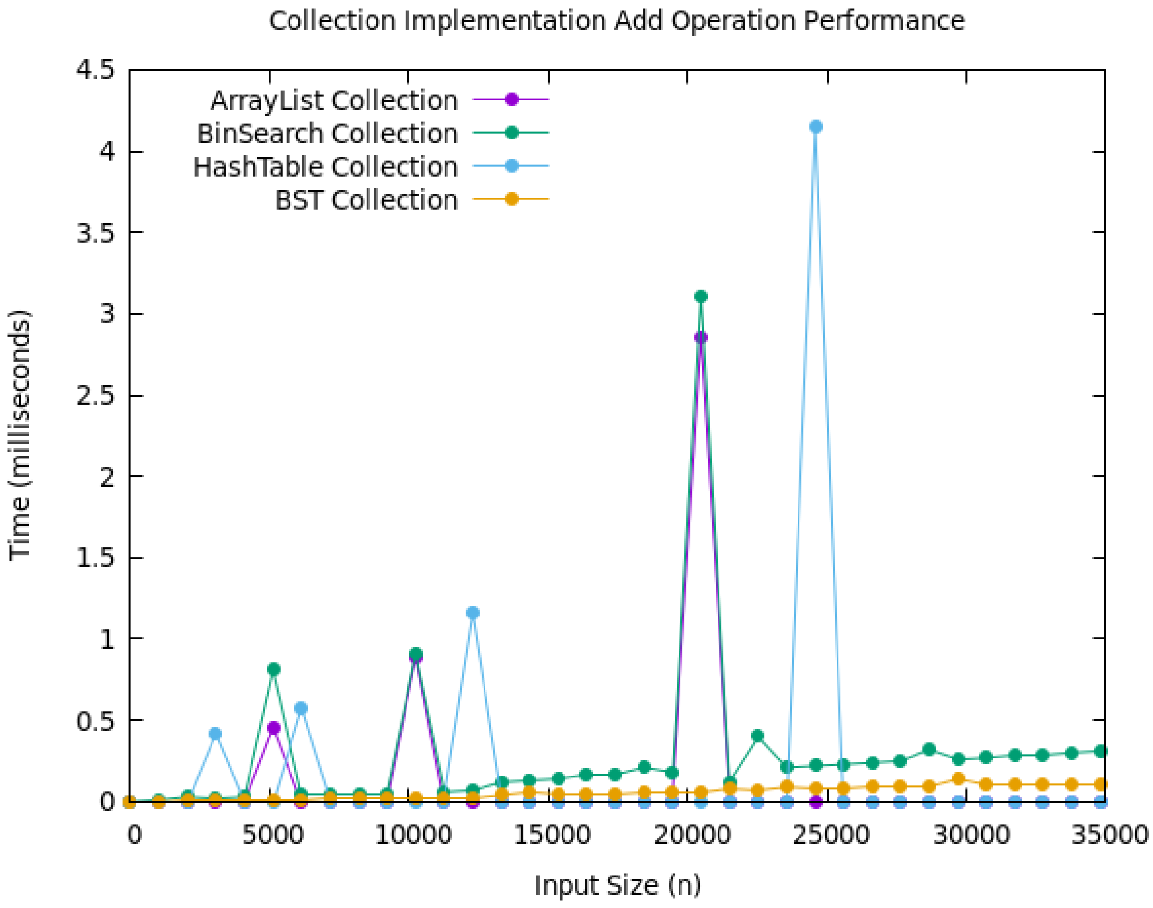
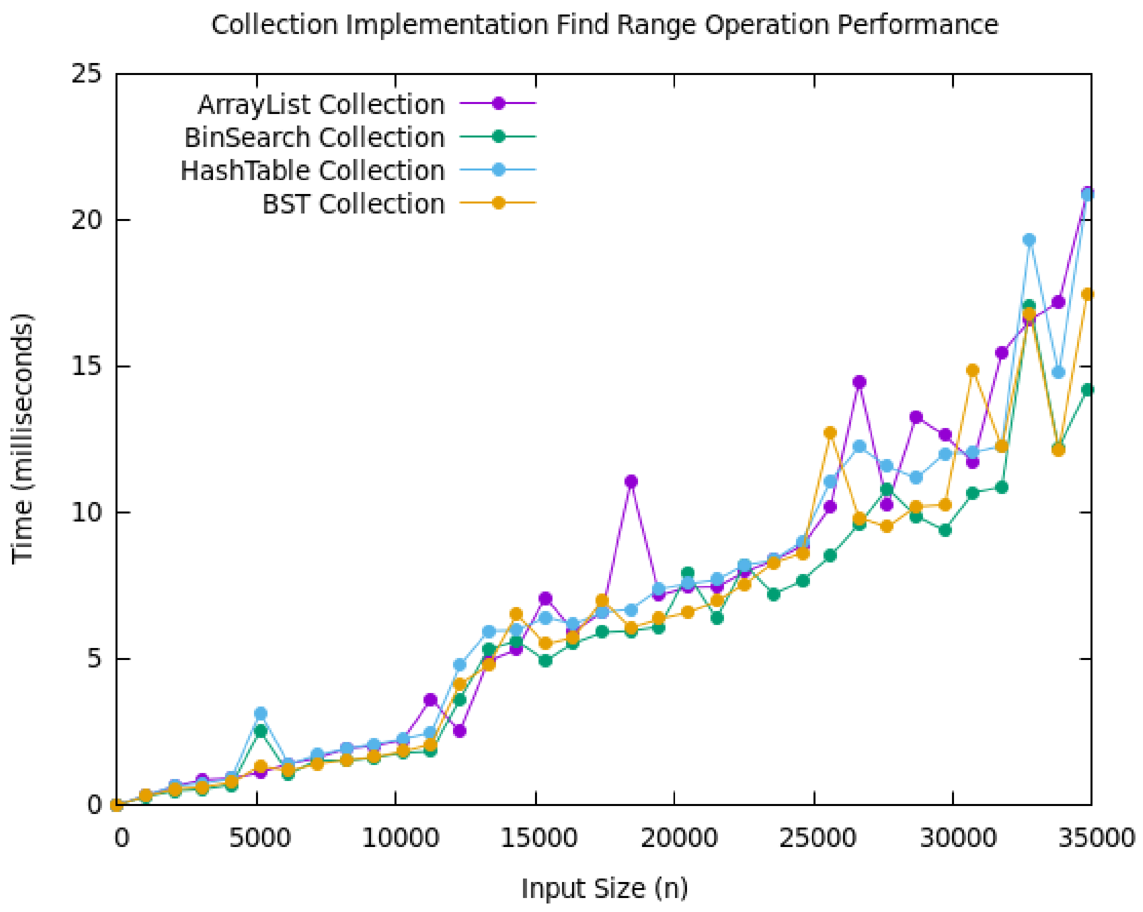
Matthew Moore

11/12/20

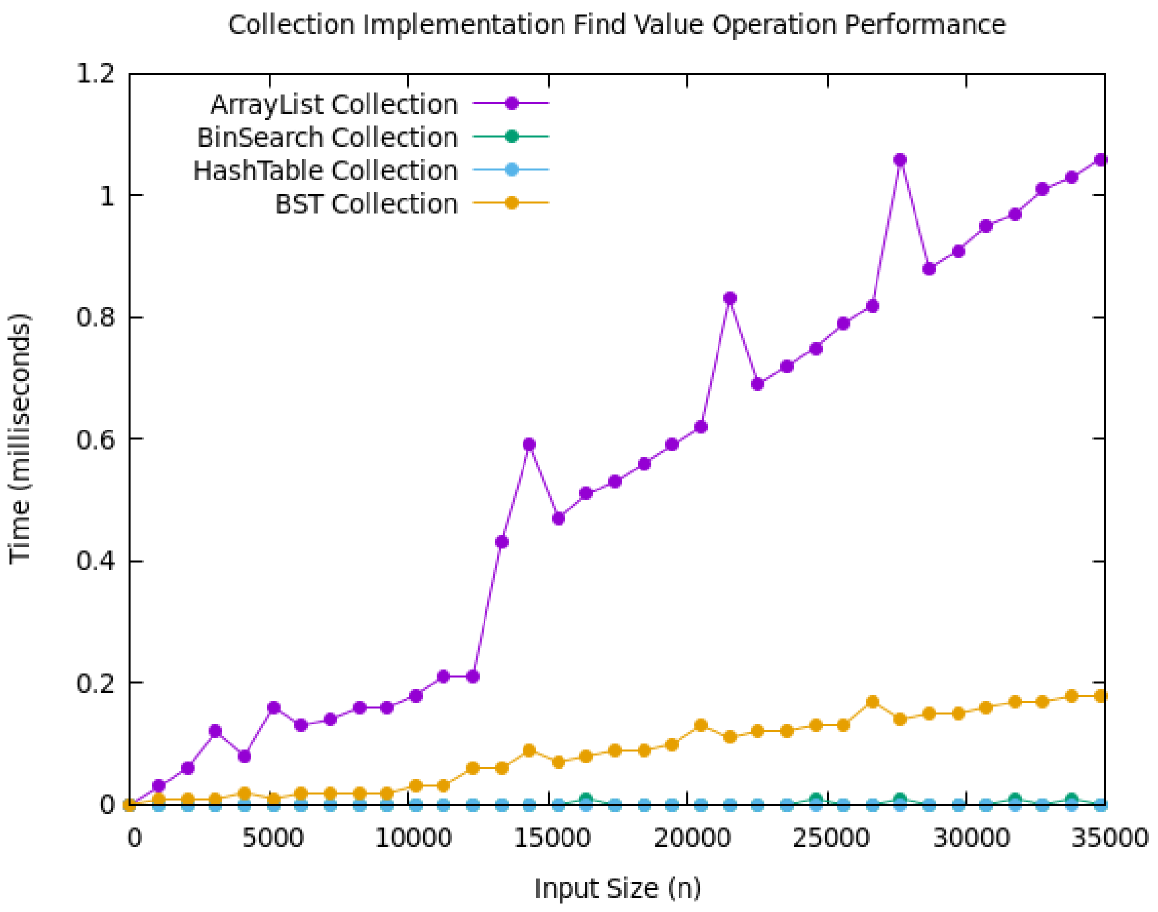
HW 7 Graph Analysis



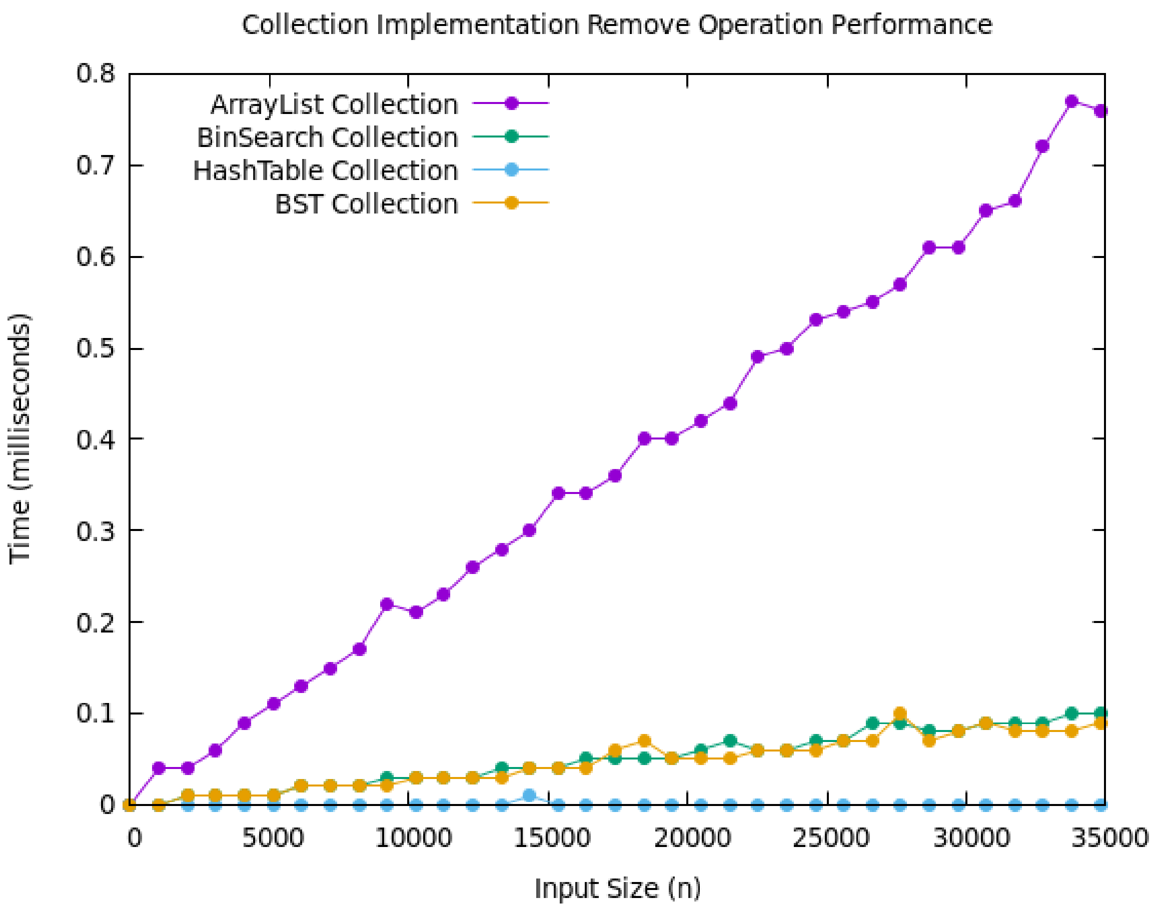
The Binary Search Tree Collection has O(log n) performance for the add operation. This is reflected by