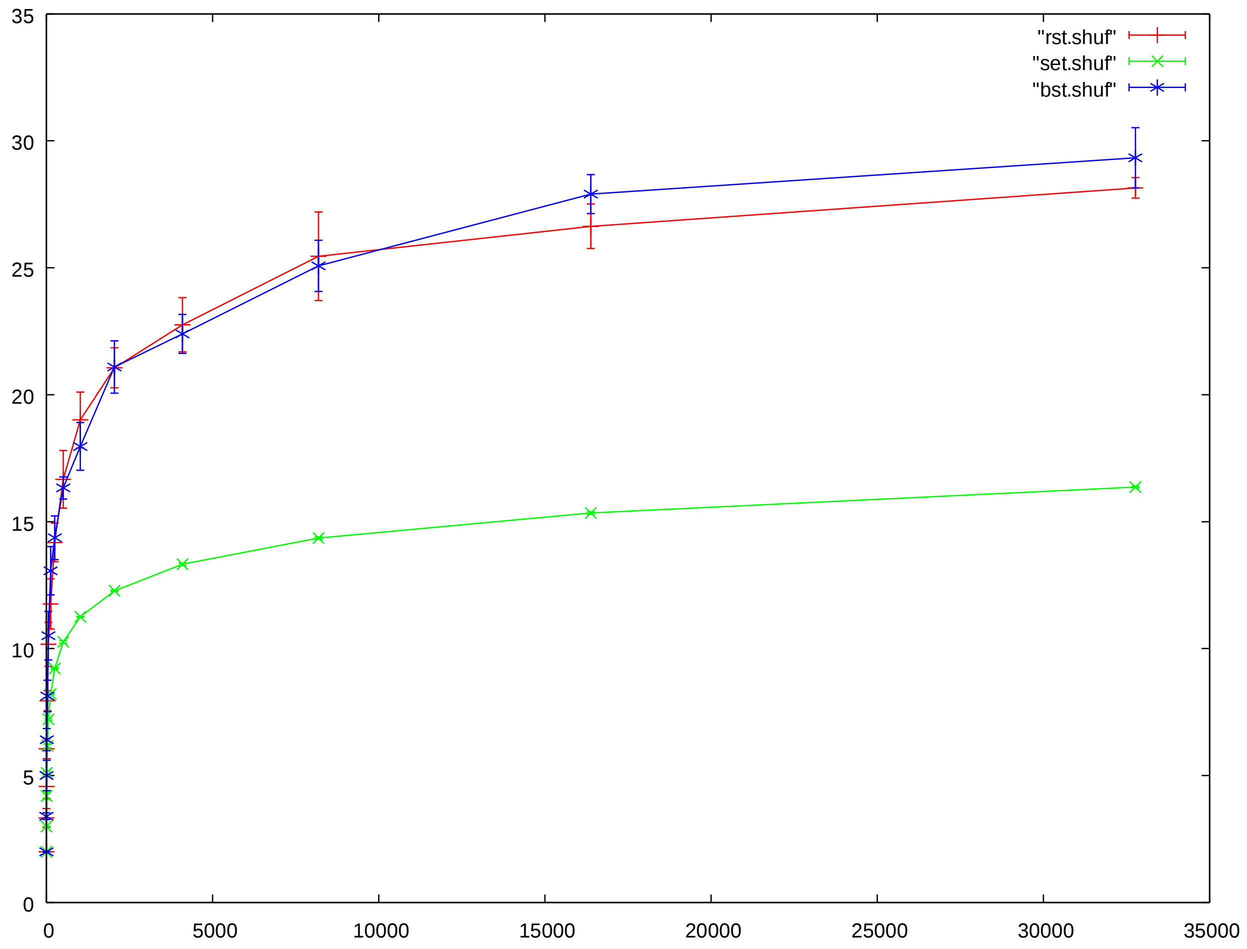
