Kwasi Osae-Kwapong HW3 4/9/21

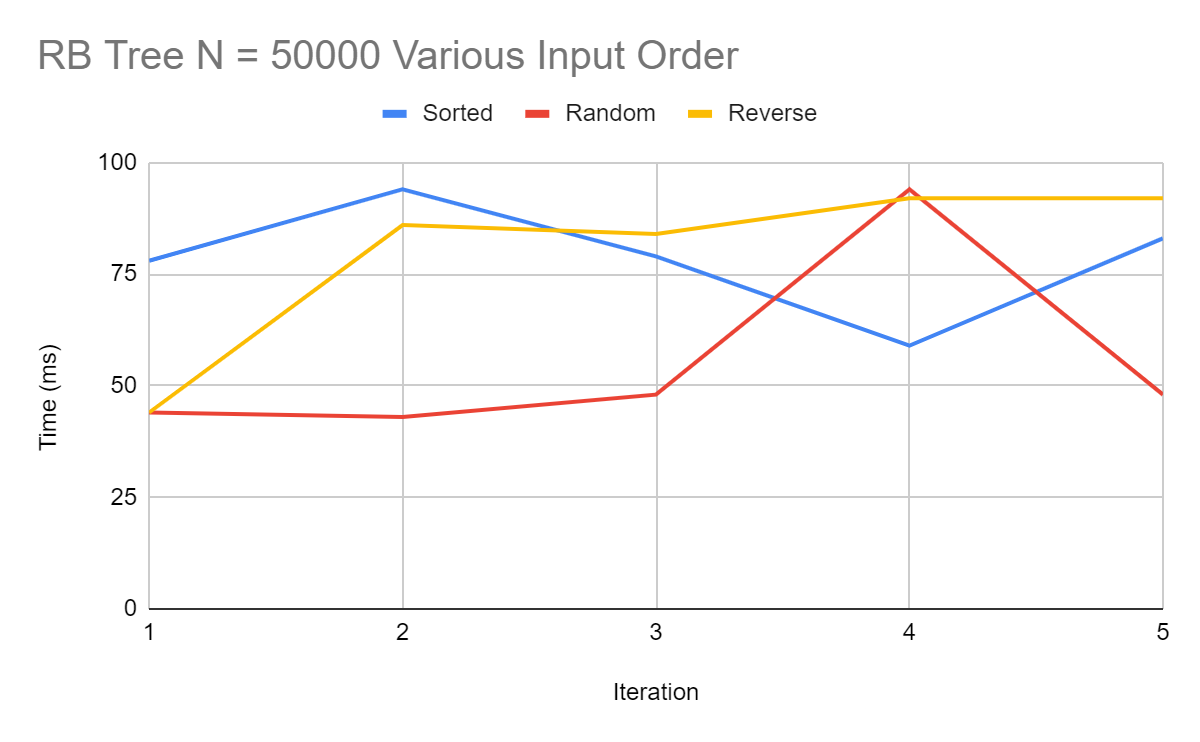
Here a binary search tree (BST) and red black tree (RBT) are implemented such that an array of integers can be inserted in either tree to produce a sorted array. Several functions have been modified to account for special cases that may violate the binary search tree or red black tree properties. Both the binary search tree and the red black tree were implemented to have a running time of O(log n) at worst case.

The major difference between the binary search tree and the red black tree is the addition of the extra bit to the red black tree of “color”. A valid red black tree maintains the five properties the tree is modified by insertion or deletion. The five properties include: that every node in the tree is either red or black, the root of the tree is always black, every empty leaf represented by the t\_nil sentinel are black, if a given node is red then both its children must be black, and for each node all simple paths from the root to a leaf must contain the same number of black nodes.

Inserting a new node into a red black tree differs slightly from its binary search tree counterpart. Three special insertion cases must be accounted for to maintain the red black tree property. The first case potentially violates property 4, when z, the new node to be added to the tree, and its parent are both red. The next two cases occur when the new node z’s uncle, y, is black and z is the right child or left child in case 2 and 3, respectively. To solve case two by calling the left rotation function to turn it into case 3. A call of the right rotation and a color change is done on case 3, making z’s parent the new root of the subtree with its children both being red. This solution also preserves property five and the tree black height.