the graph and we also know this because the function allows for traversal down a single path of the binary search tree. The length of any path in a binary search tree will be log(n) in the best case scenario. In this specific example, the binary search tree is pretty well balanced, but a poorly balanced tree could have its add operation become O(n) if a single path contains all the nodes.



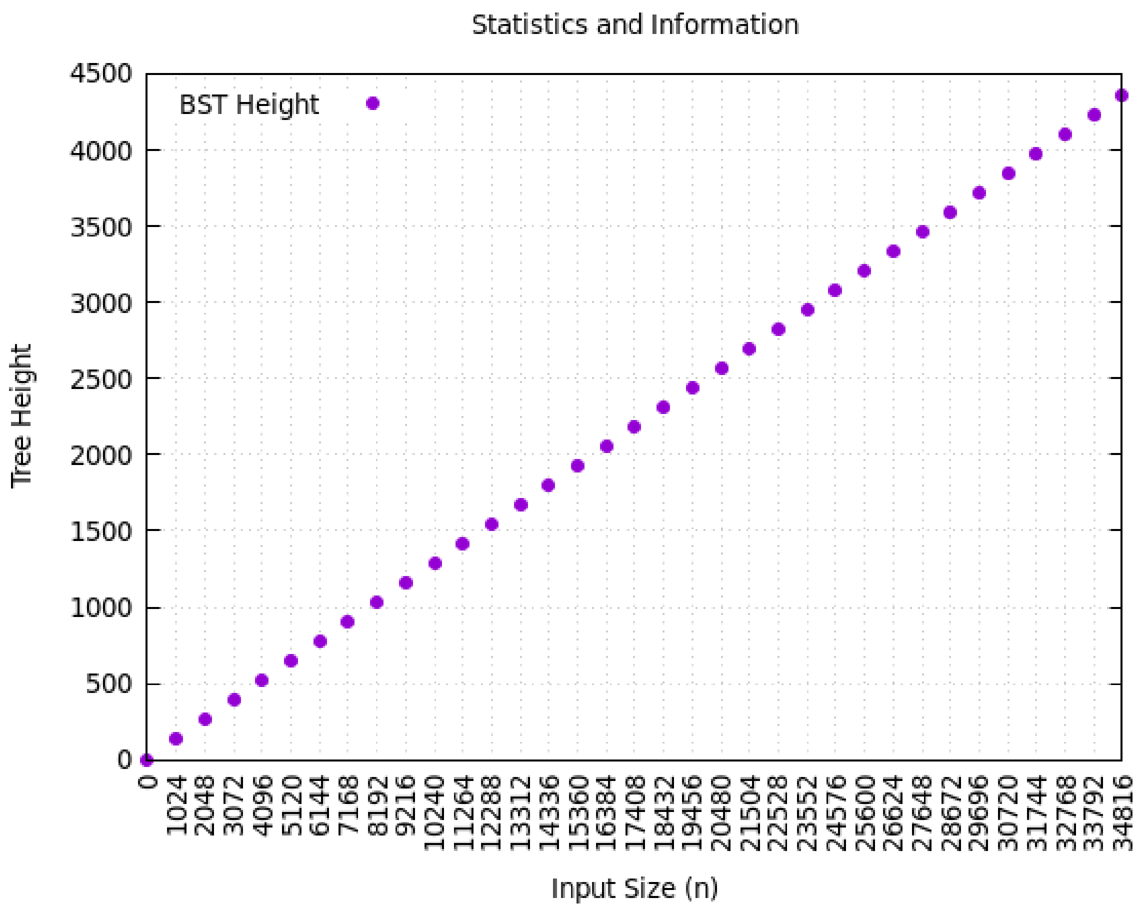
In the Find Range operation, we know that none of our options are very favorable because no matter what there will be a traversal through all the nodes within the range to add them to the array list. For Binary Search Collection, this is a O(n) operation most of the time depending on what the range size is for the given function call.



The Binary Search Tree Collection has a similar complexity to the Hash Table Collection. In the average case, the Binary Search Tree will have a complexity of O(log n) because there is a need for a traversal down a single path, which will be of size log(n) to find the correct element. In this specific example, we can see that the function is operating closer to its best case scenario of O(1) where the element being searched for happens to be toward the top of the binary search tree.



