In assignment P2, we implemented an randomized search tree (RST) on a binary search tree (BST) we constructed in assignment P1. The theoretical average time complexities for finding a value in a sorted and unsorted BST and RST are of log(n). Even though the worst time complexity for finding a value in an unsorted BST is O(n), this time complexity can be improved by implementing an RST to a BST. By performing this procedure, we have set the priority to a randomized value, and therefore, we expect to show that it is less likely that a randomized tree will be balanced. Below we compare the performance of the three types of data structures RST, BST, and SET to show how many comparisons it takes to search for a value in either sorted or unsorted order. There are a total of six cases to consider: (1) RST sorted, (2) RST unsorted, (3) BST sorted, (4) BST unsorted, (5) SET sorted, and (6) SET unsorted. Each of the six cases we take into account a benchmarking average number of comparisons for a successful find. For example, there are a total of five trials we used to determine the average comparisons and standard deviations for each data structure. The expected standard deviation results for the BST and SET average comparisons are 0. Whereas, the standard deviation expected results for the sorted RST and unsorted RST, BST, and SET will range between zero and one.

Figures (1-3) show our experimental values for running sorted and unsorted data on three type of data structures. The benchmarking program we designed to determine the time complexities for a unsorted and sorted BST, RST, and SET data structures is not accurate for calculating the standard deviation. For example, the standard deviation for BST has a standard deviation that does not follow the expected standard deviation trend. A reason for why this plot is not accurately following the expected trend could be due to implementing the standard deviation incorrectly within a series of loops. Though, when analyzing the average number of comparisons for BST, RST, and SET, the averages follow the expected time complexity trends. For instance, the average time cost function for BST is proportional to log(N), which implies that the number of nodes in the tree is proportional to the number of comparisons.

```
# Benchmarking average number of comparisons for successful find
# Data structure: rst
# Data: sorted
# N is powers of 2, minus 1, from 1 to 32767
# Averaging over 5 runs for each N
#
# N      avgcomps       stdev
   1    2            0
   3    3            0
   7    4.57143            0
   15     5.86667          0
   31     7.58065          0
   63     9.47619          0
   127     11.4488         0
```

```
   255    13.0275       0
   511    15.5988       0
   1023    17.6188      0
   2047    19.7587      0
   4095    22.6752      3.37175e-07
   8191    25.7826      4.76837e-07
   16383    28.2222     0
[tsrussel@acs-cseb260-40]:p2:830$ ./benchtree rst shuffled 32767 5

# Benchmarking average number of comparisons for successful find
# Data structure: rst
# Data: shuffled
# N is powers of 2, minus 1, from 1 to 32767
# Averaging over 5 runs for each N
#
# N      avgcomps      stdev
   1    2           0
   3    3           0
   7    4.14286           0
   15    6.4       1.19209e-07
   31    8.87097           0
   63    10.5079           0
   127    12.685           0
   255    13.6902       1.68587e-07
   511    17.3346       0
   1023    18.0557      0
   2047    20.0469      0
   4095    21.6681      0
   8191    24.4866      0
   16383    26.2415     4.76837e-07
[tsrussel@acs-cseb260-40]:p2:831$ ./benchtree bst shuffled 32767 5

# Benchmarking average number of comparisons for successful find
# Data structure: bst
# Data: shuffled
# N is powers of 2, minus 1, from 1 to 32767
# Averaging over 5 runs for each N
#
# N      avgcomps      stdev
   1    2           0
   3    3           0
   7    5.14286           0
   15    7.2       8.42937e-08
```

```
   31    8.22581         0
   63    9.88889         0
   127    12.1181        0
   255    14.2275        2.38419e-07
   511    16.1135        0
   1023    18.4604       0
   2047    20.681        0
   4095    22.8264       0
   8191    24.8465       0
   16383    26.8701      0
[tsrussel@acs-cseb260-40]:p2:832$ ./benchtree bst sorted 32767 5

# Benchmarking average number of comparisons for successful find
# Data structure: bst
# Data: sorted
# N is powers of 2, minus 1, from 1 to 32767
# Averaging over 5 runs for each N
#
# N      avgcomps       stdev
   1    2           0
   3    4           0
   7    8           0
   15    16         0
   31    32         0
   63    64         0
   127    128       0
   255    256       0
   511    512       0
   1023    1024          0
   2047    2048          0
   4095    4096          0
   8191    8192          0
   16383    16384        0
[tsrussel@acs-cseb260-40]:p2:833$ ./benchtree set sorted 32767 5

# Benchmarking average number of comparisons for successful find
# Data structure: set
# Data: sorted
# N is powers of 2, minus 1, from 1 to 32767
# Averaging over 5 runs for each N
#
# N      avgcomps       stdev
   1    2           0
```

```
   3    3          0
   7    4.14286          0
   15    5.26667          5.96046e-08
   31    6.35484          0
   63    7.4127           0
   127    8.44882         0
   255    9.47059         0
   511    10.4834         0
   1023    11.4907        0
   2047    12.4949        0
   4095    13.4972        1.68587e-07
   8191    14.4985        0
   16383    15.4992       0
   32767    16.4996       0
[tsrussel@acs-cseb260-40]:p2:834$ ./benchtree set shuffled 32767 5

# Benchmarking average number of comparisons for successful find
# Data structure: set
# Data: shuffled
# N is powers of 2, minus 1, from 1 to 32767
# Averaging over 5 runs for each N
#
# N      avgcomps       stdev
   1    2          0
   3    3          0
   7    4.28571          0
   15    5.06667         0
   31    6.19355         8.42937e-08
   63    7.30159         0
   127    8.23622        0
   255    9.23922        0
   511    10.2916        1.19209e-07
   1023    11.3196       0
   2047    12.2726       0
   4095    13.3111       0
   8191    14.3295       2.38419e-07
   16383    15.3662      2.38419e-07
   32767    16.3742      0
```