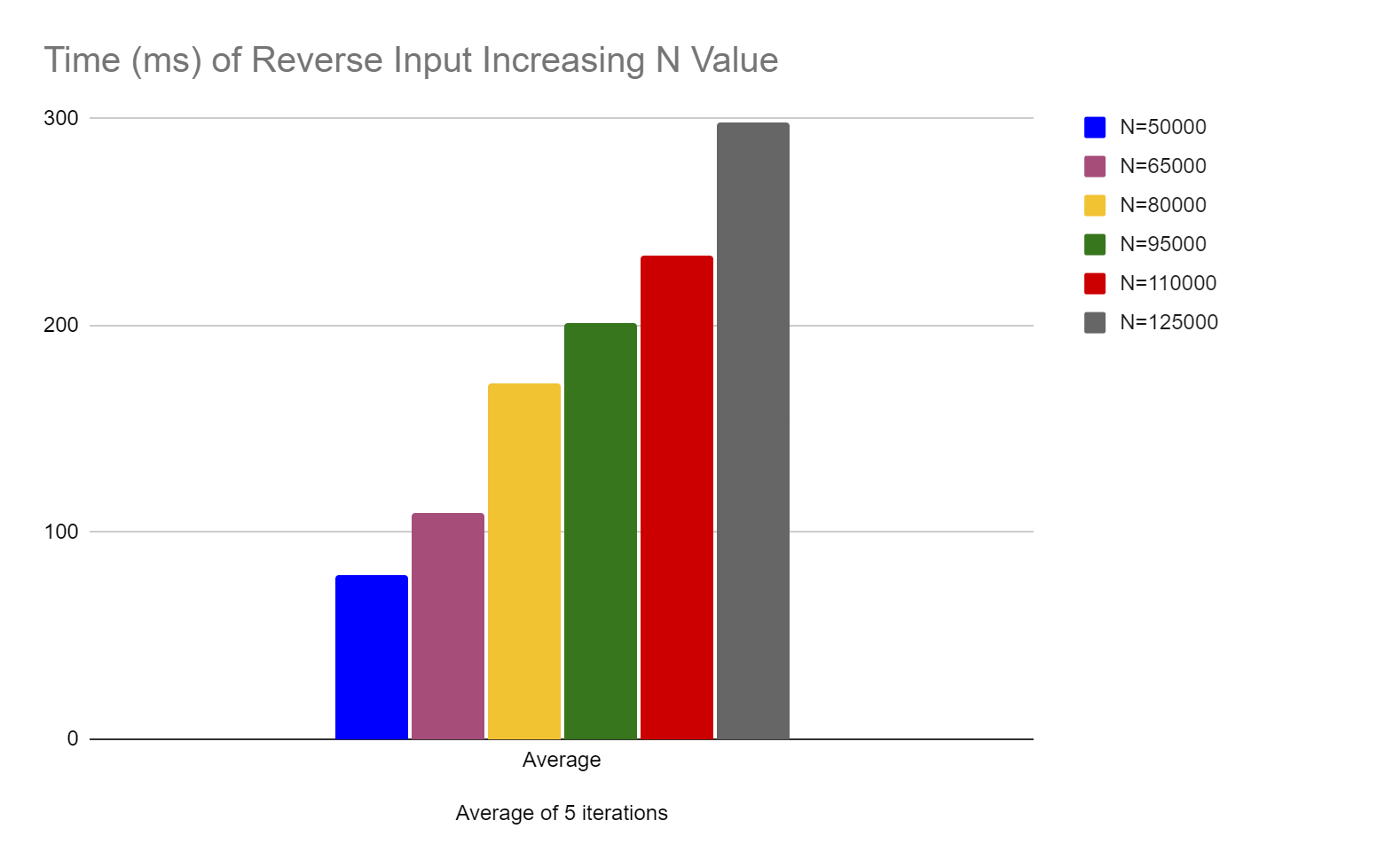
The running time of the red black tree insertion performs mostly as expected at O(log n) time with a few unusual observations. To demonstrate this performance the red black tree implementation was tested with an input size, n, ranging from 50,000 to 125,000 integers in three directions including sorted, random order or reverse order. Figure 1 shows the performance of n = 50,000 over the three input order conditions. On average, the random order input ran the fastest and reverse order ran the slowest. One would expect the sorted order input to perform the fastest here.

