Nickolas Komarnitsky

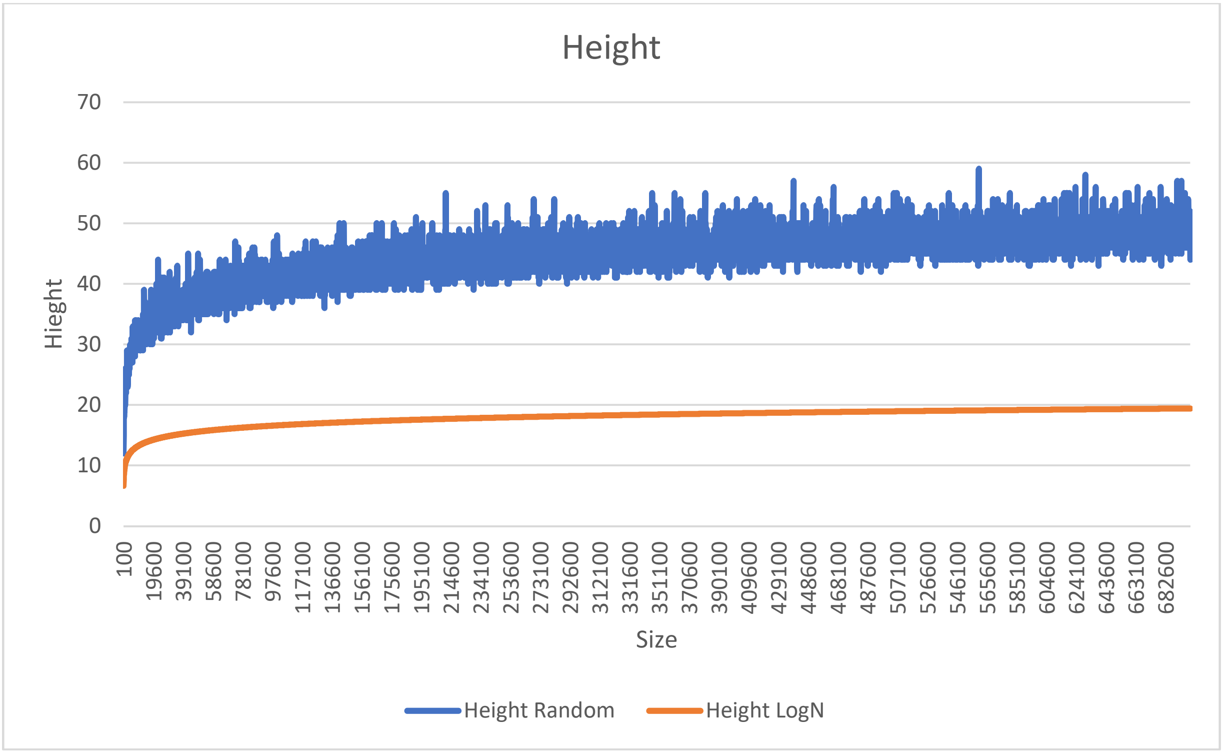
3/2/17

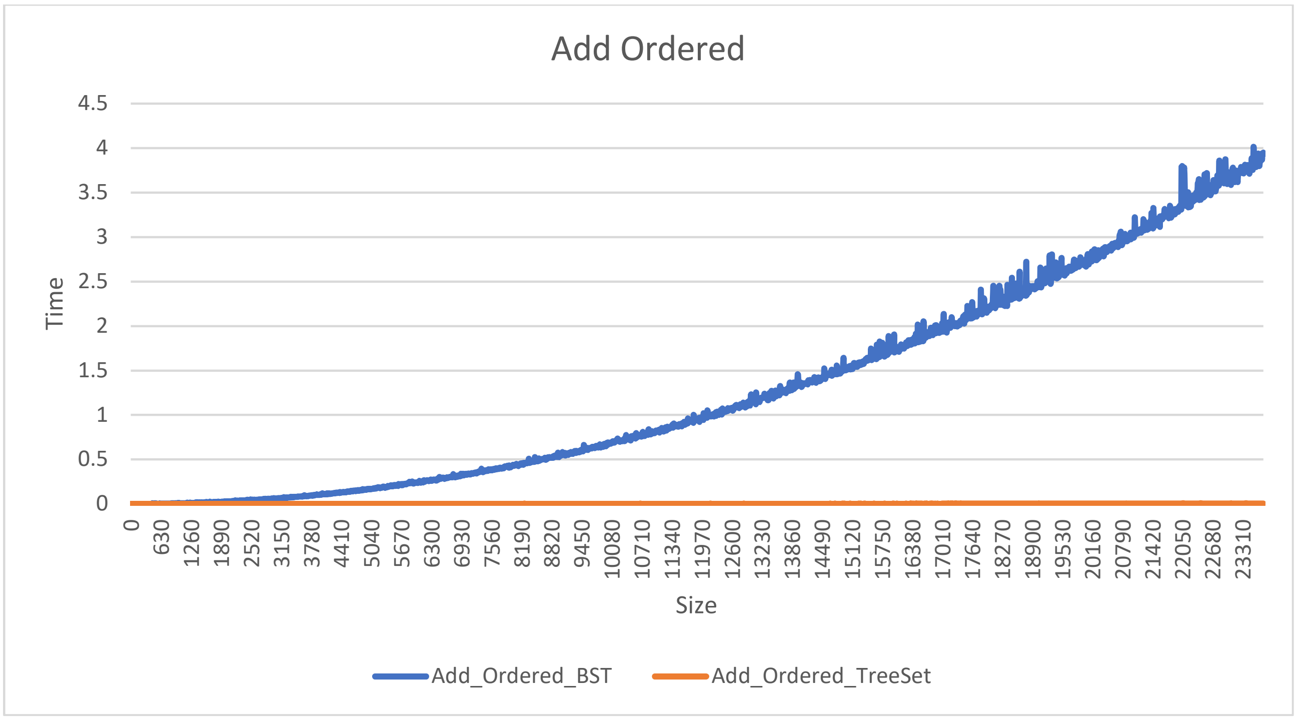
2420

