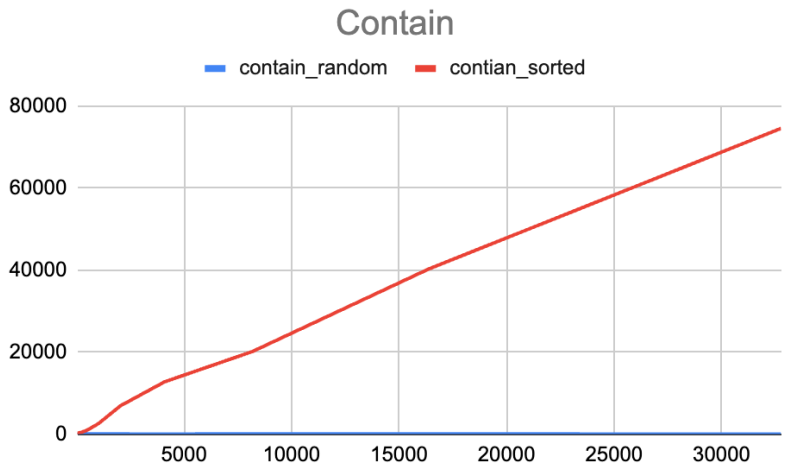
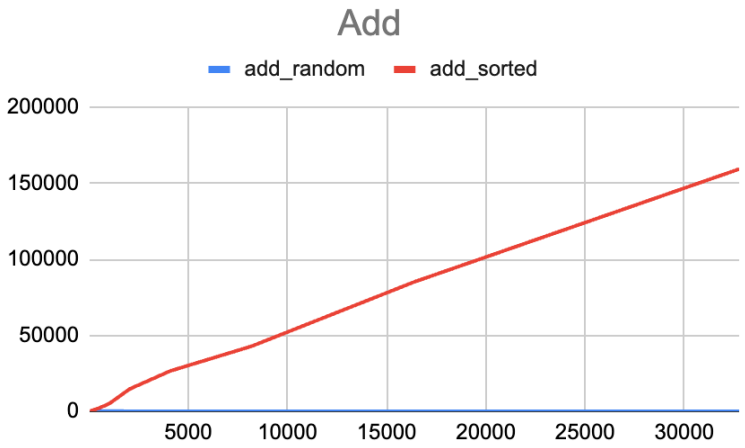
Analysis Document of Assignment 05

Tailang (terry) CAO. u1480633

1. comparison of BST with sorted and unsorted data





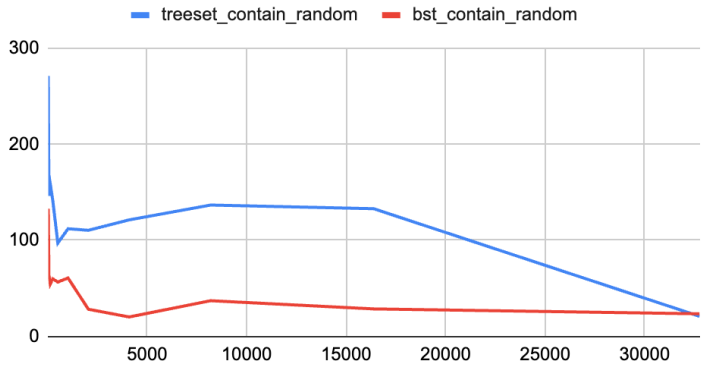
Above are the three graphs illustrating the runtime of contain, add and remove in two BST trees constructed with sorted and unsorted data.

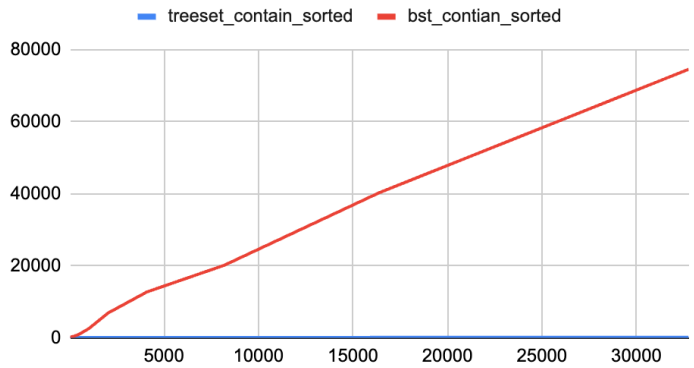
When designing the experiment, I created an array list of integers in ascending order and shuffle them into random order to create the sorted and unsorted data set. I then test the runtime of add, contain, remove of single element by operating those method with a random element, and take the average of 1000 iterations as the result. After each adding process, I remove the added element to ensure the size of the tree remains the same. Similarly, I add the removed item back to the tree if anything has been removed.