*Figure 1: Red Black Tree n = 50000 Various Input Order*



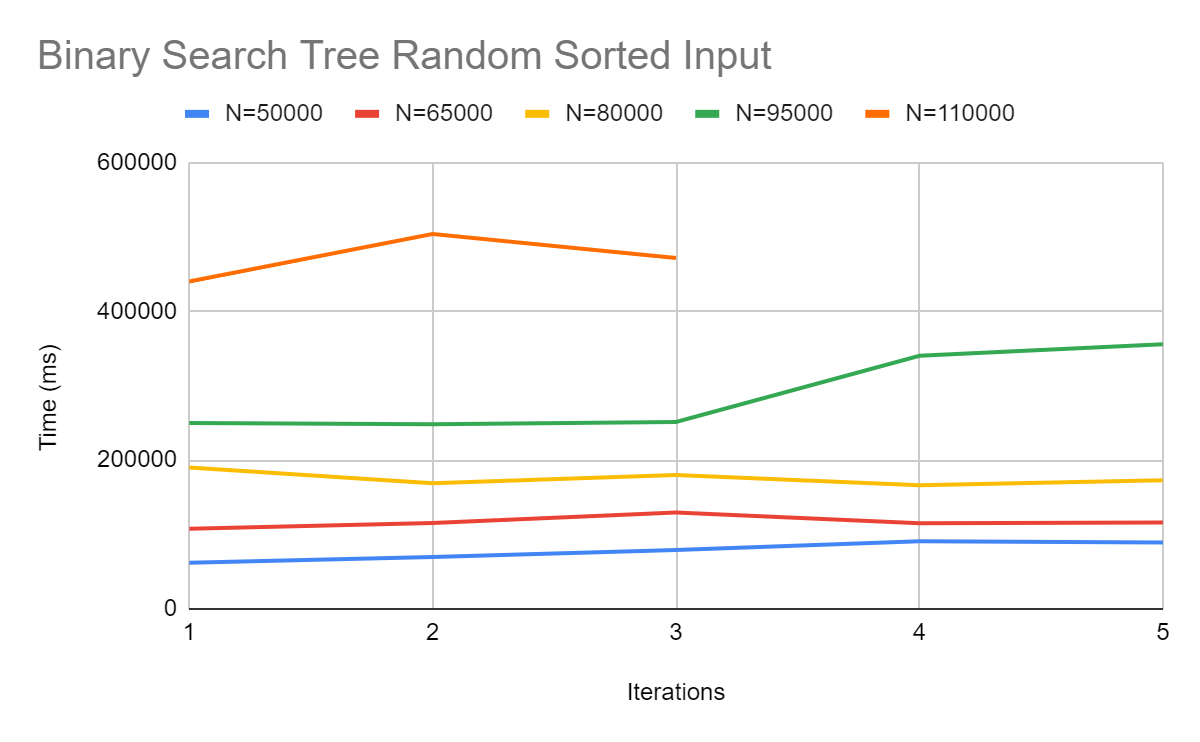
We see a more expected trend as n increases to 125,000 shown in Figure 2. Although it is expected to see an increase in running time a n grows, the magnitude at which the runtime increases is curious. The average runtime of n= 125000 is 300 ms, over three times the average run time of n = 50000 which is 80 ms. This near linear increase of running time suggests that additional running time is added during some portion of the insertion function.

*Figure 2: Average Runtime of Red Black Tree with Reverse Input*



Testing the binary search tree implementation led to compiler error and significantly larger runtimes than the red black tree. When testing both the sorted and reverse sorted inputs resulted in the following cygwin exception error “hw3 1692 cygwin\_exception::open\_stackdumpfile: Dumping stack trace to hw3.exe.stackdump”. The stackdump file generated lists the memory addresses of the functions that caused the errors. The following command points us to the lines of code in question: “addr2line.exe -f -C- e hw3.exe *memory address*”. Line 66 in bs\_tree.cpp seems to be the problem, however I could find no solution to fix the error. Surprisingly, testing the implementation with the randomly sorted input array was successful. In Figure 3 we see how the running time increases with increasing input size, however the increase in runtime is larger than expected considering that the trees should be running a O (logn) time.

*Figure 3: Binary Search Tree Random Sorted Input Runtime*



Comparing the running time of random sorted input with increasing input size shows a stark difference between the two implementations. The binary search tree running times are on the order of 1000 times greater than the red black tree implementation. Clearly, a portion of the red black tree implementation is not running as expected.