Figure 1. Sorted data structures (RST,BSTSET,)

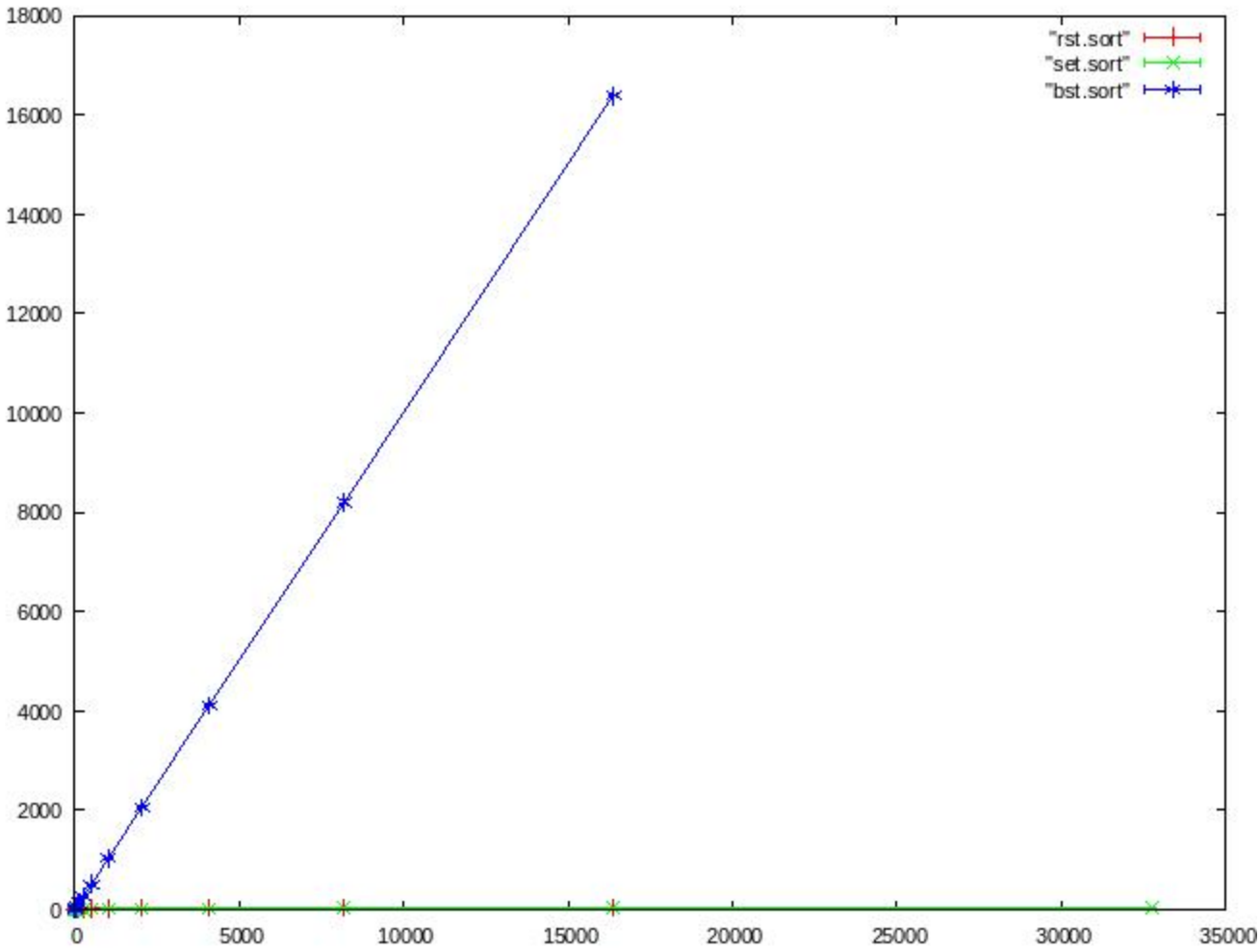
"rst.sort"
"set.sort"
"bst.sort"
18000
16000
14000
12000
10000
8000
6000
4000
2000
0
0
5000
10000
15000
20000
25000
30000
35000

Figure 2. unsorted data structures