A6 Analysis Document
Samantha Pope

1. Explain how the order that items are inserted into a BST affects the construction of the tree, and how this construction affects the running time of subsequent calls to the add, contains, and remove methods.

This order that the items are inserted into the BST is critical because it determines if the tree is balanced or unbalanced. Trees that are balanced are wide and have lower height than unbalanced trees. If a tree is balanced the add, contains, and remove methods have runtimes of O(log(N)). If it is unbalanced, this is the worst case scenario in these methods. This leads to a runtime of O(N). This is because each of these methods must go through more steps, as it must go through many layers of the tree (the height is larger) to find and access the data in the tree.

2. Design and conduct an experiment to illustrate the effect of building an N-item BST by inserting the N items in sorted order versus inserting the N items in a random order. Carefully describe your experiment, so that anyone reading this document could replicate your results.

I created an array with integers in ascending order and then loaded that array into my binarySearchTree (BST). I then tracked the time it took for the contains method to run on every integer up to the array size (blue line).I then tracked the time it took in the sorted BST to find the root (yellow line).

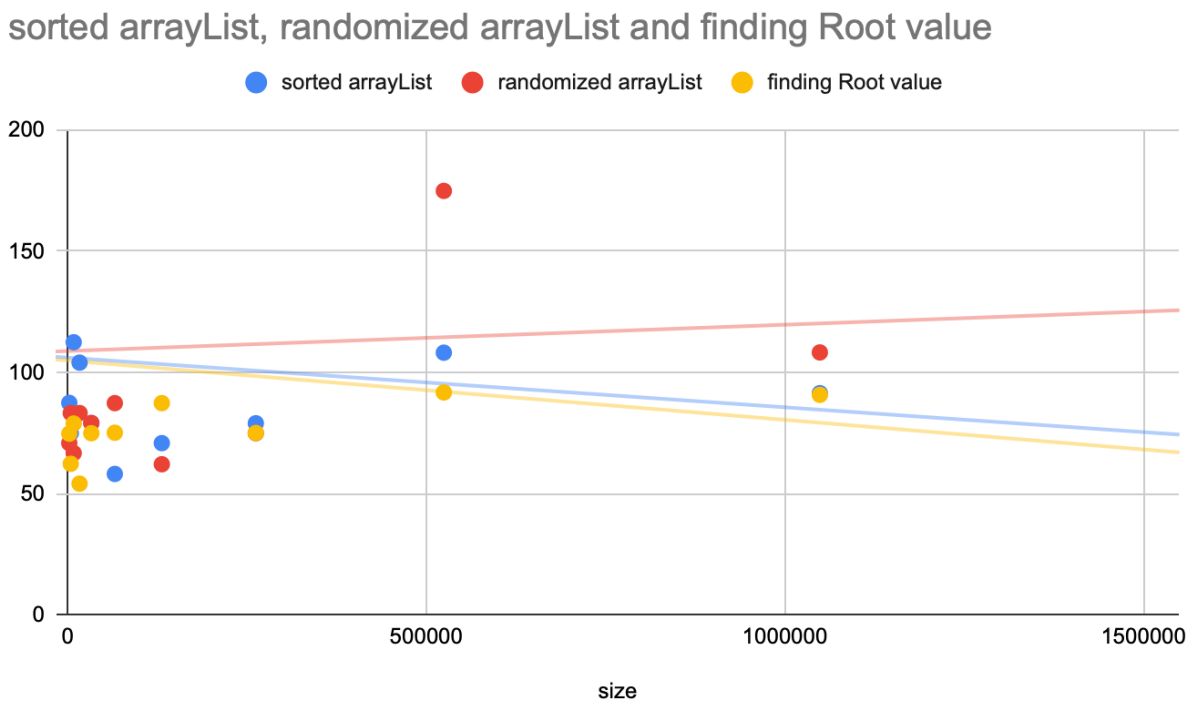
I then created an array with random integers and then loaded that array into my binarySearchTree (BST). I then tracked the time it took for the contains method to run on every integer up to the array size (red line).

