Evan Carey (cs100vbg)

P2 Benchmarking Analysis

Test:

Our benchmarking program tests the efficiency of search algorithms in several tree data structures, including a basic Binary Search Tree (BST), a Randomized Search Tree (RST), and the C standard Set. Our measure of efficiency was the number of strictly less-than (<) comparisons made during a successful search. Data inserted were integer values in the range [0,N], where N is the largest power of 2 less than a specified M. This was done to ensure our insertion algorithm resulted in complete binary search trees, when possible. The results we show use M = 32,768 and N = 32,767. For each test i = [1,15], we inserted values $[0,N_i]$, where $N_i = 2^i - 1$. For a complete tree, this would result in a height i. Data were inserted in either a sorted, increasing order or a random order (using the C++ standard random shuffle). We then ran our search algorithms on every node in the structure and recorded the average number of comparisons done.

Results:

Our results for each data structure are plotted below, where the x-axis represents number of elements and the y-axis represents number of comparisons performed by the search algorithm. The results for the Set, Binary Search Tree, and Randomized Search tree structures are shown in green, blue, and red, respectively.

Plot A shows performance on a $\log_2$ scale when data were inserted in a random order. These findings are consistent with the each structure's expected $O(\log_2 n)$ time complexity for searching.

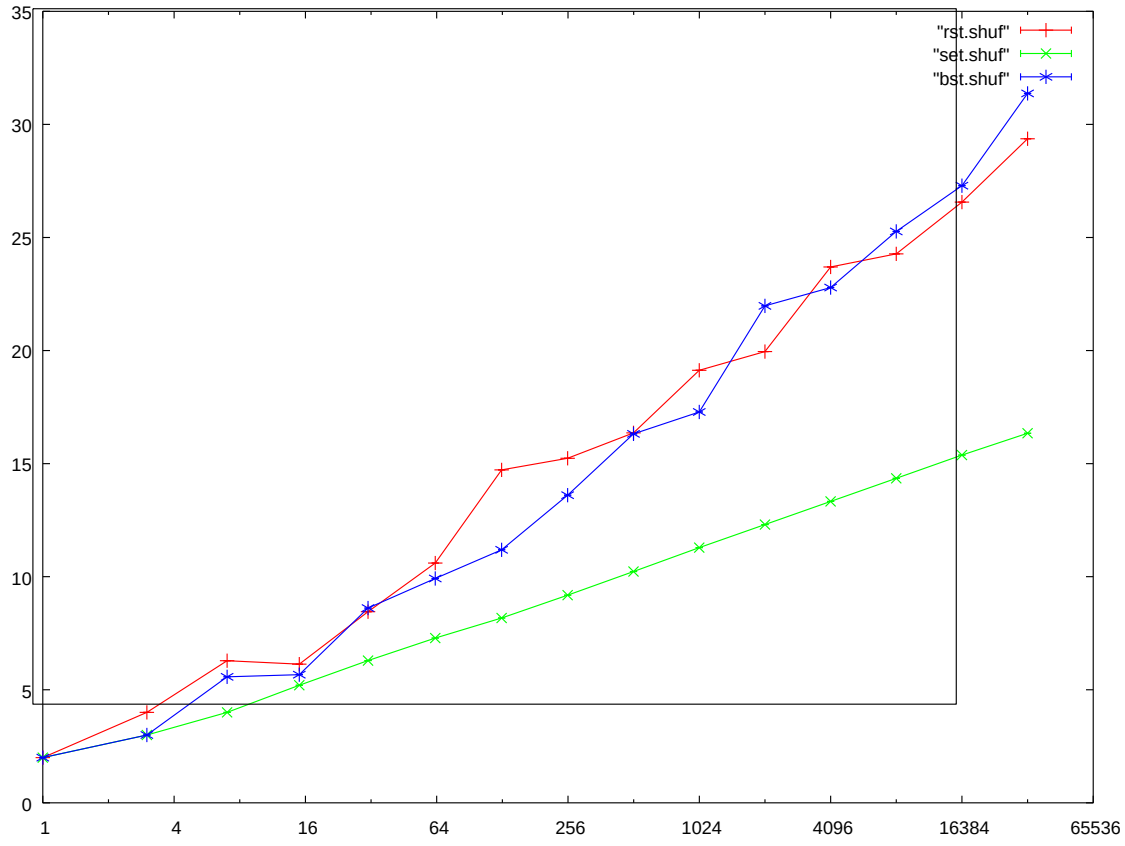
Plot B and C show the performances when data were inserted into each structure in an increasing sorted order. Plot B is on a $\log_2$ scale, and plot C is on a linear scale. While the search efficiency for the Set and RST structures are in $O(\log_2 n)$ time, it can be seen that the search efficiency for the BST is in O(n) time. This is consistent with our expectations that the worst-case time complexity for searching a BST is when the data are inserted in a sorted order.

