1. Which version of the tree had the fastest processing:
    - the tree from Comparison A (built from sorted list, ascending)
    - the tree from Comparison B (built from shuffled list)
    - or the tree from Comparison C (build from sorted list, descending)?
    - Why do you think that is?

Answer:

Comparison B had the fastest processing time. In my case with 40,000 records, Comparison A took 1599 milliseconds, Comparison B took 51 milliseconds, Comparison C took 21710 milliseconds to build the tree front the records.

The reason why Comparison B took a shorter time to build is that because input data set was shuffled, it enabled a more balanced tree than Comparison A and C. In case of Number of nodes in left subtree: 6374 and the Number of nodes in right subtree: 33625. This enables Comparision B to achieve closer to O(log n). On the other hand, Comparison A and C, had left subtree 3 and right subtree had 39996 nodes, which made it almost like linear insertion time.

2. How can you explain the difference in building and processing time between the Comparison A tree (built from sorted list, ascending) and the Comparison C tree (built from sorted list, descending)?
    - The tree built from the sorted list ascending took less time to build and process. Why?
    - Hint: Try drawing a small tree to see what is going on. Perhaps a tree built from 1, 1, 2, 2, 3, 4, 4, 5 and from 5, 4, 4, 3, 2, 2, 1.

Answer:

It is because our tree allows duplicate value to be added and if it is duplicate, inserted in the left subtree.

If the records are in descending order, Comparison C, and duplicates added to the left subtree, it makes linear insertion time O(n). When the highest number becomes the root. Everything else must be left subtree, like a linked list.

In the case of ascending order, Comparison A, root becomes the smallest number. The difference between comparison C is that if the record is duplicate, it creates left subtrees and other values to be in the right subtrees. This makes the difference in

processing time. Comparison A took at least a little bit of advantage of the binary tree structure.

3. For Comparison B, which processing was faster- the tree (built from the shuffled list) or the actual shuffled list? How would you describe the big-o of what was going on in the processing with these two structures?

In the case of tree, tree is sorted now. Therefore, retrieval big-o is O(log n). Thanks to the binary search tree, we can reach this time. Even with duplicates, it will be the same O(log n) because we are returning the first matching result.

In the case shuffled list, it is linear search, O(n).

4. What is the main characteristic of a binary search tree affects its efficiency?

The main characteristics is it is sorted, every value on the right subtree is bigger than the current node, and every value on the left subtree is smaller (and equal in case of duplicate is allowed,) than the current node. Because of this characteristic, we can disregard half of the subtree on every step in the process which results in O(log n).