I performed each of those experiments three times with array sizes from 2^10 to 2^20. I used the System.nanotime() function to track the runtime in nanoseconds. For each array size, I then calculated the average time by finding the total time of calling the method divided by the count of iterations through the loop.

3. Plot the results of your experiment. Since the organization of your plot(s) is not specified here, the labels and titles of your plots(s), as well as your interpretation of the plots, are critical.

This plot shows the time it took to run contains on the sorted versus randomized BST. I found that the randomized performed not as well as the sorted BST. This difference is greater at larger sizes of the sets. This makes sense, as the larger the tree is, the larger the height will be. This means that the unorganized/randomized BST could have a large amount of levels that the contains method has to search through to find if the variable is present or not.

I then plotted this experiment against finding the root of the sorted array, as shown in yellow. This method was much faster than the other two methods. It performed most similar to the sorted BST, but still performed faster especially at higher values.



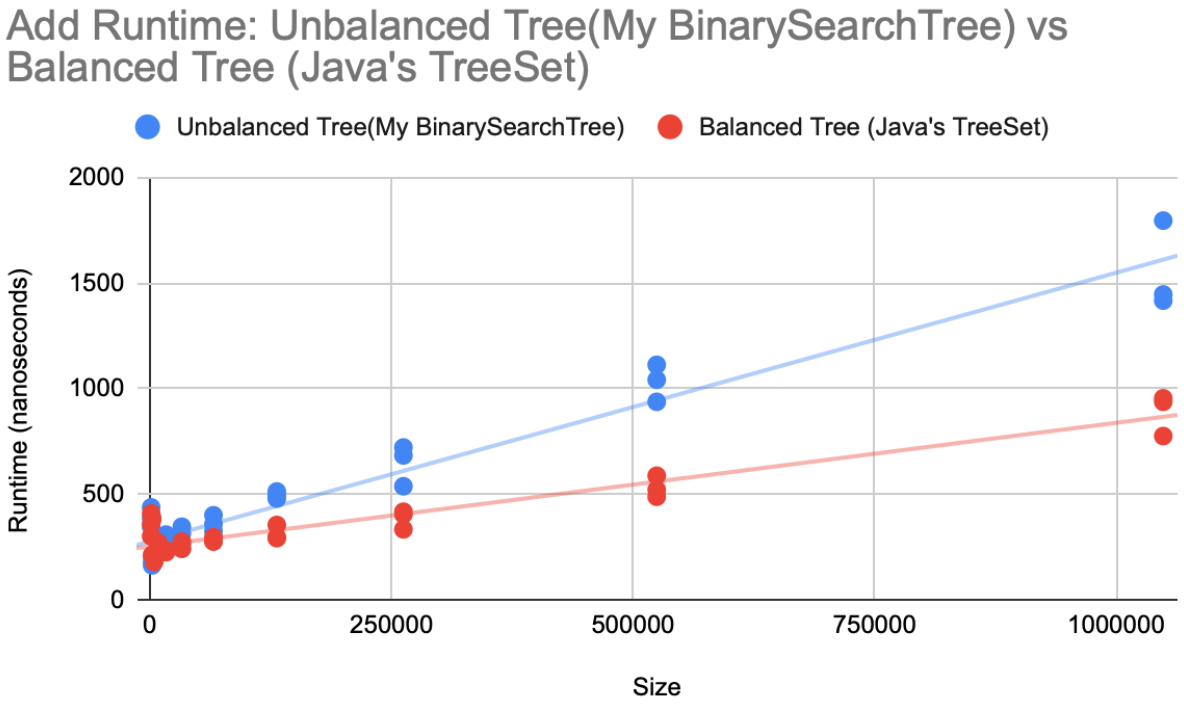
4. Design and conduct an experiment to illustrate the differing performance in a BST with a balance requirement and a BST that is allowed to be unbalanced. Use Java's `TreeSet` as an example of the former and your `BinarySearchTree` as an example of the latter. Carefully describe your experiment, so that anyone reading this document could replicate your results. Plot the results of your experiment. Since the organization of your plot(s) is not specified here, the labels and titles of your plots(s), as well as your interpretation of the plots, are critical.