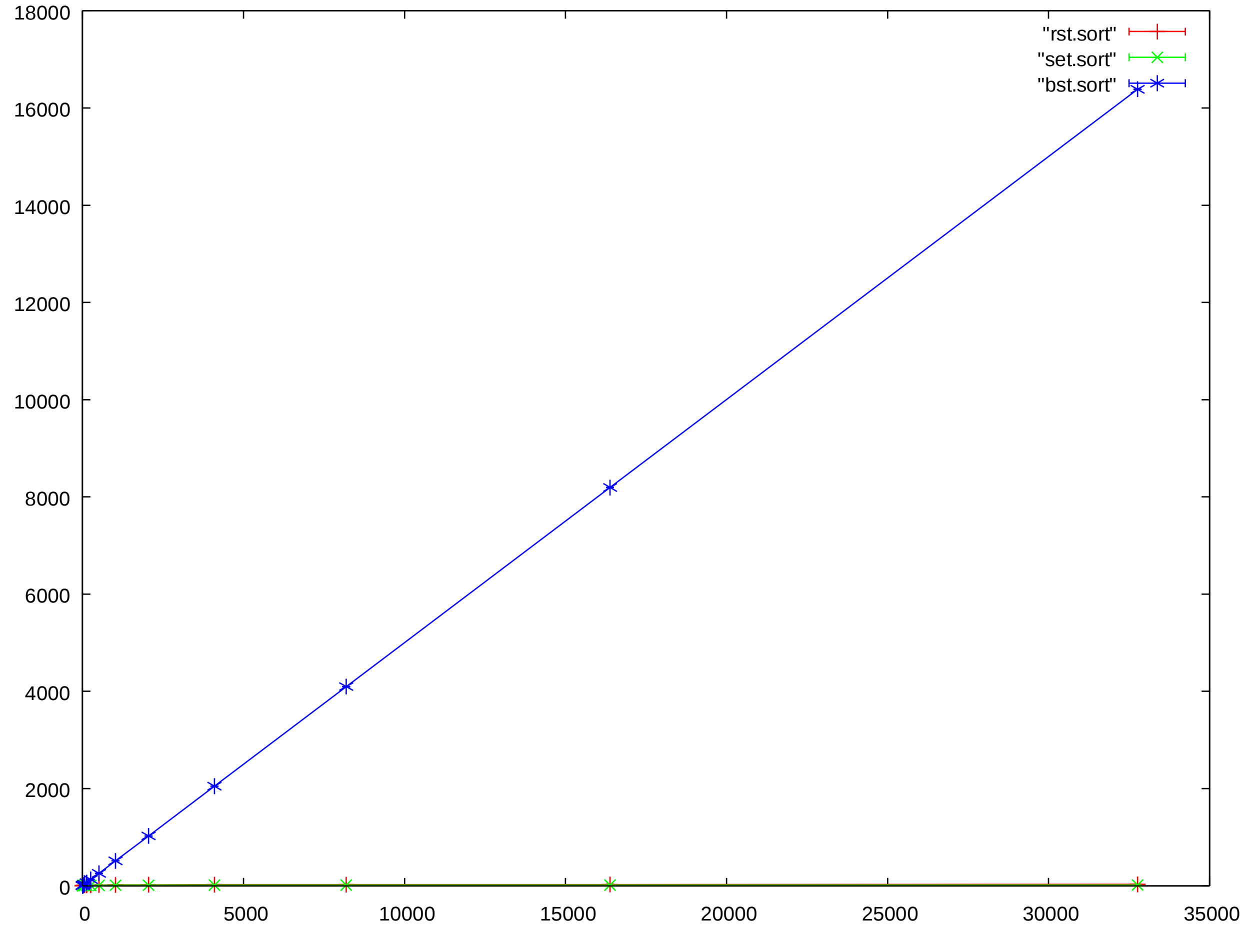

