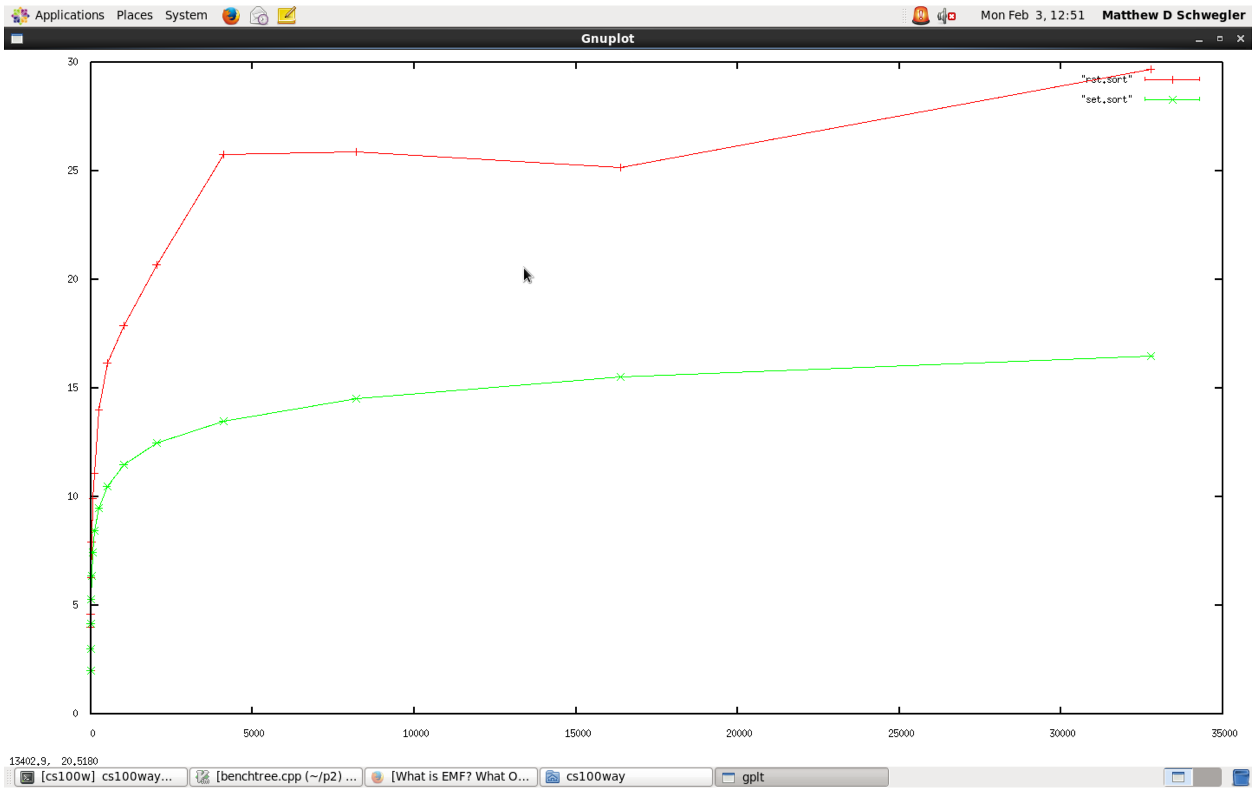
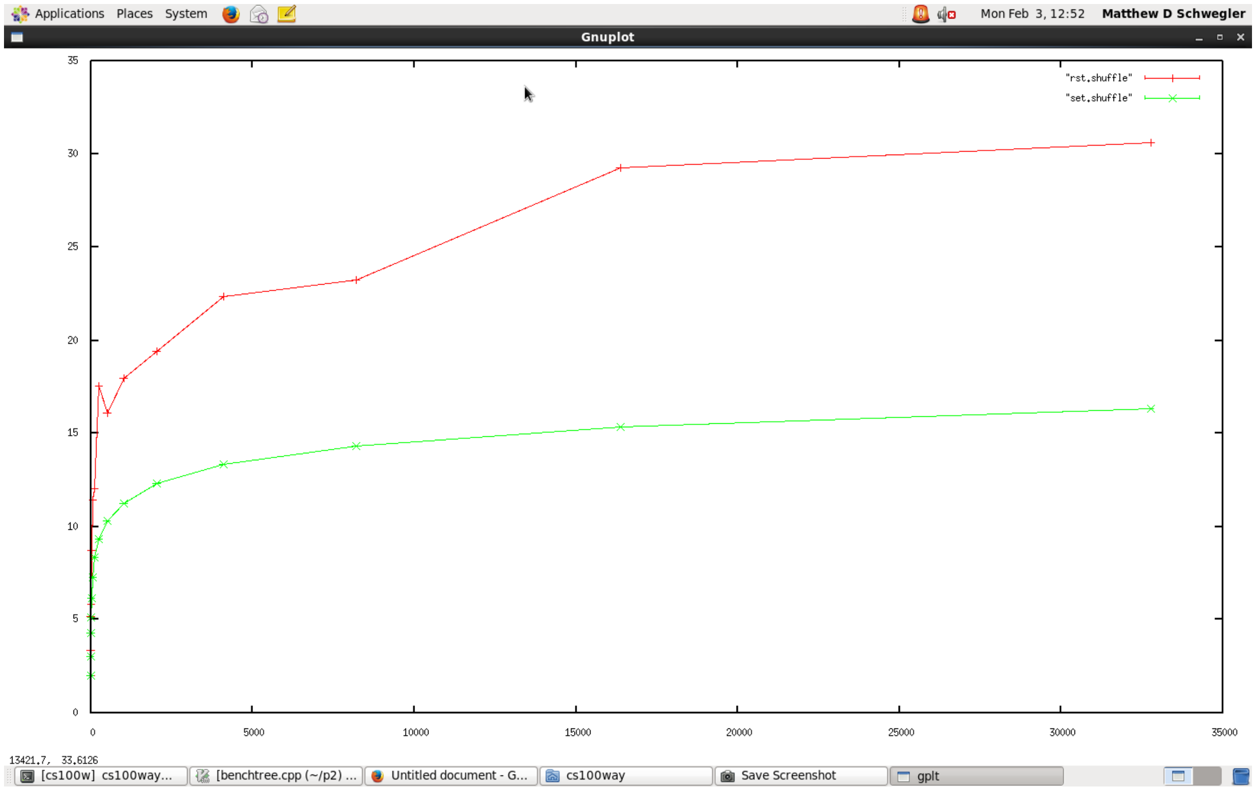
Matthew Schwegler
ID: A09293823
Assignment 2

