Algorithms and Collections  
Coursework Part 2

In our program, we created implemented use of out binary search tree class, the java.util.TreeSet class and the java.util.HashSet class. We created items which were chosen randomly to fill each set. We then tested to see the average number of comparisons required to find a specific item This was done a number of times with successful searches and then again with unsuccessful searches. These were repeated numerous times to gain a range of results, between 10 and 50000. These tests were performed on the BST, the TreeSet and the HashSet. The averages were found for each and these are the results:

Comparisons Required to Search for an Item – BST(Successful)

With the Binary Search tree class, we also calculated the height of the tree and its number of leaves. As the size increased, the average height of the tree grew also, but at a much lower rate than the size. The leaves however grew at a great speed, increasing at a similar rate to the size. When the size was 5000, the number of leaves was over 1600. As with the size being 50000, the leaves grew ten times greater. The average comparisons grew steadily by a small amount between sizes, despite there being many more items.

Comparisons Required to Search for an Item – BST(Unsuccessful)

Compared to the test with successful searches, both the average number of leaves and average height were very similar, due to the trees being produce in the same way. There were actually slightly less comparisons on average with the unsuccessful searches compared to the successful ones.

Comparisons Required to Search for an Item – TreeSet(Successful)

The average height of the TreeSet grew at a steady rate and remained the same for each test with a particular size. This is down to the way in which a TreeSet order its items. With the TreeSet, the comparisons were consistently quicker than the binary search tree at every size, but not by a great deal.

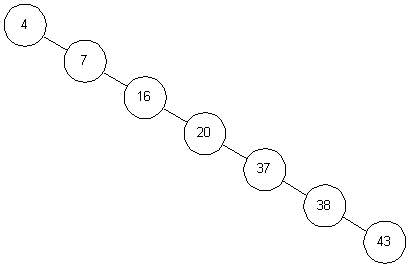
Comparisons Required to Search for an Item – TreeSet(Unsuccessful)

Like what had been seen with the binary search tree class, the height remained the same as the successful searches. The comparison results between the successful and unsuccessful searches show that unsuccessful searches took slightly longer. This wasn’t the case in the final search results (they were very similar) but this may have been down to the batch of results.

Comparisons Required to Search for an Item – HashSet

Unlike the other two instances that were tested, the HashSet class works differently. As shown by the results, there is no variation. The results either return 1 for successful or 0 if unsuccessful.

Comparisons Required to Search for an Item – BST added In Order

When the items were added in ascending order and the BST was tested, there were notable changes.  
The of the tree was equivalent to the size-1. The number of leaves were equal to 1 as the tree was unbalanced, similar to this tree:  


*Source:* <https://www.cs.auckland.ac.nz/~jmor159/PLDS210/niemann/s_bin.htm>In a successful search, the number of comparisons was equal to the value that was being searched for as it passed through the tree in ascending order. In an unsuccessful search, the number of comparisons was the size of the tree, as it passed through every node.

Overall Thoughts

Overall, binary search trees are useful when the searches fail, but TreeSet is rather more efficient when the search is successful. HashSets are much more efficient due to their simple searches, but they are not represented in a tree however, with no height nor leaves.

Serialization

A *serialVersionUID* had to be added and a **static final long** was used with the deserialization so it knows the receiver is getting a compatible object with serialization from the same place as the object the sender serialized. WriteObject and readObject methods were added so the program could serialize then deserialize. The Item class was made Serializable alsoand had its own *serialVersionUID* added. The code to test the serialization and deserialization was added to the *main* in the test class. This was added in a try-catch to account for exceptions.

Critical Appraisal

Acknowledgments

Slides accompanying Collin’s book covering TreeSets  
Watt and Brown’s lecture on Hashing  
https://www.cs.auckland.ac.nz/~jmor159/PLDS210/niemann/s\_bin.htm