Matthew Moore

Mr. Bowers

12/2/20

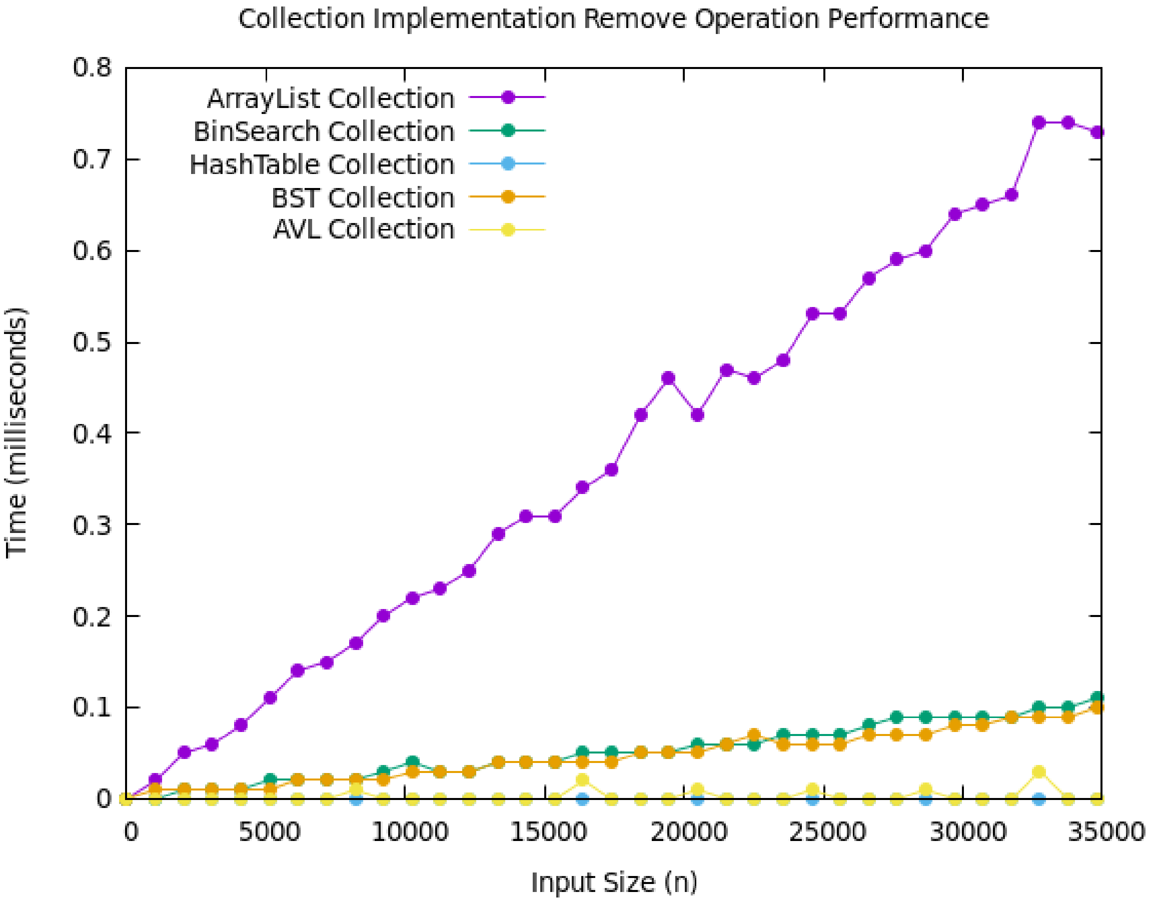