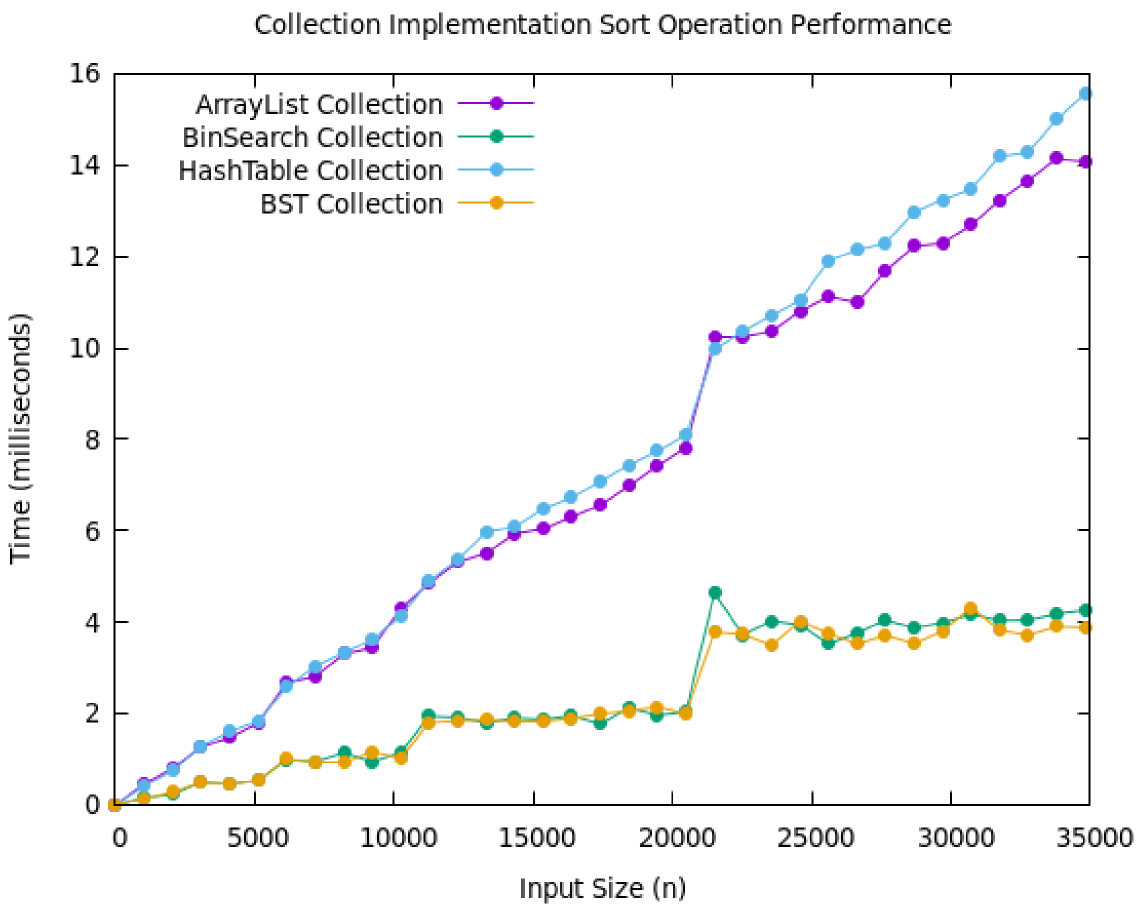
We can see that the trend for the Binary Search Collection is on the order of O(log n) on average. There is a need for traversal down a path to find the element to remove. Then, from there it is always going to be a couple more manipulations to rearrange the binary search tree to maintain its form and sorted nature. This specific scenario depicts the Binary Search Tree closer to the best case scenario of O(1) likely because the elements being removed are close to the root of the tree.



Looking at the Binary Search Collection tree's height we can see that it is increasing in a very linear fashion when elements are being added. This shows how the tree is relatively inefficient with large data sized because the height could potentially be only the value of log n if tracked appropriately. Without any constraints on the height of the tree, things get out of control and the tree becomes more inefficient.

Chart:

Operation	ArrayList	LinkedList	SortedArray	HashTable	BST
Add	O(1)	O(1)	O(n)	O(1)	O(n) Avg: log n
Remove	O(n)	O(n)	O(n)	O(1)	O(n) Avg: log n
Find-value	O(n)	O(n)	O(n)	O(1)	O(n) Avg: log n
Find-range	O(n)	O(n)	O(n)	O(n)	O(n)
Sort	O(n^2)	O(n^2)	O(n)	O(n^2) *quick sort*	O(log n)



Finally, the sort operation has a very similar complexity to the Binary search Collection sorting method. This is because they both are using the exact same type of algorithm to sort the list of elements. They both already have the elements in sorted order because of the way they are inserted into the list. Thus, all that is needed is for them to be individually placed into the array list. This operation will therefore be O(N) no matter what.