Homework 6 - Binary Search Tree

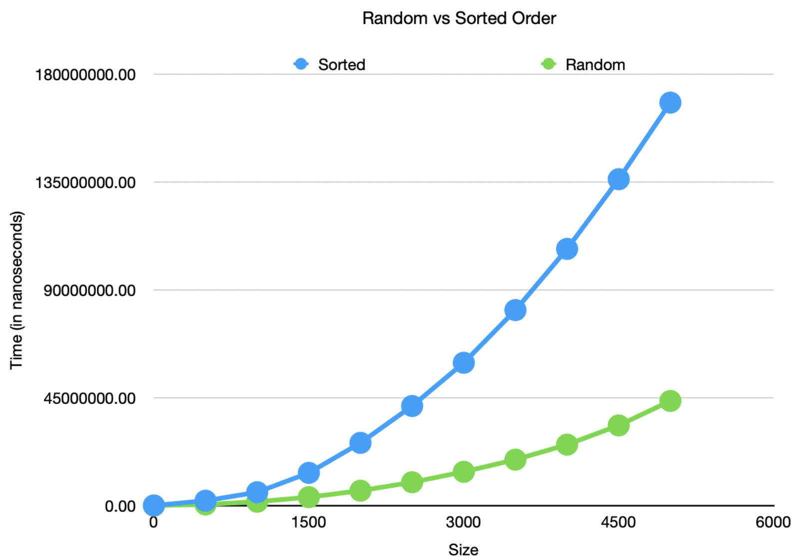
Lindsay Haslam

1. If items are inserted into sorted order, the tree becomes a linked list. This would result in the worst-case scenario for add, contains and remove methods. The Big O notation for that scenario would be O(N). If items are inserted in a random order, it is much more probable that the tree is balanced, with a better time complexity of O(logN).

2. & 3. Build and plot the experiment of a random and sorted set.

BSTRandomvsSortedOrder

1	517.91	259.57
501	1990663.7	427900.37
1001	5542642.93	1569165.86
1501	1.360663914E+07	3478533.36
2001	2.613999959E+07	6181231.71
2501	4.152783794E+07	9707491.22
3001	5.957882118E+07	1.405862133E+07
3501	8.156111331E+07	1.914390042E+07
4001	1.0710725159E+08	2.540390793E+07
4501	1.3619881334E+08	3.34264483E+07
5001	1.6817571547E+08	4.363806114E+07



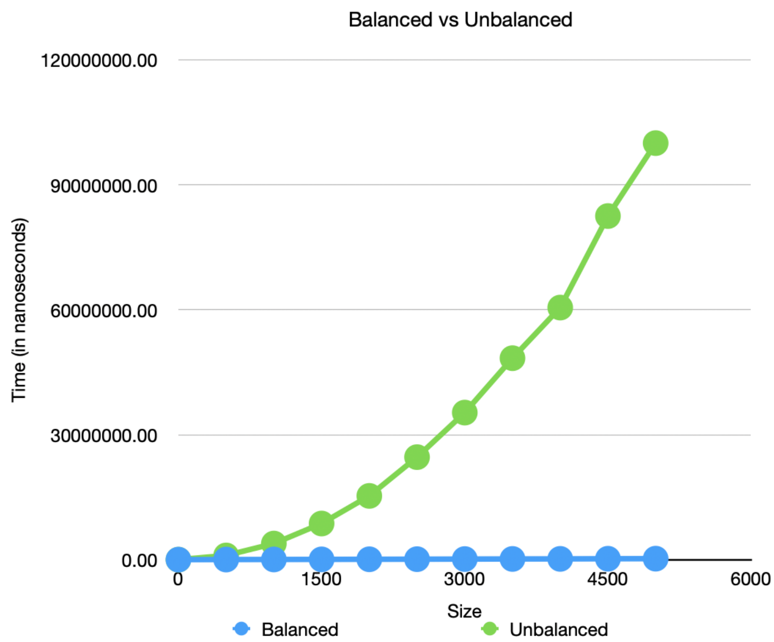
In this experiment, we tested the contains() method and calculated the average times for a sorted and random set. The program iterates over different sizes of sets (powers of 2, starting from beginning to end with an increment of 500). It creates an ArrayList and populates it with integers from 0 to “size -1”, then adds all elements from ArrayList using addAll(). We measured the time it takes to perform contains() for both sorted and random, and repeated that process multiple times using ITER_COUNT. And then we calculated the average of those times, and printed them onto a file.

It’s pretty clear that a random set performs better than the sorted set. It seems that the sorted set closely resembles O(N) while the random set has a much lower pitch, resembling somewhat of O(log N). As you can see from the chart, the sorted set is less efficient than the random set.

4. Build and plot the experiment of a balanced and unbalanced BST.

BSTTree

1	393.33	470.75
501	25278.34	1009997.04
1001	47367.09	3863667.86
1501	66199.98	8676411.23
2001	91539.13	1.531077252E+07
2501	115664.14	2.461884798E+07
3001	143025.09	3.531331704E+07
3501	170240.83	4.838096413E+07
4001	183796.28	6.053033629E+07
4501	219652.13	8.25116946E+07
5001	236538.73	1.0002051745E+08



This is an experiment to test the performance of the contains() method in a balanced and unbalanced tree set. Similar to the previous experiment, we set up the range of the sets using “beginning”, “end” and “increment” variables. It creates an ArrayList and populates it with integers from 0 to “size -1”, then adds all elements from ArrayList using addAll(). We measured the time it takes to perform contains() for both sorted and random, and repeated that process multiple times using ITER_COUNT. After that, we calculated the average and wrote those results into a file.
Similar to the chart above, the balanced set performed significantly better than the unbalanced set. Unbalanced resembles that of O(N), while balanced resembles O(1), which is dramatically different in time complexity.

5. I think a BST *could* be one way to represent a dictionary, but I feel like its worst-case scenario could be detrimental. As we saw in these experiments, the performance of a BST is greatly impacted by how balanced it is. If something is inserted into a large dictionary, the process to get that dictionary balanced would likely take a lot of time. As long as it is well balanced, a BST could be an efficient way to represent a dictionary, but if it isn’t, it could be dramatically inefficient.

6. Inserting words in alphabetical order can lead to an unbalanced tree, or a linked list, which is very inefficient. Searching in an unbalanced BST can approach linear time complexity O(n), as compared to O(log n) that a balanced BST can be. One way you could fix the problem is by using red-black trees that maintain a sense of balance, even if they aren’t balanced perfectly. This could help prevent an accidental creation of a linked list.