HW 8 Graph Analysis

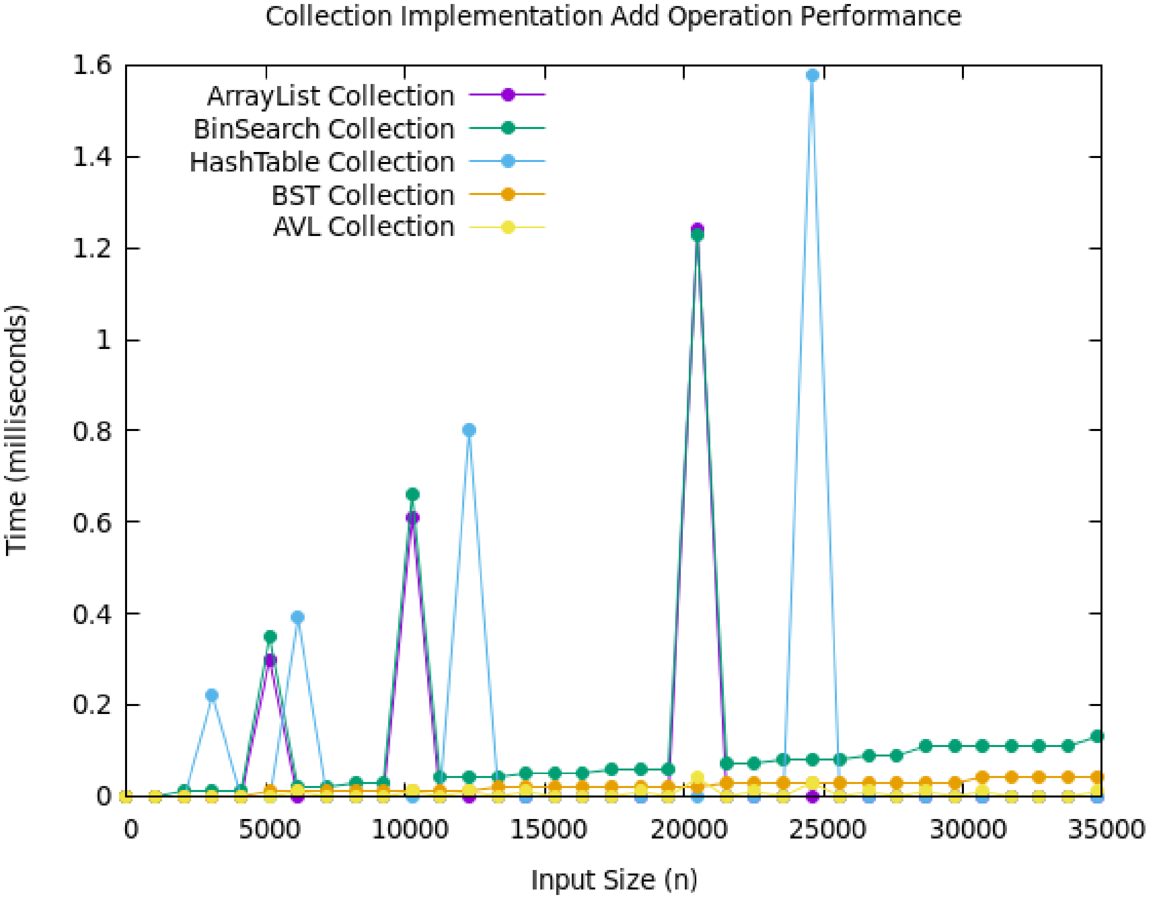
Conclusion:

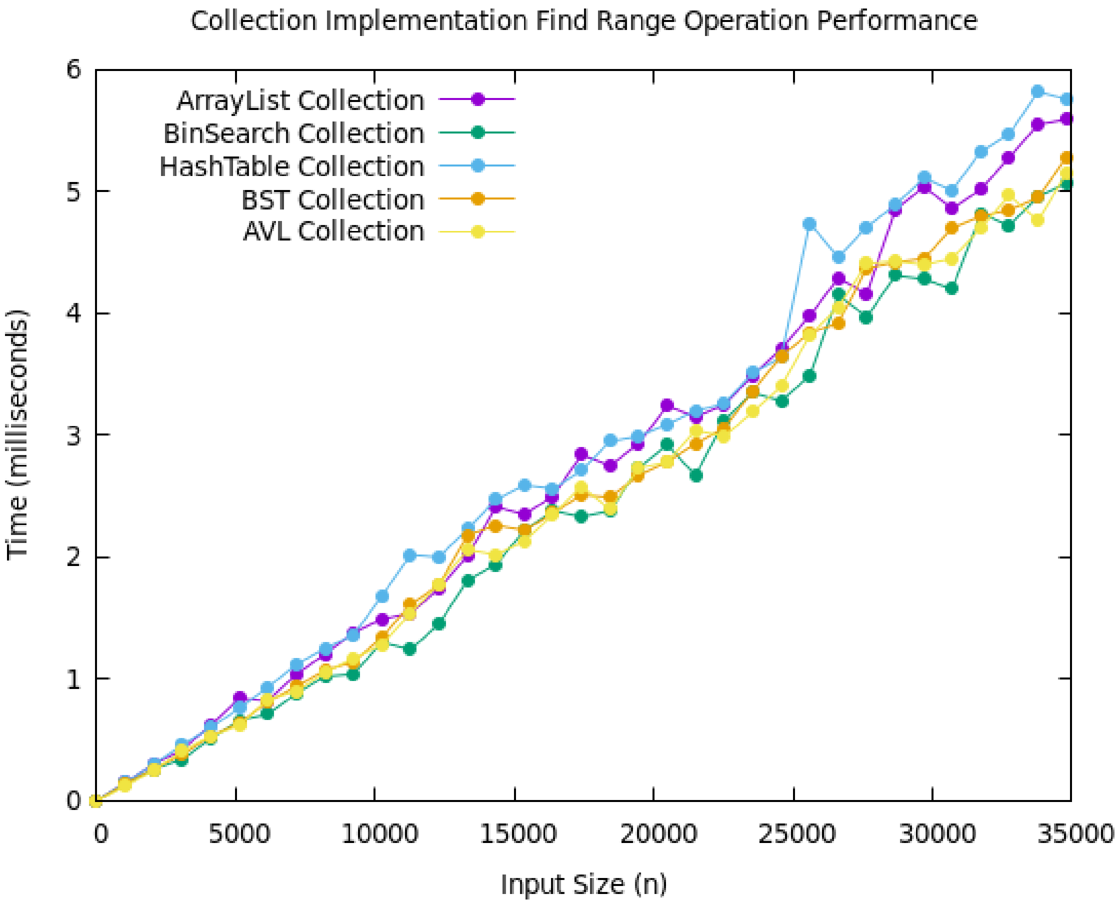
These performance graphs display the AVL Collection's obvious dominance over the regular BST Collection, and it stands as a close competitor with the Hash Table Collection. The AVL Collection operates the add, remove, and find value function at a complexity of O(log n), which is far better than the BST Collection at O(n). We know this because the collection maintains strict height constraints, so each path never passes the height of log n. Although this takes some power using extra recursive steps, it is far more effective than a simple binary search tree. The sort and find range operations also perform the best they could possibly do with a complexity of O(n). The AVL Collection maintains the list in a recursive sorted order so it can easily be retrieved through a linear search through the entire list. There is no quicker way to perform this maneuver so this is a very successful data structure overall.