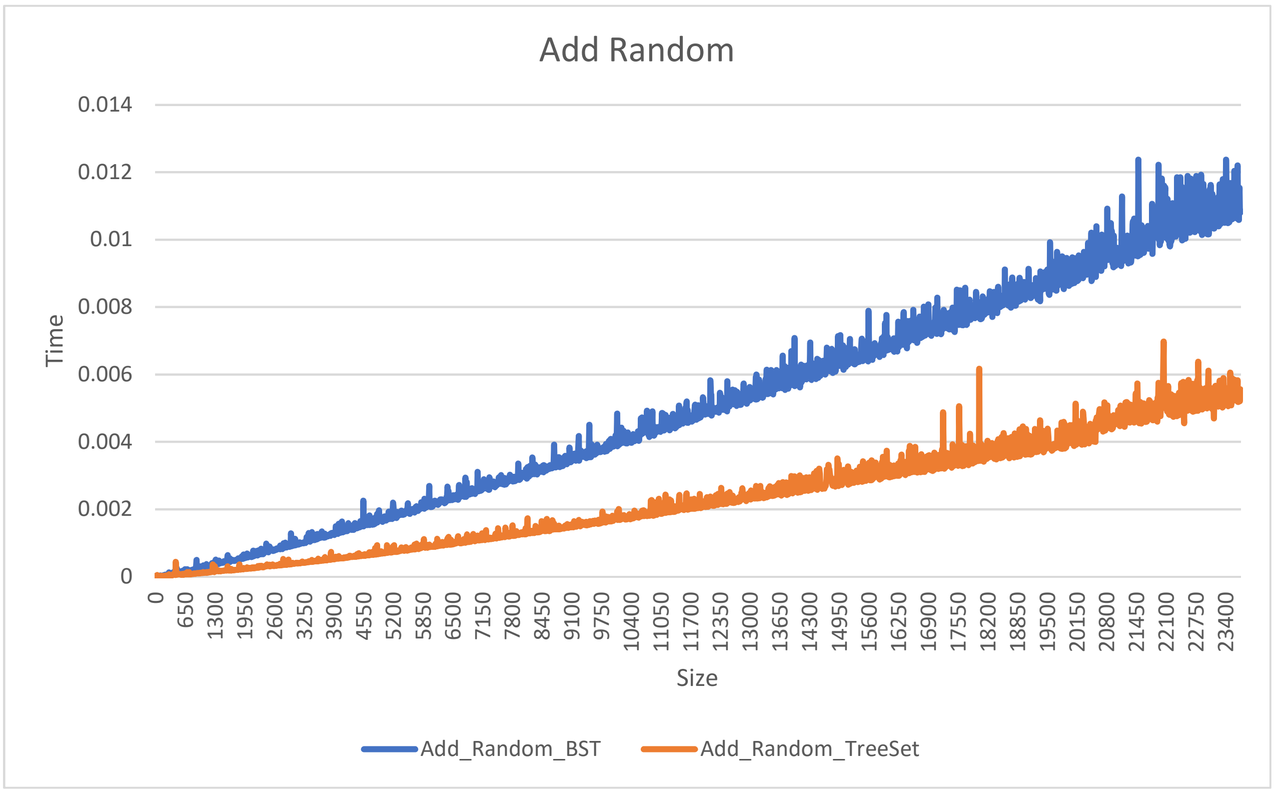
Assignment 07

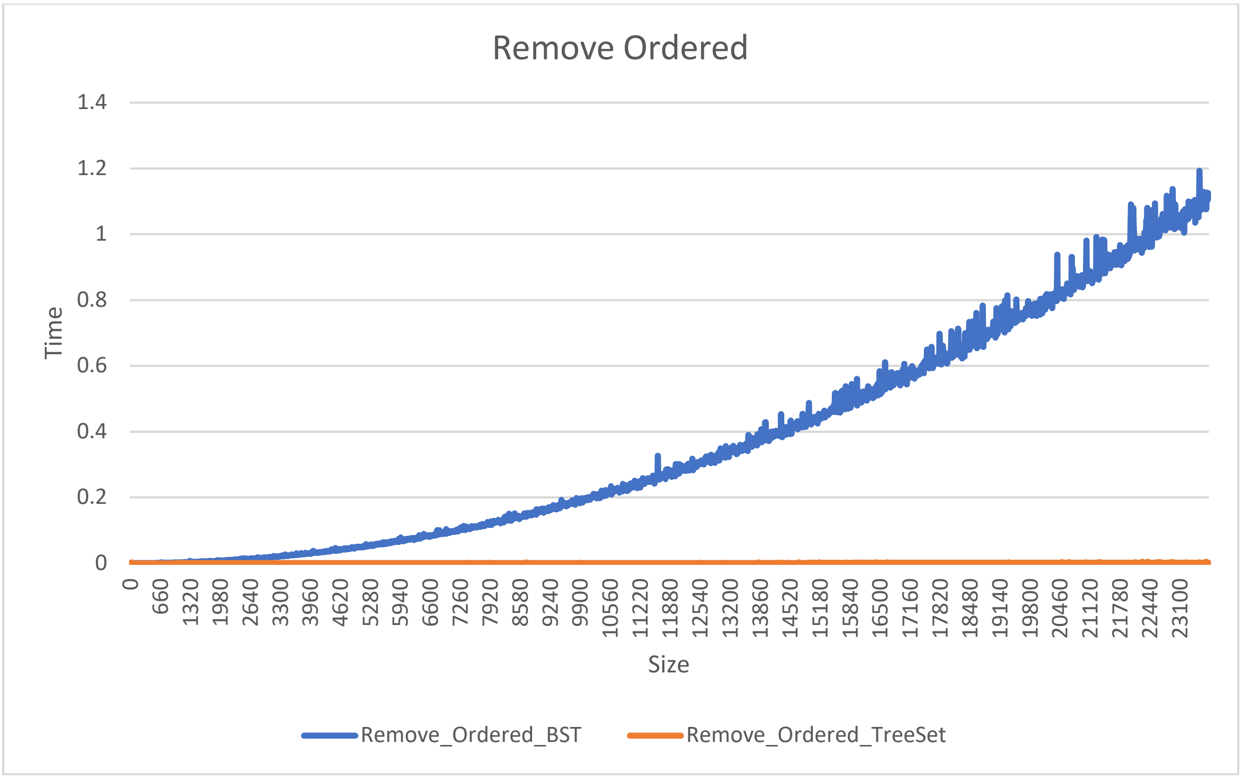
Analysis

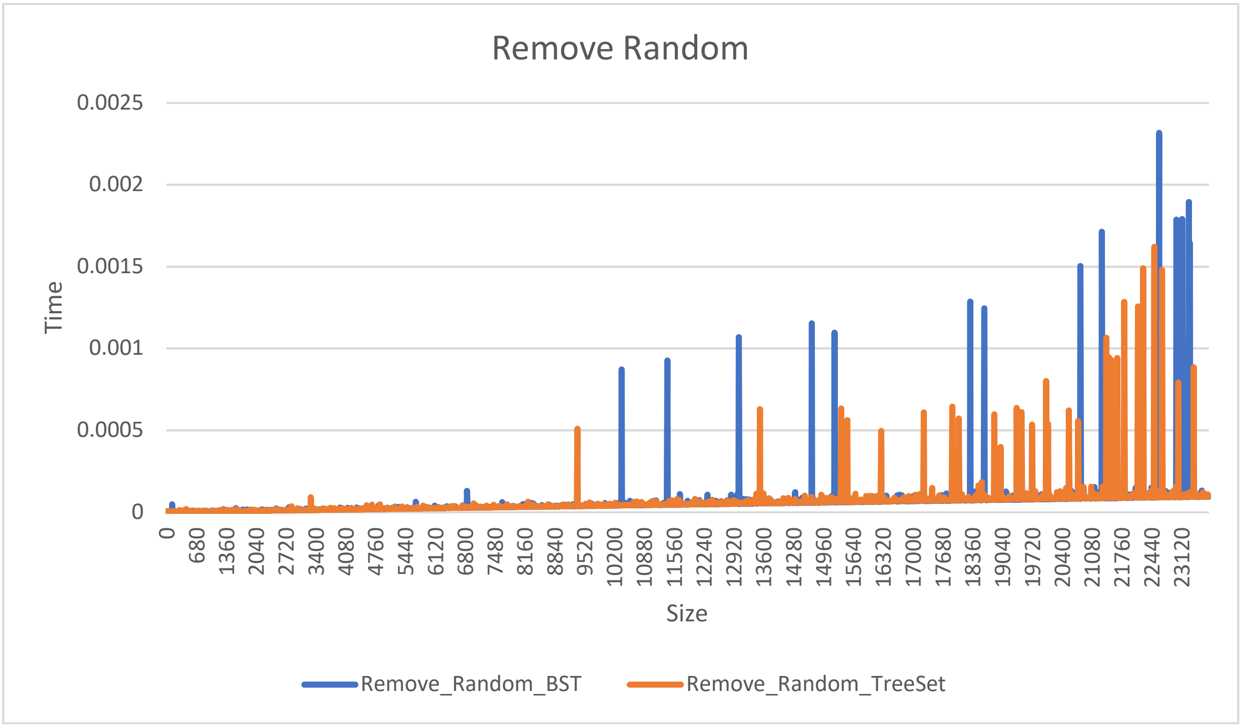