For all of the following experiments I used my binarySearchTree BST as an example of an unbalanced tree and the built in treeSet as an example of a balanced tree. I created an array with random integers and then loaded that array into my binarySearchTree (BST) and the Java built in TreeSet (TS). I did this to ensure that the array was random, but the same for both sets.

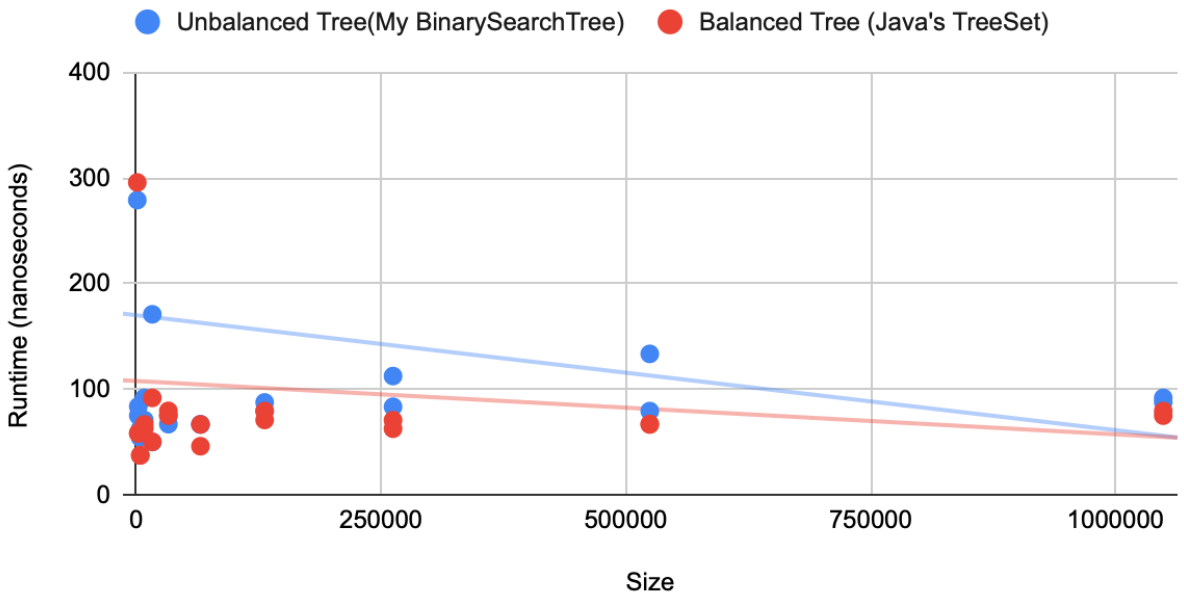
I put both the BST and the TS in the same for loop and timed how long it took to add each element of the array. I conducted this experiment 3 times with array sizes from 2^10 to 2^20 (first graph). I found that an unbalanced tree took a much greater time to add to the tree than the balanced tree. This is what I expected, as unbalanced trees have a greater height and negatively affect runtime for the add method.

To test contains, I put all the same, randomized array into the two set types. I then made a loop that timed the contains method for both of the functions that iterated through the array and called contains(array.get(i)). The results for this are shown in the second graph. I found that, again, the contains method was faster when using a balanced tree set. I again tested this three times for array sizes from 2^10 to 2^20.