Conclusions:

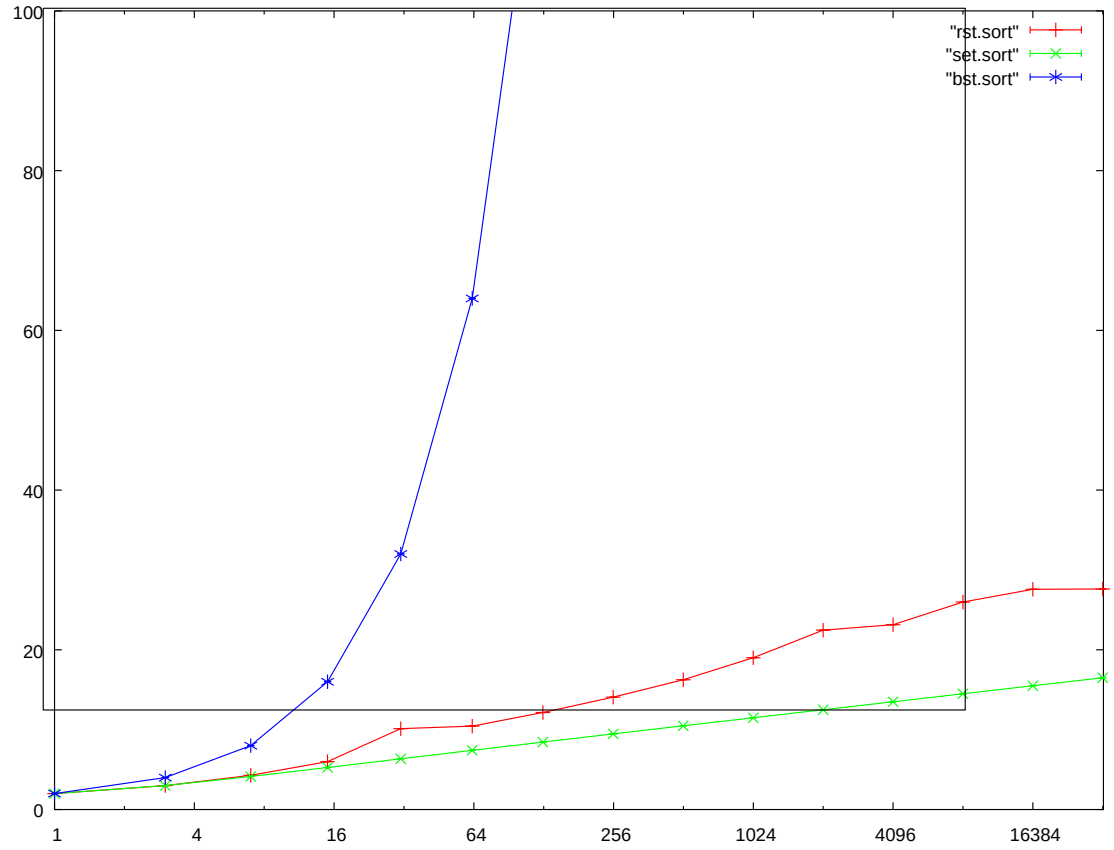
The results were consistent with our expectations that, in the average case (i.e. when data are inserted in a random order) searching all three structures can be done in $O(\log_2 n)$ time, and in the worst case (i.e. when data are inserted in a sorted order) searching the Set and RST remains in $O(\log_2 n)$ time. The BST, however, functionally becomes a linked list and searching is performed in O(n) time.

It can be seen in our results that, though all are in the same complexity class, the Set's search algorithm outperformed both the BST's and RST's. This should not serve as any indication that searching a set is more efficient than an equivalent BST or RST. The variation is more likely a result of a difference in implementation and our use of the number of strictly less-than comparisons made during a search as our sole measure of performance.

(A) Data inserted in random order ($\log_2$ scale)



(B) Data inserted in sorted order ($\log_2$ scale)



(C) Data inserted in sorted order (linear scale)