Graphs

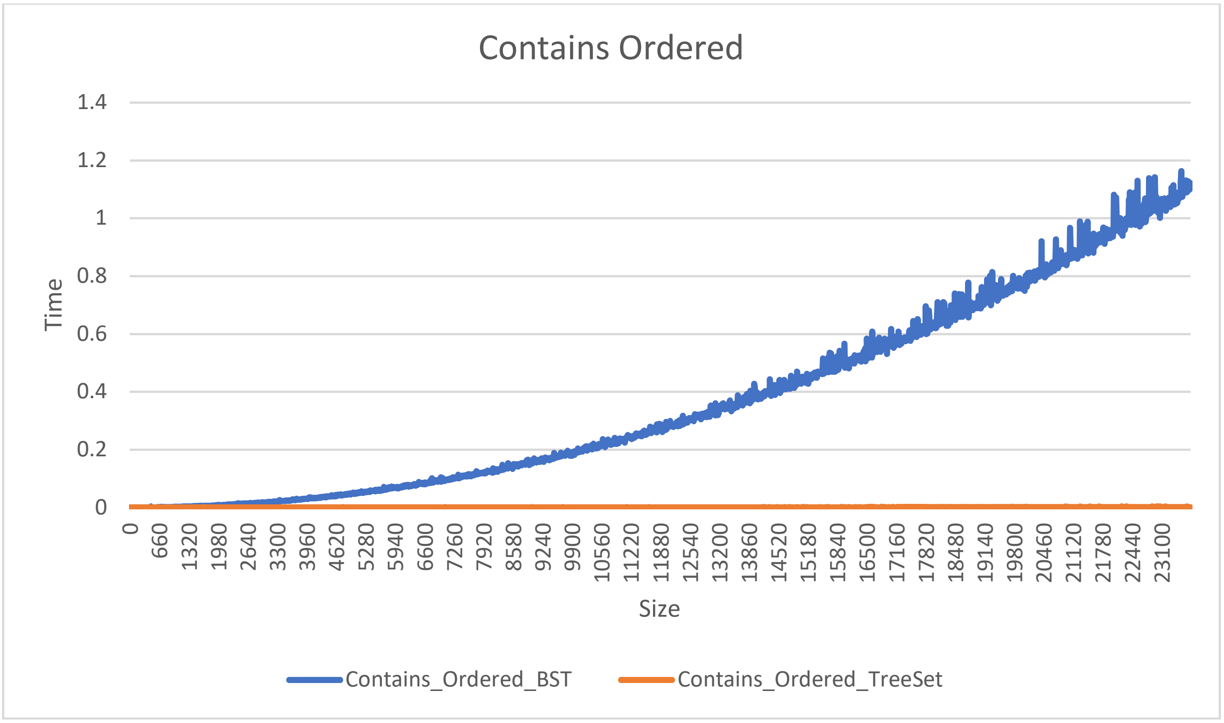


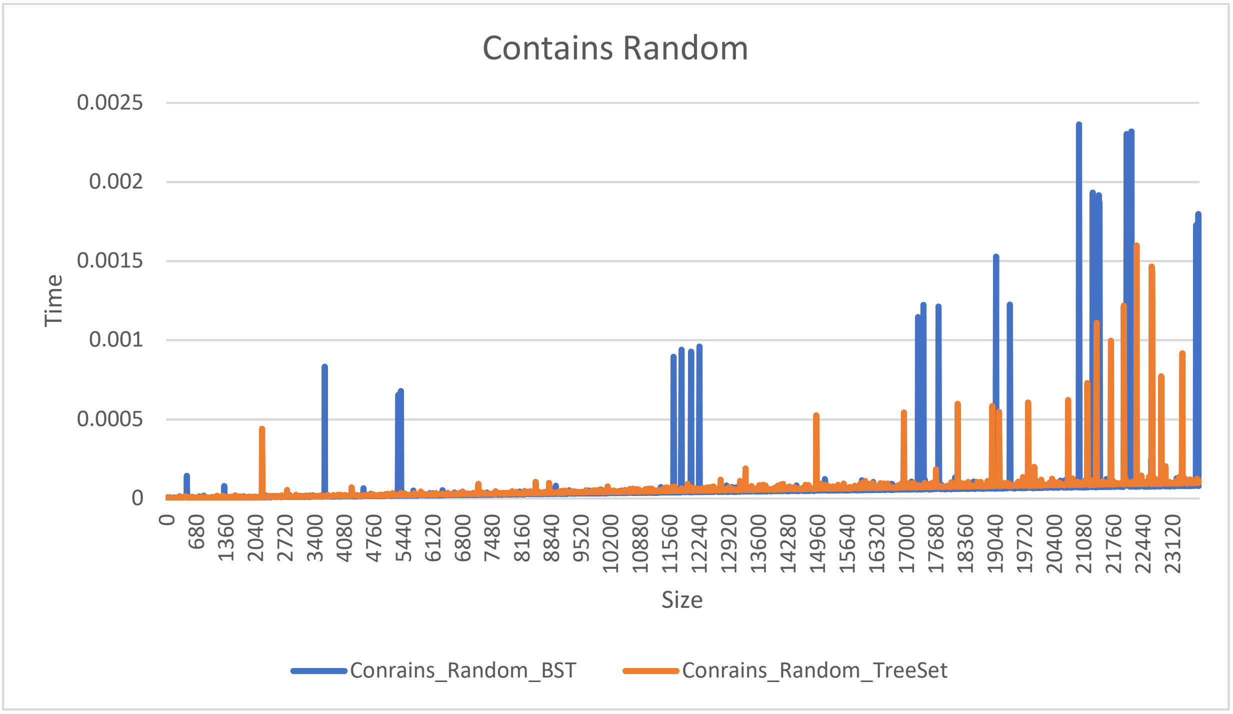

Add Ordered
Time
4.5
4
3.5
3
2.5
2
1.5
1
0.5
0
0
630
1260
1890
2520
3150
3780
4410
5040
5670
6300
6930
7560
8190
8820
9450
10080
10710
11340
11970
12600
13230
13860
14490
15120
15750
16380
17010
17640
18270
18900
19530
20160
20790
21420
22050
22680
23310
Size
Add_Ordered_BST
Add_Ordered_TreeSet