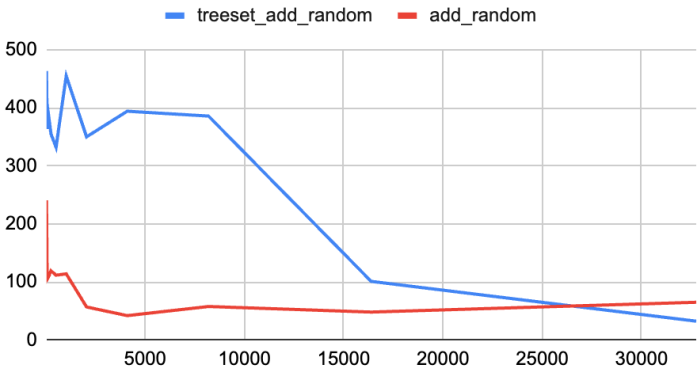
The result shows that in all three scenarios, the data of the BSP tree constructed with random data has a better performance. This is because when building a tree with sorted data, the tree would be highly unbalanced and the height of the tree would be the same of the size of the dataset. This would result in a runtime of time complexity of O(N).

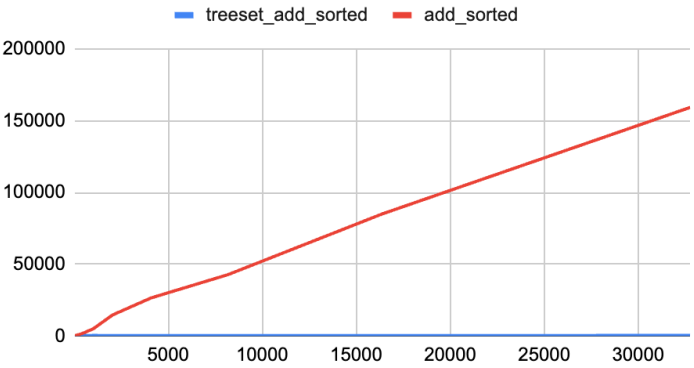
For BST trees constructed with unsorted data, the tree is relatively more balanced, with the height of logN, result9ing in a runtime of time complexity of O(logN).

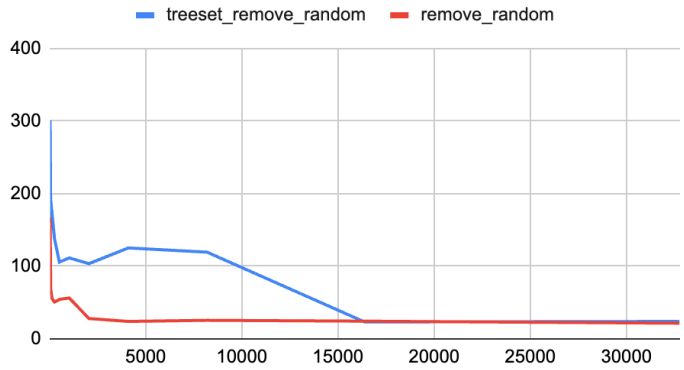
2. Comparison between Java treeSet and BST trees

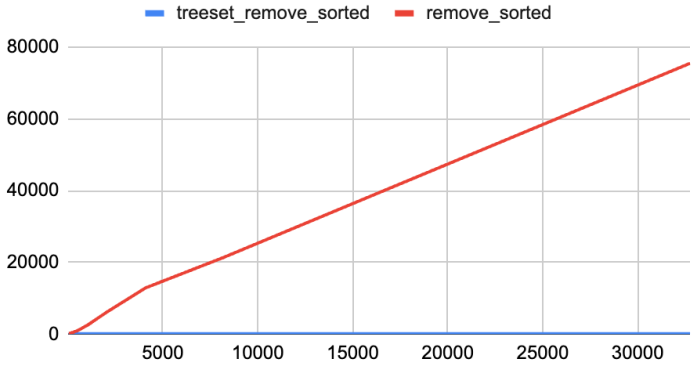












Above are the runtime illustration of the contain, add and remove method in java treeSet trees and BST trees constructed with random and sorted datasets.

It can be seen that for all trees constructed with sorted data sets, the treeSet trees has a significantly higher performance comparing with the BST trees. For trees constructed with random data sets, the runtime performance doesn't has much difference.

The experiment is designed in the similar way with the experiment in the last questions, where I created random and sorted datasets and test the runtime of operation of the three methods with one element and take the average over 1000 iterations of both BST trees and Java treeSet trees.

3. BST as dictionary data structure

According to the above experiment, I believe self-balanced BST trees are good to be as the data structure of a dictionary, while un-self-balanced trees are not suitable for being the data structure of a dictionary.

As dictionary is edited in the alphabet order of each words, using BST tree as the data structure of a dictionary can be seen as creating a BSP tree with sorted data set. In this case, an un-self-balanced BSP tree would result in a highly un-balanced tree with huge tree length and horrible runtime for each operation.

However, for a self-balanced BSP tree, the runtime of any operation would be O(logN), which is a reasonable and efficient time complexity and therefore can be used as the data structure of a dictionary.