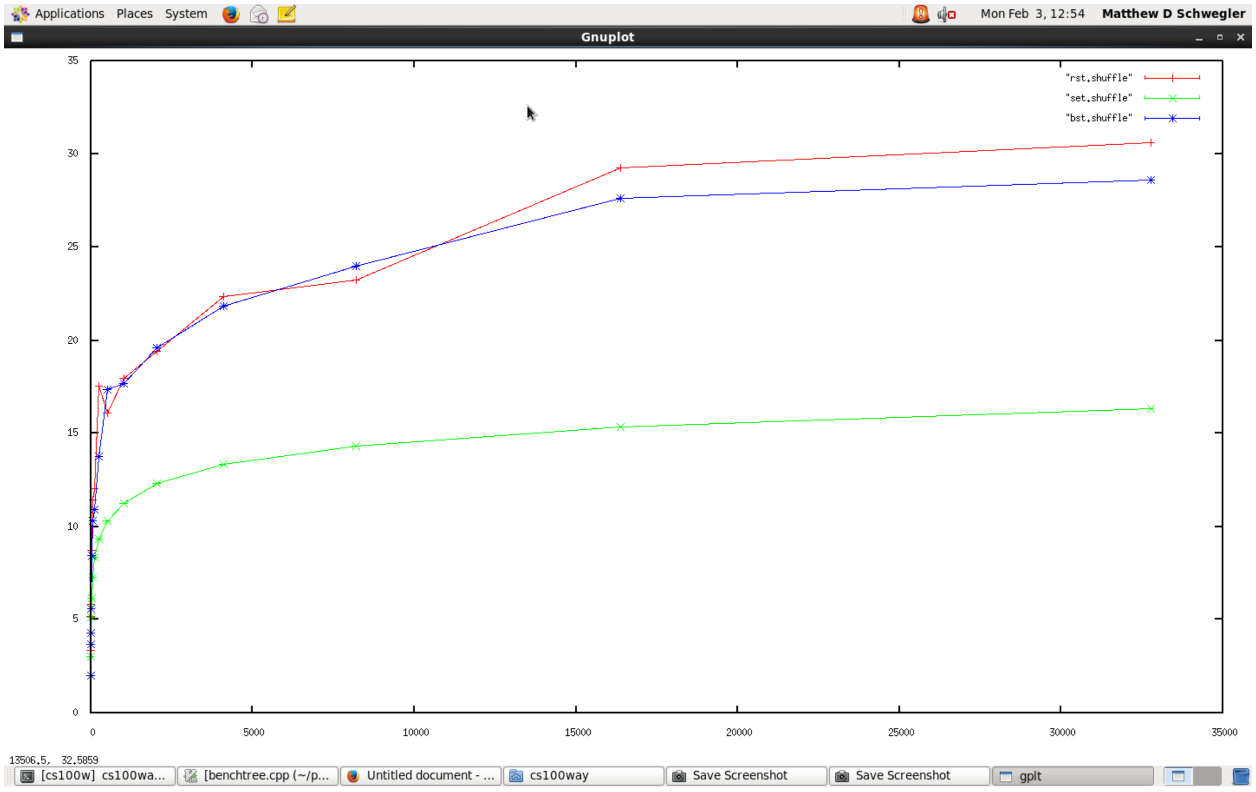
Disclaimer: I could not save the gnuplot files directly to my computer as it would give me a “disk space quota exceeded.” I had to resort to taking screen shots hence the low quality shots below. I did try to clear disk space and contact ACMS neither of which solved my problem



The above Compares rst.sorted (RED LINE) to set.sorted (GREEN LINE)



The above compares rst.shuffle(RED LINE) to set.shuffle(GREEN) LINE



The above Compares rst.shuffle(RED) set.shuffle(GREEN) bst.shuffle(BLUE)

In analyzing the the efficiency of my RST, BST, and SET it was apparent that how the C++ library implements a red black tree is superior to both RST and BST in efficiency. However RST came pretty close in the average case over both shuffle and sort. RST gave comparable result on large data sets. However as the graphs above demonstrate RST was prone to more erratic behavior shown by the sharper graph compared to sort’s smooth curve. This is expected however because the RST is based on random priorities to form the structure of the tree. This leads to different structures each time and slightly different performances. SET does not suffer from this as its rotations cause a fixed structure based on the balance of its nodes. BST.sorted is omitted in the above graphs because it is outperformed so badly it makes the curves of RST and SET hardly visible. As expected it performs just like a linked list on sorted input. Finally worth mentioning is BST on shuffled input as it compares similar to RST. This is also expected because essentially RST is attempting to ensure all input is treated as if it was randomly inserted into the data structure. These are my findings from analyzing BST, RST, and SET using GNU plot.