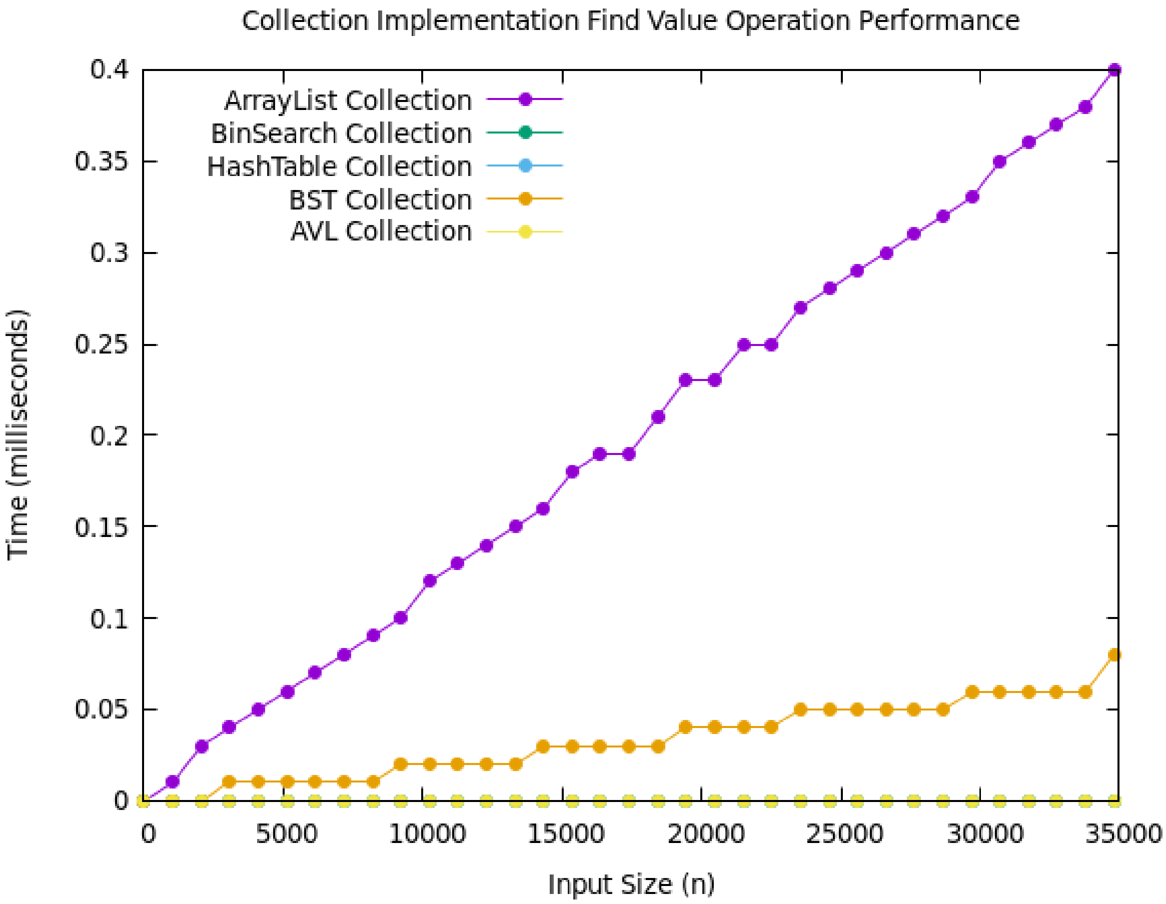
Problems:

I did not find a lot of trouble with this assignment, but I struggled to figure out why my AVL collection is so slow with the add and remove operation. Everything was working for me well, but I was calling the height function one too many times and it dragged the program down significantly. I quickly realized that the height could be adjusted way more efficiently through conditional statements and a simple addition or subtraction based on the case. This sped up the performance by more than a factor of ten which really surprised me.