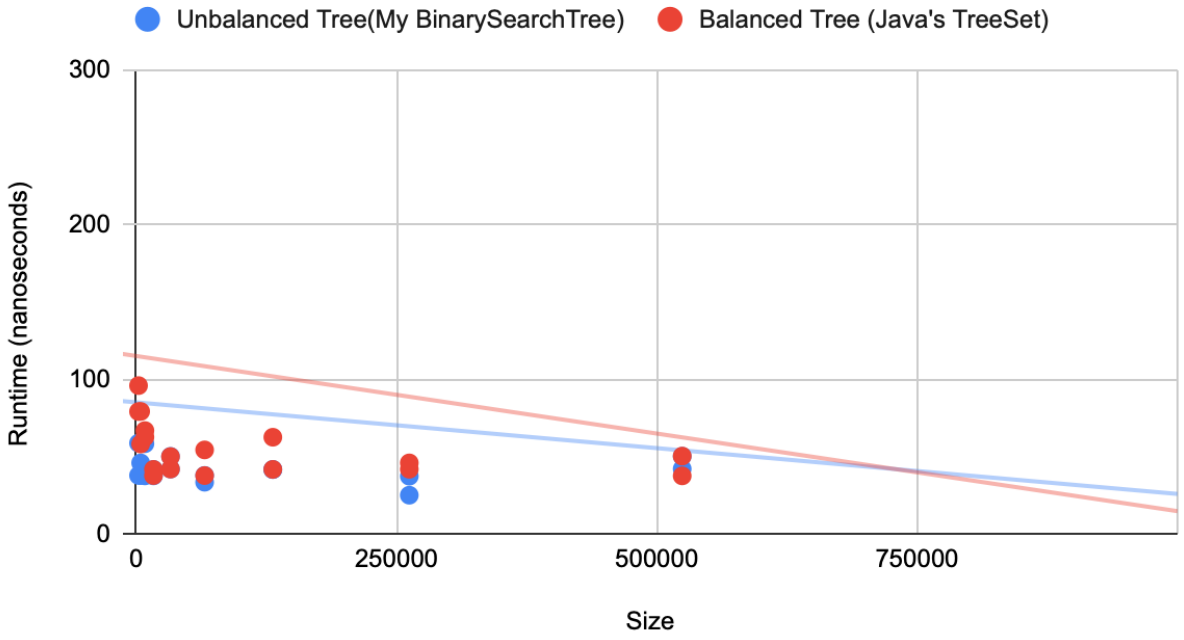
I then tested contains on an integer I knew was not in the set. I did this by bounding the random generator to be from 0 to 100000. I then asked if the sets contained -1. I plotted these results in the third graph below. I found, again, that the balanced tree performed this more quickly than the unbalanced tree. I wanted to look at a node that was not in the tree to see how long it would take the tree to go through all levels of the height and thought that this would show the difference more clearly (see third graph). I ran this experiment three times for array sizes of 2^10 to 2^20. I found that there was a size where the balanced tree started to perform not as well as the unbalanced tree. I am not sure why this is the case.



Contains Runtime: Unbalanced Tree(My BinarySearchTree) vs Balanced Tree (Java's TreeSet)



Contains Call on Element not in Tree



5. Discuss whether a BST is a good data structure for representing a dictionary. If you think that it is, explain why. If you think that it is not, discuss other data structure(s) that you think would be better. (Keep in mind that for a typical dictionary, insertions and deletions of words are infrequent compared to word searches.)

I do not think BST would be a good data structure for a dictionary. That would require a data set that could take an input (a word) and return a definition. I think that a BST could be used to implement it, but I don't think it would be effective. I am assuming that we would store the words in alphabetical order. I think the tree could be forced to be unorganized based on how the language is organized (ex: 10.6% of words in English start with s while only 0.16% of words start with x). The dictionary needs to be in order, but it is most critical that the definition is found. This is why I think a hash map would be more effective and is designed to better replicate a dictionary.

Source for words example:
https://funbutlearn.com/2012/06/which-english-letter-has-maximum-words.html

6. Many dictionaries are in alphabetical order. What problem will it create for a dictionary BST if it is constructed by inserting words in alphabetical order? What can you do to fix the problem?

This could be an issue because if it is loaded in alphabetical order, there will only be children nodes on one side of each parent. This will create a degenerate tree that only has one side and is very long. This destroys the main reason that BSTs are useful (quick search time), because it becomes one long line or list of data. At that point, we could just use a linked list or another data structure that has all of the data in one line. To fix this, we could rebalance the tree after things are added by changing the root. We could also randomize insertions from the dictionary, rather than loading the words in direct order.