PA2 benchtree.pdf

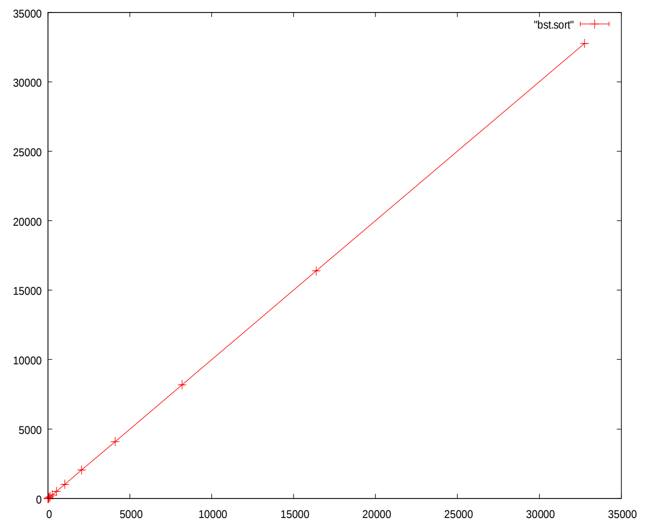

Fig.1

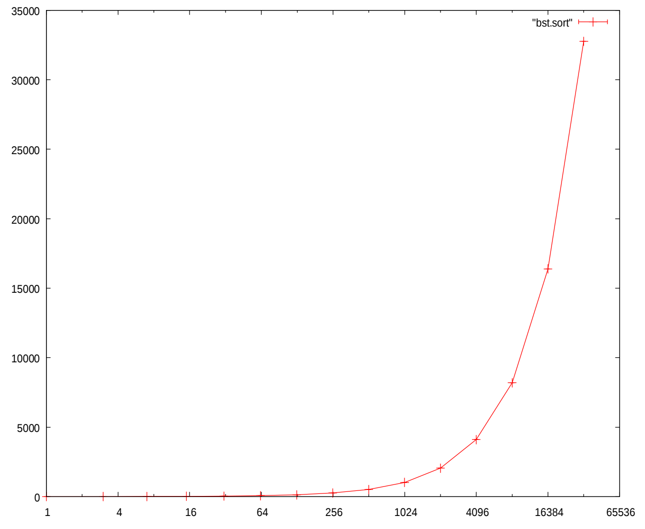

Fig.2

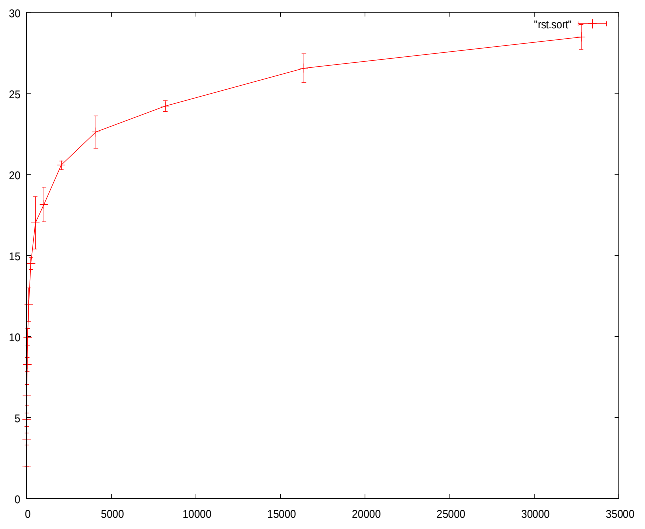

Fig.3

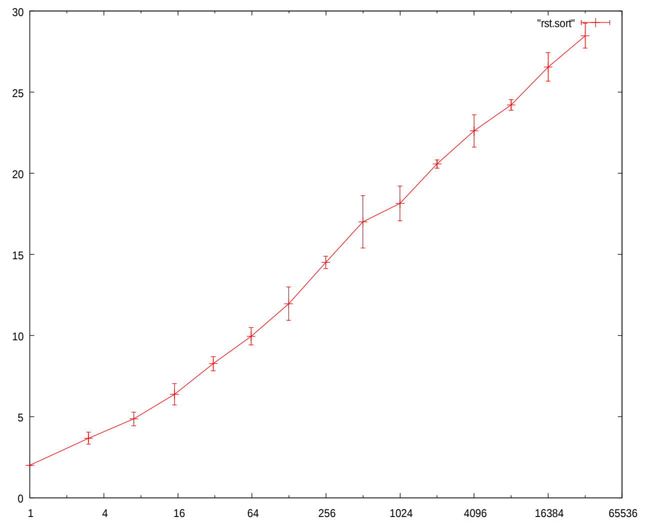

Fig.4

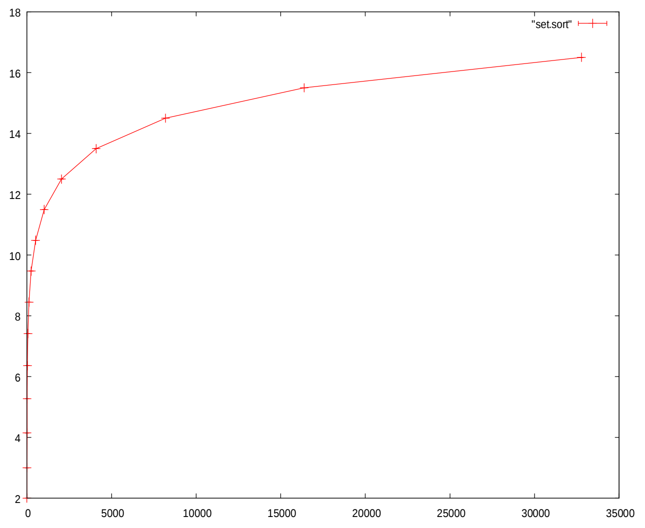

Fig.5

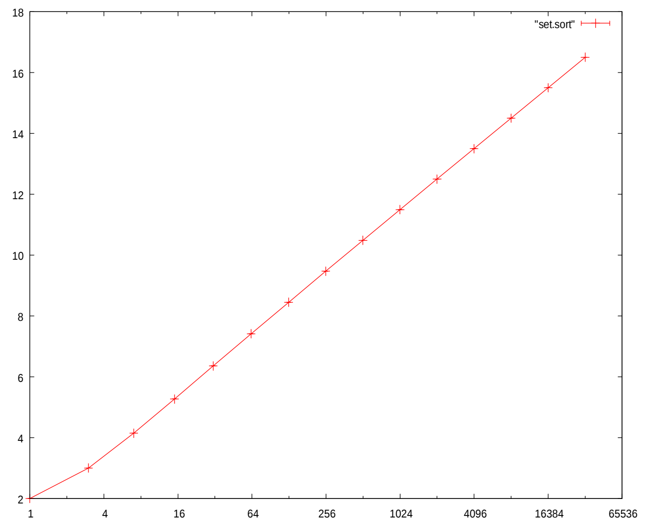

Fig.6

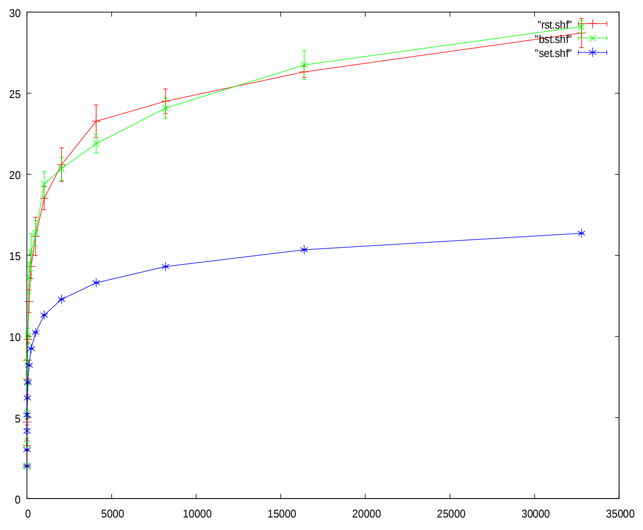

Fig.7

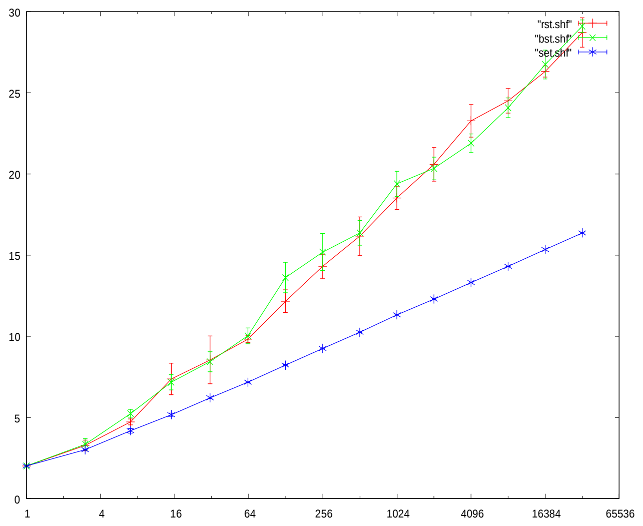

Fig.8

1.
Fig.1 & Fig.2 are for BST in sorted order in linear x scaling and log x scaling. We can see that it is the worst case O(N) for BST when we insert nodes in sorted order.
Fig.3 & Fig.4 are for RST in sorted order in linear x scaling and log x scaling. As shown in graph, the big-O of RST is about 1.386 log N(O(log N)). Its performance is better than BST, because it avoids the worst case in BST of O(N).Since we use RST to randomly assign keys to nodes, it may be unstable when N is small.
Fig.5 & Fig.6 are for SET in sorted order in linear x scaling and log x scaling. Red-black tree has the best performance among three. The time complexity is about O(log N).
Fig.7 & Fig.8 are for BST, RST and set in shuffled order in linear x scaling and log x scaling. We can find that when shuffled, big-O of BST and RST are similar and SET is still the best.

2.
To sort by any BST, we can in-order traverse this BST and record each node's value in an array. Because it will touch each node once, the big-O is about O(N). If we define it by sorting performance given in the question, there are no comparison required.
To sort by hash table, firstly we need to sort keys of this hash table. Quick sort may applicable and will take O(N*log N) in average case. Then we find each value by key in that hash table which takes roughly constant time. Thus, the big-O of sorting by hash is O(N*log N). The sorting performance is about N*log N.
Judging by sorting performance, BST is faster than hash table in average case.

From Yiqiu Liu & Lijun Chen