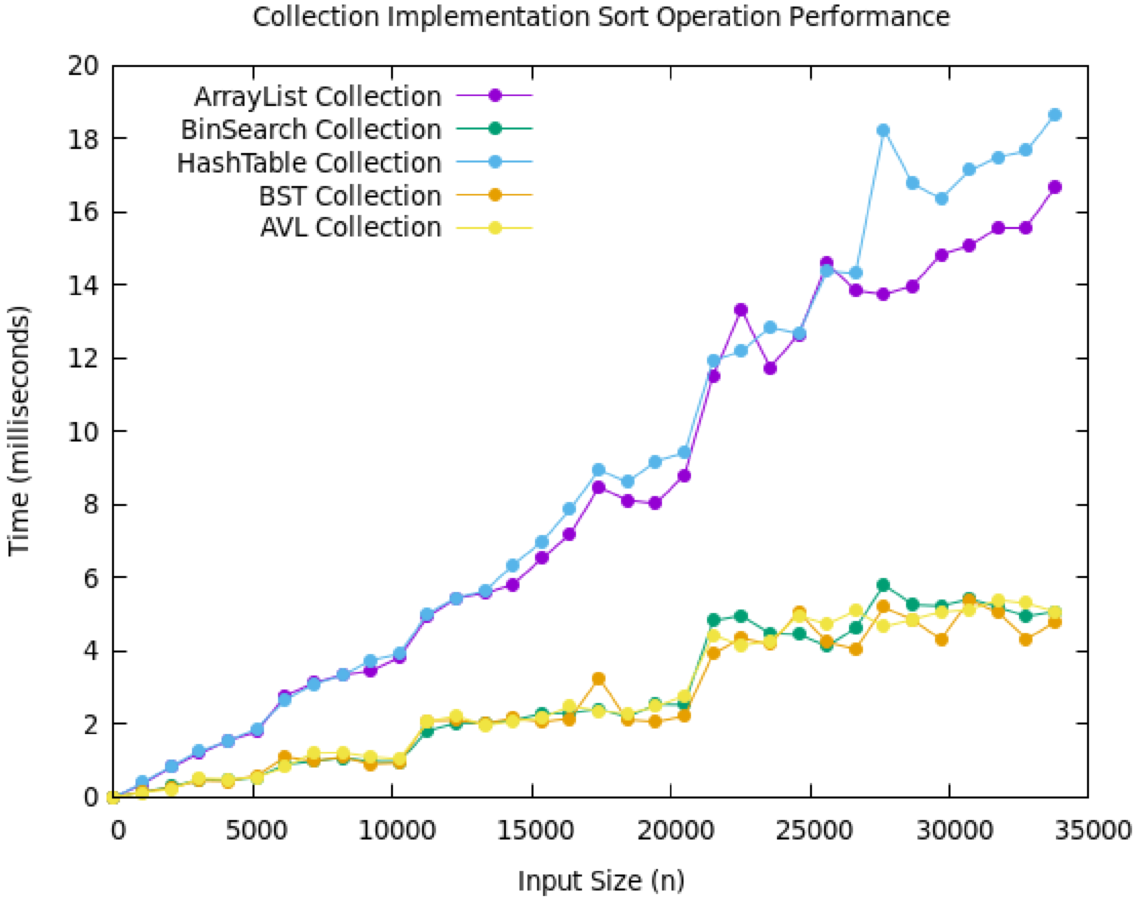

Collection Implementation Remove Operation Performance
ArrayList Collection
BinSearch Collection
HashTable Collection
BST Collection
AVL Collection
Time (milliseconds)
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0
5000
10000
15000
20000
25000
30000
35000
Input Size (n)