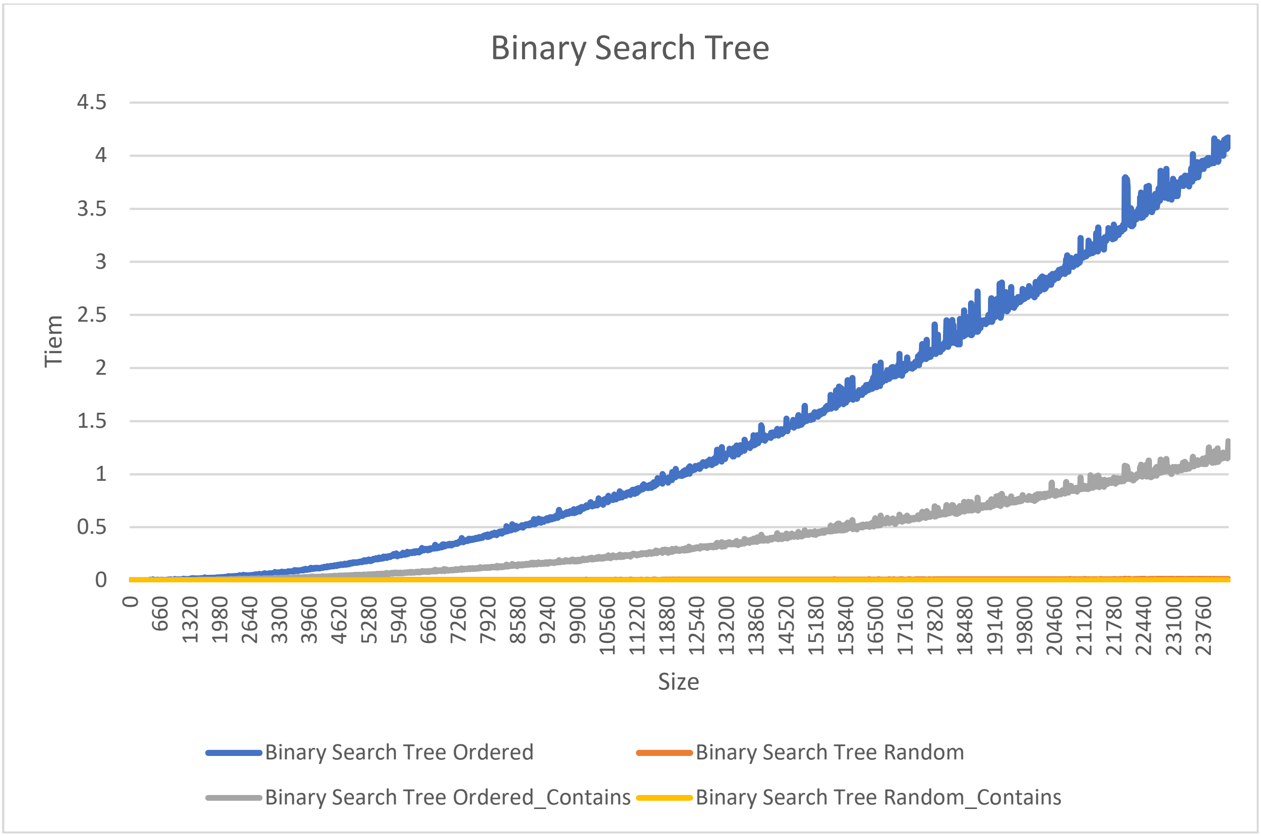

Add Random
Time
0.014
0.012
0.01
0.008
0.006
0.004
0.002
0
0
650
1300
1950
2600
3250
3900
4550
5200
5850
6500
7150
7800
8450
9100
9750
10400
11050
11700
12350
13000
13650
14300
14950
15600
16250
16900
17550
18200
18850
19500
20150
20800
21450
22100
22750
23400
Size
Add_Random_BST
Add_Random_TreeSet


Remove Ordered
Time
1.4
1.2
1
0.8
0.6
0.4
0.2
0
0
660
1320
1980
2640
3300
3960
4620
5280
5940
6600
7260
7920
8580
9240
9900
10560
11220
11880
12540
13200
13860
14520
15180
15840
16500
17160
17820
18480
19140
19800
20460
21120
21780
22440
23100
Size
Remove_Ordered_BST
Remove_Ordered_TreeSet


Remove Random
Time
0.0025
0.002
0.0015
0.001
0.0005
0
0
680
1360
2040
2720
3400
4080
4760
5440
6120
6800
7480
8160
8840
9520
10200
10880
11560
12240
12920
13600
14280
14960
15640
16320
17000
17680
18360
19040
19720
20400
21080
21760
22440
23120
Size
Remove_Random_BST
Remove_Random_TreeSet


Contains Ordered
Time
1.4
1.2
1
0.8
0.6
0.4
0.2
0
0
660
1320
1980
2640
3300
3960
4620
5280
5940
6600
7260
7920
8580
9240
9900
10560
11220
11880
12540
13200
13860
14520
15180
15840
16500
17160
17820
18480
19140
19800
20460
21120
21780
22440
23100
Size
Contains_Ordered_BST
Contains_Ordered_TreeSet


Contains Random
Time
0.0025
0.002
0.0015
0.001
0.0005
0
0
680
1360
2040
2720
3400
4080
4760
5440
6120
6800
7480
8160
8840
9520
10200
10880
11560
12240
12920
13600
14280
14960
15640
16320
17000
17680
18360
19040
19720
20400
21080
21760
22440
23120
Size
Conrains_Random_BST
Conrains_Random_TreeSet



In this assignment, I learned how to construct a binary tree. I learned the coding for a binary tree and I made timing code a little bit more toned, to work better. I learned the difference in time that a Binary Search Tree and a Balanced Tree have. The binary tree we made was unbalanced, but I also looked into how to make one that is balanced. The experiments that I ran came out to show graphs with log(n) complexity for adding and contains when the items are added in order. When items were added randomly the graph came out O(N), taking much less time to find items as well. For the tree, without balancing implemented, adding items in order was much worse than adding them randomly in all cases. The TreeSet, which does balance it, was almost always O(N). The height of the tree when added randomly was about double of the ideal height of the tree when perfectly balanced.

