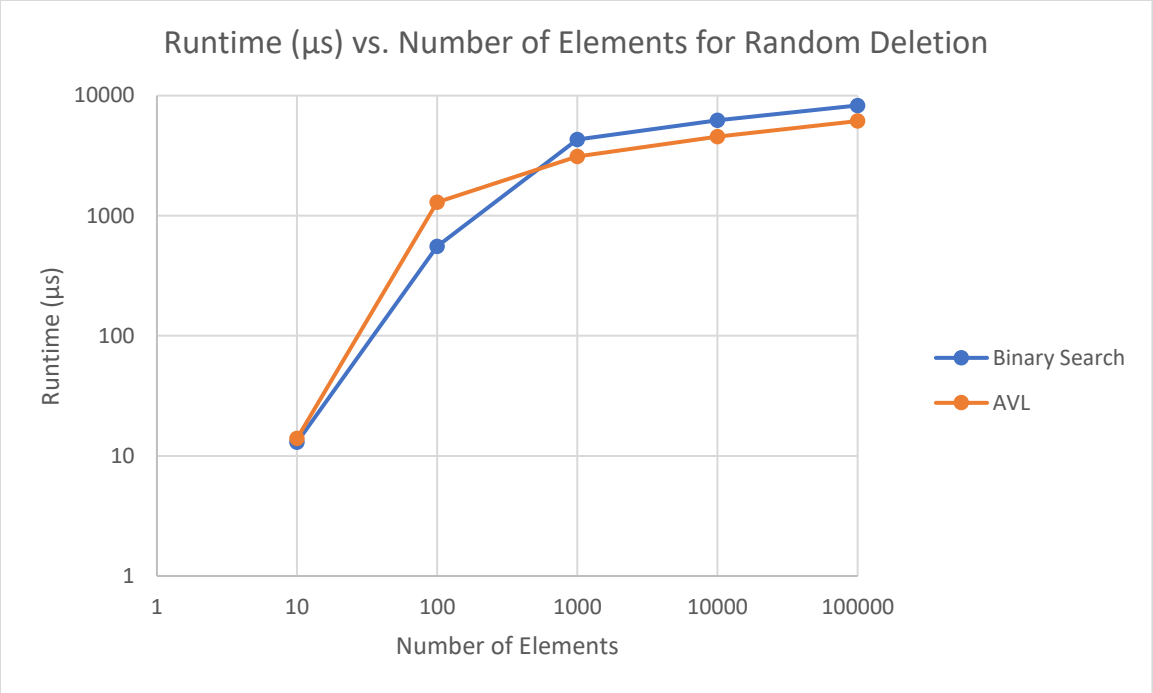



Runtime (µs) vs. Number of Elements for Decreasing Insertion
Runtime (µs)
100000000
10000000
1000000
100000
10000
1000
100
10
1
1
10
100
1000
10000
100000
Number of Elements
Binary Search
AVL
Red-Black


Runtime (µs) vs. Number of Elements for Random Insertion
Runtime (µs)
100000
10000
1000
100
10
1
1
10
100
1000
10000
100000
Number of Elements
Binary Search
AVL
Red-Black


Runtime (µs) vs. Number of Elements for Increasing Deletion
100000
10000
1000
100
10
1
Runtime (µs)
1
10
100
1000
10000
100000
Number of Elements
Binary Search
AVL


Runtime (µs) vs. Number of Elements for Decreasing Deletion
100000
10000
1000
100
10
1
Runtime (µs)
1
10
100
1000
10000
100000
Number of Elements
Binary Search
AVL



After testing the insert and delete functions for each of the following tree implementations, the following was concluded:

Binary Search Tree Insertion: Worst Case O(n), Average Case O(log n)

Binary Search Tree Deletion: Worst Case O(n), Average Case O(log n)

AVL Tree Insertion: Worst/Average Case O(log n)

AVL Tree Deletion: Worst/Average Case O(log n)

Red-Black Tree Insertion: Worst Case O(log n)

The runtime data closely supports the theoretical analysis of each function. The red-black tree's increasing order insertion was surprisingly slow given its theoretical advantage when it comes to insertion and deletion functions. This may be due to a slower implementation of the red-black tree using an extra pointer for the parent nodes or a discrepancy in runtime data because of CPU usage. Despite discrepancies in performance where I would have otherwise expected the red-black tree to outperform the other two

tree implentations, each function was bounded exactly as depiceted in the theoretical analysis – though the multiplicatve constants of the black-red tree may be higher. The only thing to account for this is the additional pointer in my implementation. These results agree almost exactly with the theoretical.