Order Statistic Lab
COSC 3523 – Analysis of Algorithms
Carter Hidalgo Dylan Walton
11/2/2023

Introduction

The purpose of this lab is to play with Binary Search Trees and determine performance ranges based on tree height. This analysis will perform a slightly enhanced version of the proposed lab in an effort to expose and better explain the results of the experiment.

Scenario

The program will be given three parameters, **start**, **stop**, and **step**, which will dictate the behavior of the experiment. **start** represents the initial input size of the BST (i.e. if **start** equals 100_000 the tree will have 100_000 values entered). **stop** will represent the final input size and **step** will represent the increment size. For each **step** from **start** to **stop**, the program will **sample** a set number of BSTs and find the **min**, the **max**, and the **average** for the sample size. It will report back these three values for a given step size at which point the program will increment to the next step size and again re-sample to find a new min, max, and average.

The values of these parameters for the program are as follows.

- **start** is 100_000
- **stop** is 2_000_000 / 10_000_000
- **step** is 100_000
- **sample size** is 50 / 100

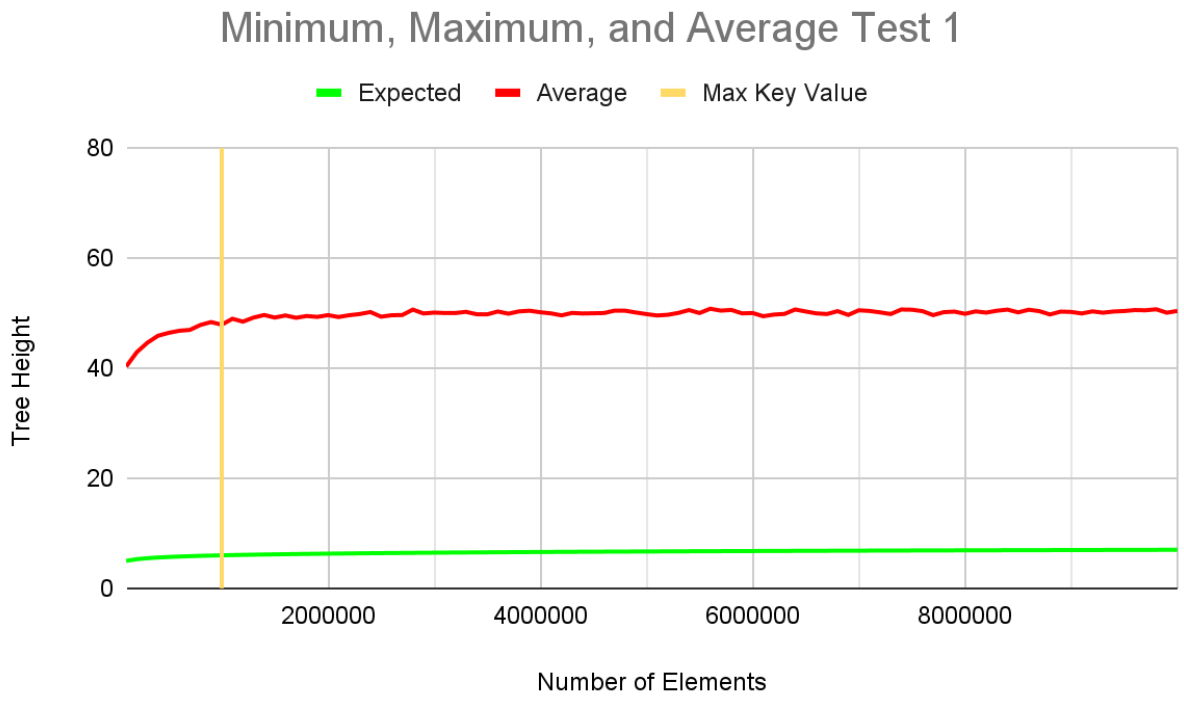
The random integer values being generated are in the range from 0 to 1_000_000. The reason for this is to attempt to demonstrate how the height of the BST is dependent on **both** the number and range of values being input into the tree. The starting values have been carefully chosen so that the number of values will initially be much smaller than the range, increase until it is equal with range, and finally be much larger than the range.

NOTE: Sample size has been decreased to 50 or 100 in the interest of turning in this report on time. Testing on small data sets revealed that increasing the sample size from 50 or 100 to 1_000 did not alter minimum and maximum tree height values substantially, so we will make the assumption that this holds for large data sets as well. Nevertheless,

a modified version of the program was run with a sample size of 1_000 and those results have also been included.

Test Run 1

The first test will generate 100 data points. Each data point is the min, the max, and the average tree height for 50 BSTs. The maximum value being entered into the tree is 1_000_000, which is represented as the vertical dashed line on the graph below. The number of values being input starts at 100_000 and increases to 10_000_000. The expected height is calculated as log(Number of Elements) and is show in green.