How do the classes you implemented interact with each other?

How does the Node class "support" the BST Class?

How does the BST "support" the Sorted Set?

The Spell Checker class uses a tree of a dictionary file to check if it has the word. The Binary Search Tree class has an private static inner class Node that acts as the container for the data. It has a root Node that has a left and right Node inside it that point to the rest of the tree. The Binary search tree uses the methods from the Sorted Set class; add removes, contains, etc. because it is a data structure. Sorted Set is all of the methods needed for a data set. The Binary search tree is simply a way of storing data in such a way as to allow us to access items quickly by putting them in certain places.

Discuss why BSTs are naturally recursive.

Where in this project did you use recursion? Where did you use iteration?

Describe an example of an iterative or recursive method that you could easily change to recursive or iterative? Why are some methods easily interchangable and some not so? What is an example of a non-interchangable method?

BST's are naturally recursive because of how it is structured. When calling a function you want to do the same thing to multiple nodes, "move" throughout the tree until you get where you want to. For the add, remove, contains methods we used recursion. Iteration was only used in the containsAll, addAll, removeAll methods in order to do the specified function on all items in the given array. Almost all of the methods could be made recursive or iterative. The add method for example, could simply have a traveler Node, that changes until you're at where you want to be and puts in the data. Some methods you only want to do once though, so they wouldn't be recursive. In Node, the Successor method only needs to be called itself once, it then uses the getLeftMost or getRightMost functions to find the successor and return it, but the function itself is just a simple check, no need for recursion. NumChildren is non-interchangeable as well as it is simply a check on the current Nodes values in order to determine how many children the current Node has, 0,1, or 2.

How much time did you spend on this project?

What was the most time consuming part of the programming? What can you do better in the future? Did you plan enough? Did you allocate enough time from the start of the week or did you wait until the last minute to get things done?

What problems came up that took disproportionate amounts of time?

Did writing the tests first help your development? In what way?

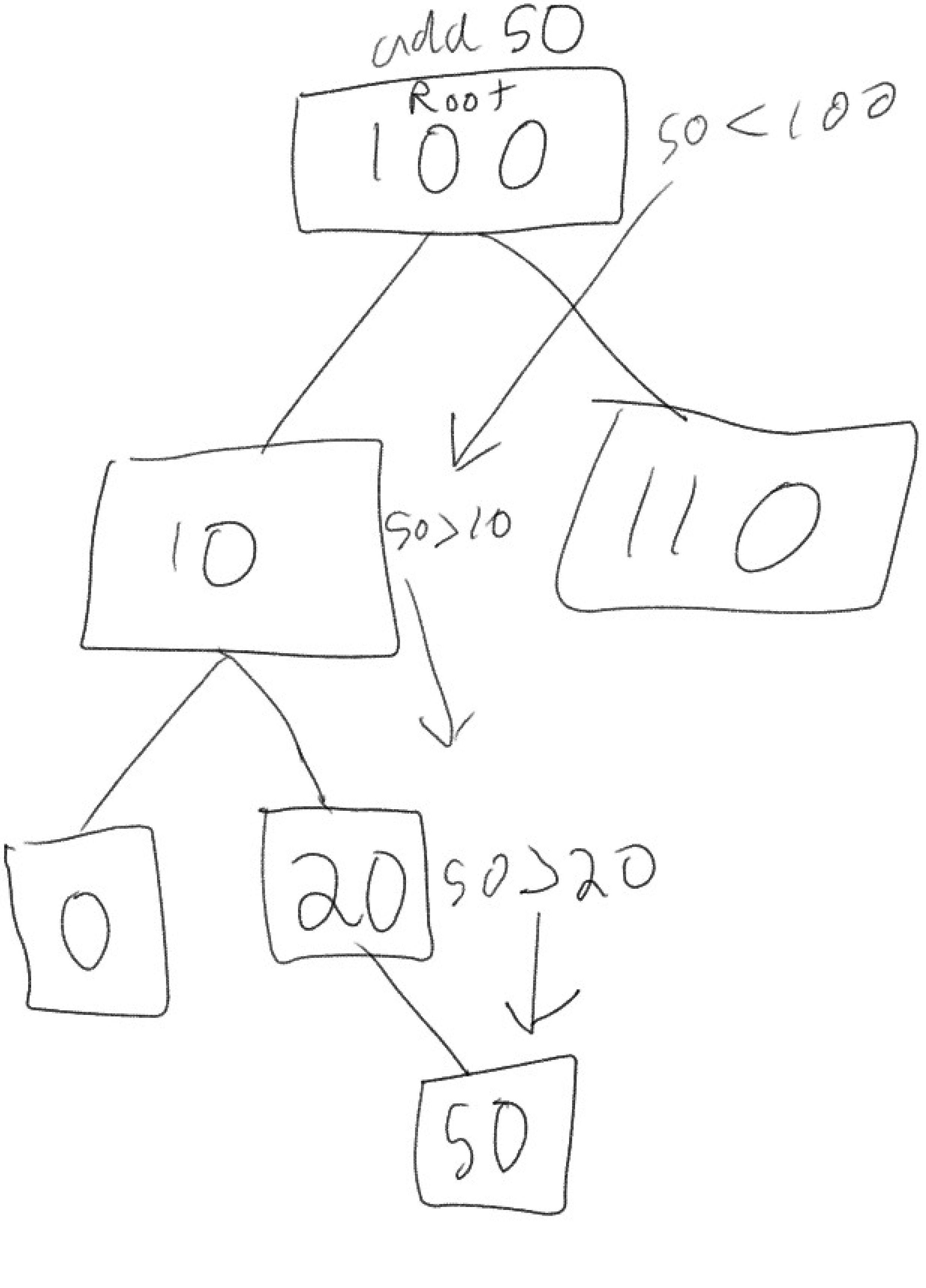
We spent about 10 hours on this assignment. The remove method was the most time consuming. We planned enough time, but the remove function wasn't working perfectly for a while. The other functions took very little time to complete, probably under an hour was spent on those. The remove function took a little while, drawing it out helping us to visualize and make it work well. Writing the tests first help identify and fix the edge cases for each method.