35
30
25
20
15
10
5
0
0
5000
10000
15000
20000
25000
30000
35000
"rst.shuf"
"set.shuf"
"bst.shuf"


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

Time Analysis Report

Lora Tam

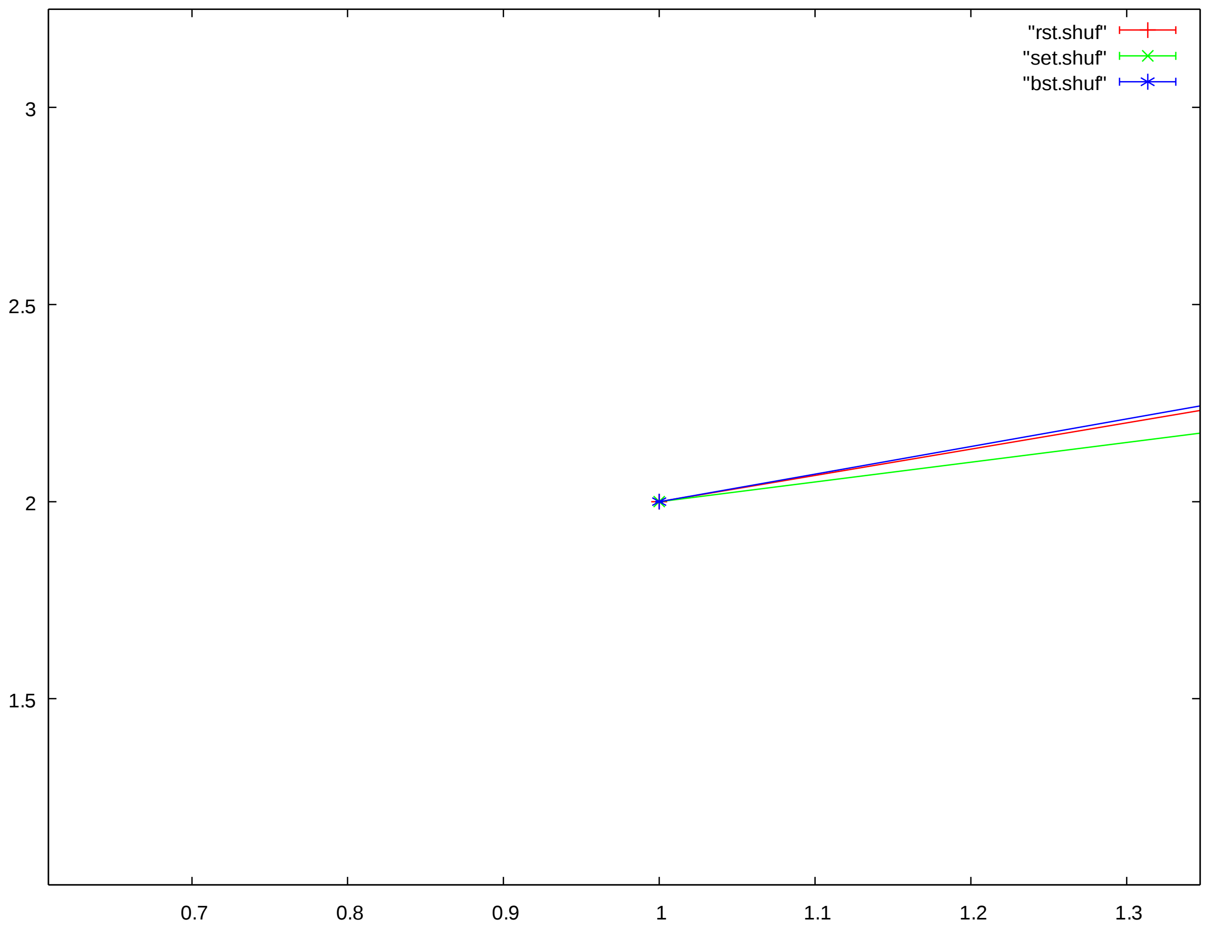
For the first graph, it is displaying the rst.shuf,bst.shuf, and set.shuf which are the outputs for
./benchtree rst/set/bst shuffled 32768 5.

For the second graph, it is displaying the rst.sort, bst.sort, and set.sort, which are the outputs for
./benchtree rst/bst/set sorted 32768 5.

Both of these graph agree with the theoretical big-o time cost analysis. For the first graph, it is showing unsorted and all of them should show O(log n) for insert which is shown above. These agree with the theoretical big o time analysis because in theory they should be O(log n).

For the second graph, it is sorted therefore BST should show O(n) because it is basically inserting like a list if the data is all sorted such that it will keep adding to the right if the next consecutive number is larger than the previous therefore it is big o of N. The others look logarithmic compared to bst and when enlarged shows that it is slightly O(logn) which is shown on graph 2. Due to RST self balancing because of the priority it will be log of o N for RST which is shown in the second graph. Therefore both of these graphs show or agree with the theoretical big 0 time cost analysis.

As the graph below shows, the shuffle of all three lines start at (1,2).


"rst.shuf"
"set.shuf"
"bst.shuf"