Rationalization

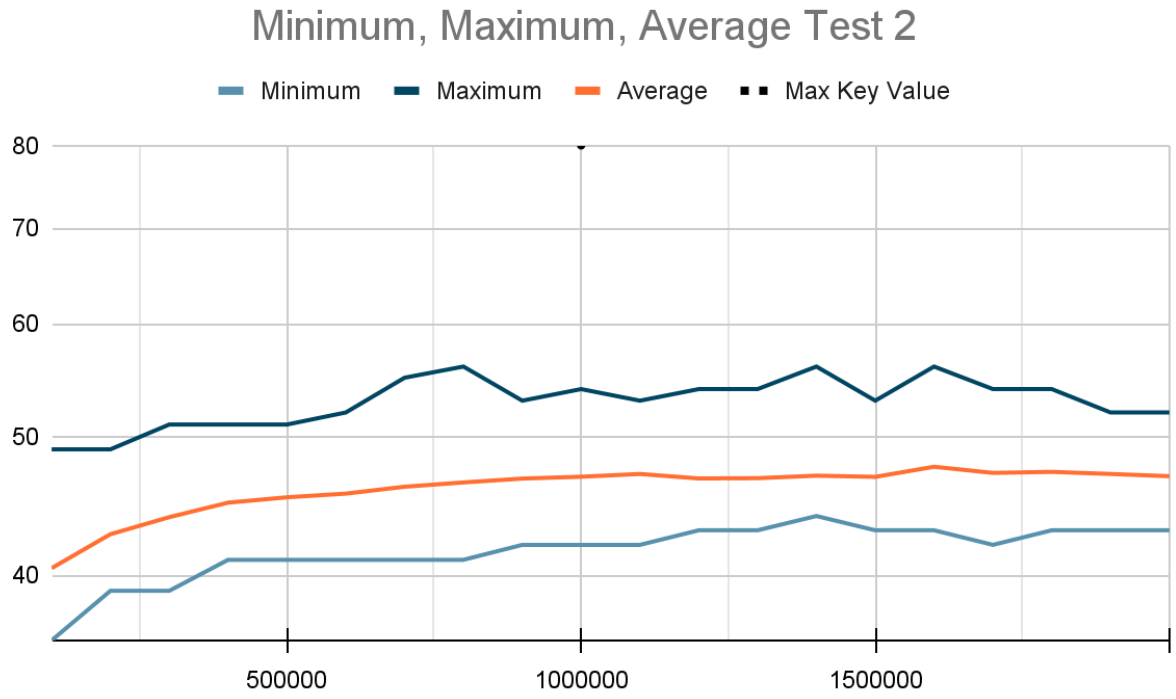
The graph above reveals a particularly interesting fact about randomly generated BSTs. The average height of the BST grows until the number of elements is roughly equal to the maximum value being entered, at which point the growth flatlines. The reason for this can be explained by understanding what happens when new extremums are entered. If a new minimum value is entered, it will traverse down the left child paths until it reaches the old minimum value, at which point it will become the left child itself and replace the old minimum value. The same is true for the maximum and right child. If the range of values is significantly larger than the number of elements (i.e. the entire integer

range versus 100_000 elements) the likelihood of a new extremum being entered with each new value is much higher. This would lead to skewed trees with much higher heights than expected. But, as the number of elements reaches and eventually surpasses the maximum value, the chance that new extremums are entered with each new data point decreases, and so the height of the tree stops growing as fast.

Also note that the average tree height is larger than the expected tree height, but relative to the worst case tree height of n, the average tree height is roughly equal to log(n) and therefore matches the expected results.

Test Run 2

Please note the graph below is on a logarithmic scale for the vertical axis to better show the changes in height. The parameters for the second test run have been modified so that the maximum input size is now 2_000_000 with a maximum random value of 500_000. 20 data points were generated.



Rationalization

The minimum, maximum, and average values once again start out increasing very slightly and after passing the max key value (middle grid line) appear to level out. One

reason for how horizontal these results are is that the BST height grows very slowly since it is tied to log(n). For relatively low values of n for which the growth of log(n) is minimal, it is difficult to show slight changes in the growth of the tree as the input size increases.

Conclusion

The results of the lab demonstrate that the relationship between the input size and the input range does influence the tree height. On average, the actual tree heights are larger than the expected tree height, but as mentioned before, relative to the worst case tree the result is roughly equal to the expected results. Some suggestions for bringing the average result down closer to the expected result is to change the data structure from a random binary search tree to a self-balancing tree such as an AVL or Red-Black tree. As it stands, the random nature of the tree gives a close approximation of best case results while leaving room for further improvement.

Original Lab Results

This section gives the results for the original proposal. Create 1_000 BSTs and find the minimum, maximum, and average height across all the trees. The maximum random value was 500_000.

Minimum Tree Height: 36
Maximum Tree Height: 49
Average Tree Height: 40.151

Thus the results when creating 1_000 trees versus only 50 or 100 are functionally the same.

GitHub

The GitHub repository for this project can be found here:
https://github.com/CarterHidalgo/COSC3523__BinarySearchTree__Java.git