Many dictionaries are in alphabetical order. What problem will it create for a dictionary BST if it is constructed by inserting words in alphabetical order? Explain what you could do (did do?) to fix the problem.

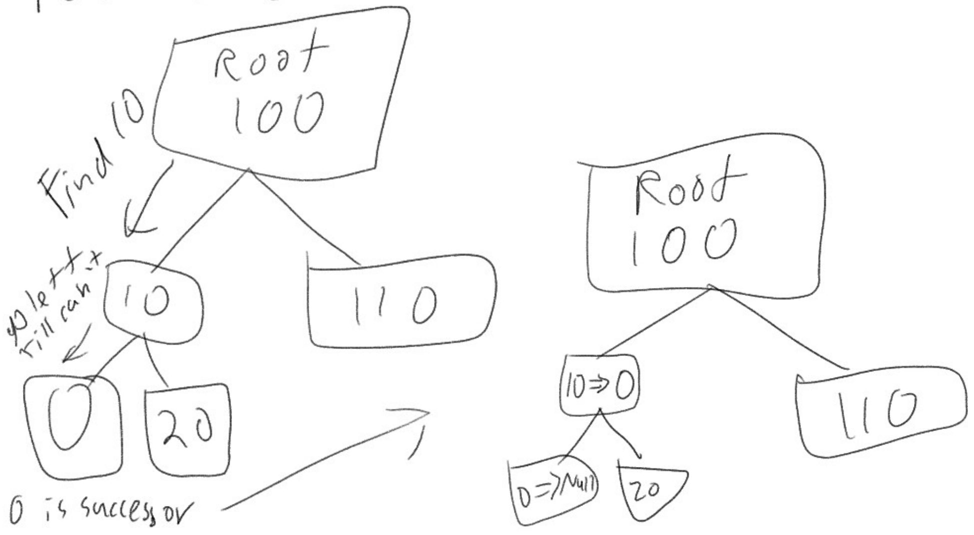
a) How "unbalanced" was the tree you created for the spell checker application? b) How do you know? c) What effect does this have on the efficiency of the BST.

Inserting into our unbalanced BST would have a problem, it would end up being a right heavy tree with no values on the left side. If the tree was balanced, repositioning items when added so as to make the tree balanced would fix this problem. Our code didn’t percolate up the tree to readjust the position of the items.

Drawings include the pictures you drew to help understand the process of insertion and deletion of nodes from a BST.


add 50
Root
100
50 < 100
10
50 > 10
110
0
20
50 > 20
50

Remove 10


Find 10
Root
100
go left till can't
10
110
0
20
0 is successor
Root
100
10⇒0
110
0⇒Null
20