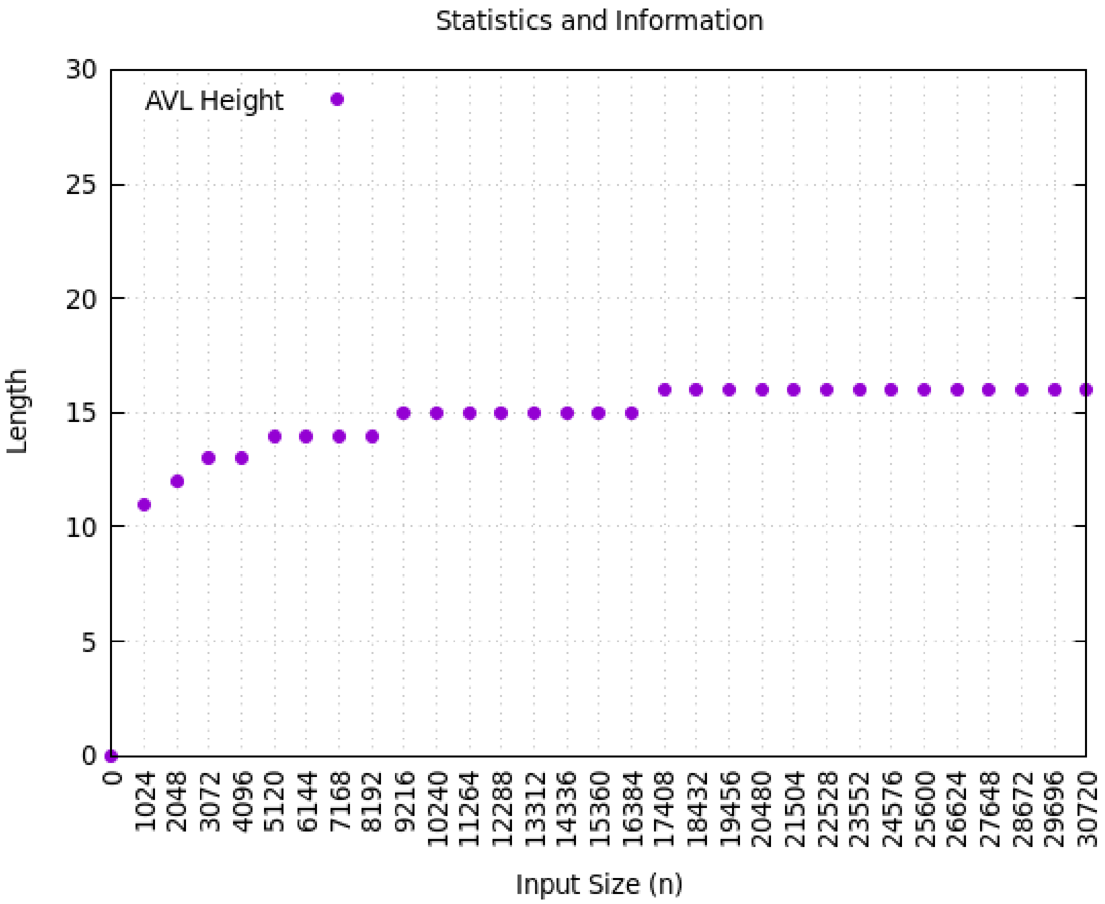

Collection Implementation Add Operation Performance
ArrayList Collection
BinSearch Collection
HashTable Collection
BST Collection
AVL Collection
Time (milliseconds)
1.6
1.4
1.2
1
0.8
0.6
0.4
0.2
0
0
5000
10000
15000
20000
25000
30000
35000
Input Size (n)


Collection Implementation Find Range Operation Performance
Time (milliseconds)
Input Size (n)
0
1
2
3
4
5
6
0
5000
10000
15000
20000
25000
30000
35000
ArrayList Collection
BinSearch Collection
HashTable Collection
BST Collection
AVL Collection


Collection Implementation Find Value Operation Performance
Time (milliseconds)
Input Size (n)
ArrayList Collection
BinSearch Collection
HashTable Collection
BST Collection
AVL Collection
0
0.05
0.1
0.15
0.2
0.25
0.3
0.35
0.4
0
5000
10000
15000
20000
25000
30000
35000

\


Collection Implementation Sort Operation Performance
ArrayList Collection
BinSearch Collection
HashTable Collection
BST Collection
AVL Collection
Time (milliseconds)
0
2
4
6
8
10
12
14
16
18
20
0
5000
10000
15000
20000
25000
30000
35000
Input Size (n)



Functions	**Array List**	**LinkedList**	**Sorted Array**	**Hash Table**	**BST**	**AVL**
Add	O(1)	O(1)	O(n)	O(1)	O(n)	O(log n)
Remove	O(n)	O(n^2)	O(n)	O(1)	O(n)	O(log n)
Find Val	O(n)	O(n^2)	O(log n)	O(1)	O(n)	O(log n)
Find Range	O(n)	O(n^2)	O(n)	O(n)	O(n)	O(n)
Sort	O(n^2)	O(n^2)	O(n)	O(n^2)	O(n)	O(n)