

Analysis Report

Structures

For this project the two data structures I chose were,

Struct1.java

Struct 1 is a queue based data structure using an array implementation. This class was developed by me.

Struct2.java

Struct 2 is a binary search tree using a linked base implementation with nodes. The base code was developed by "algorithms.tutorialhorizon.com" and was modified to accept generic type values and count primitive operations by me. The original source code can be found at:

<http://algorithms.tutorialhorizon.com/binary-search-tree-complete-implementation/>

Runtime Hypothesis

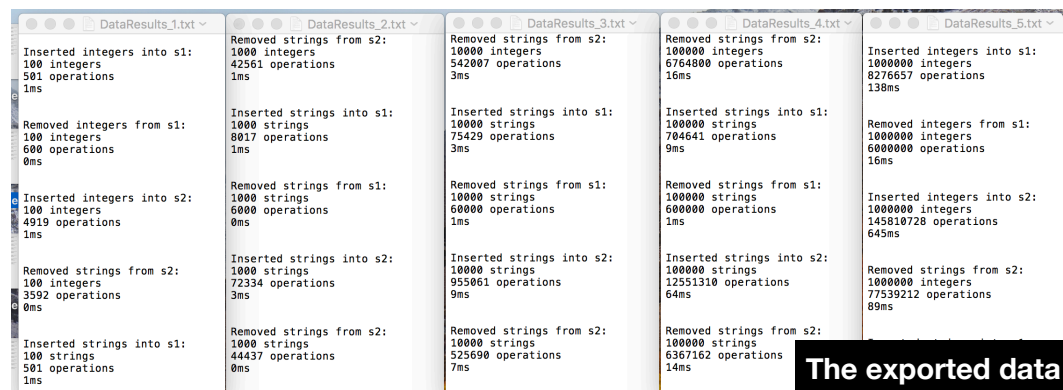
Struct1: for my queue structure I'm expecting a $O(1)$ runtime for both my "enqueue" and "dequeue" methods as I am using an array based implementation and the data being inserted or removed always has a set index for every method call.

Struct2: for my Binary Search Tree structure I'm expecting average case $O(\log n)$ and worse case $O(n)$ for my insert and remove methods. Reason being is that on every "insert" call, data is being compared and sorted and stored in linked nodes.

Overview: When it comes to efficiency, I feel that struct1 will perform faster with less operations than struct2. However, in this project I will only be testing insert and remove. In a case where specific data needs to be found, I think struct2 will outperform struct1 with less operations. This is because the data is organized when inserted, as opposed to the queue structure where data is inserted without organization.

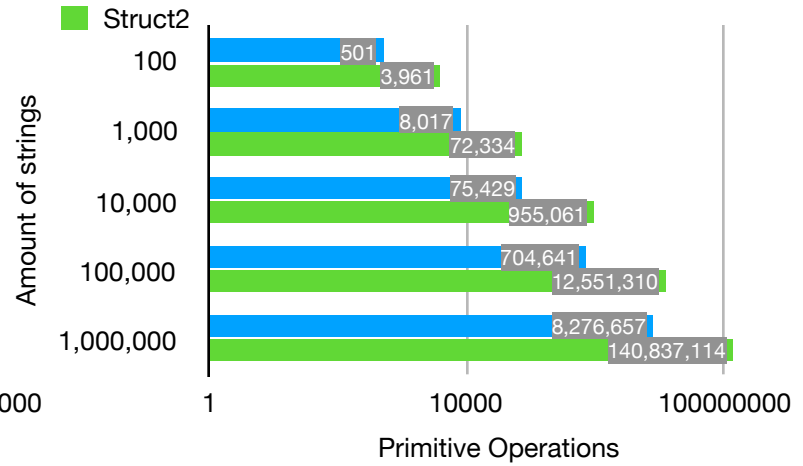
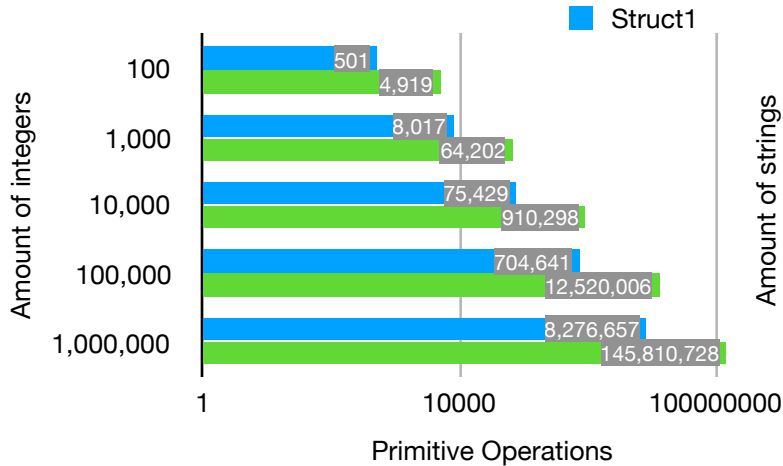
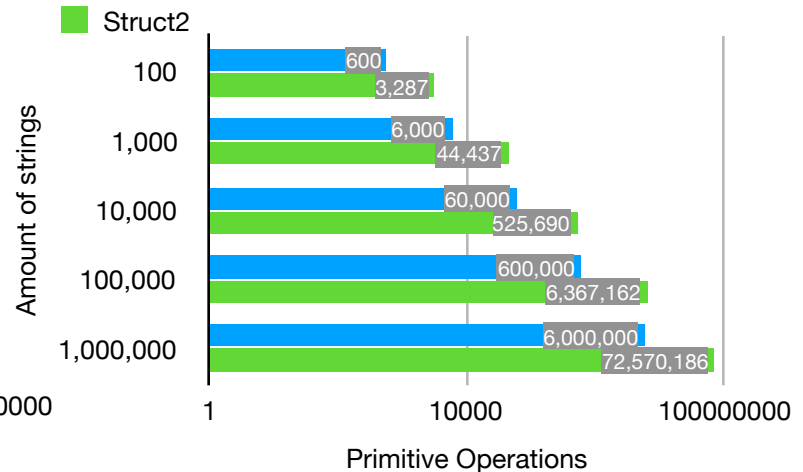
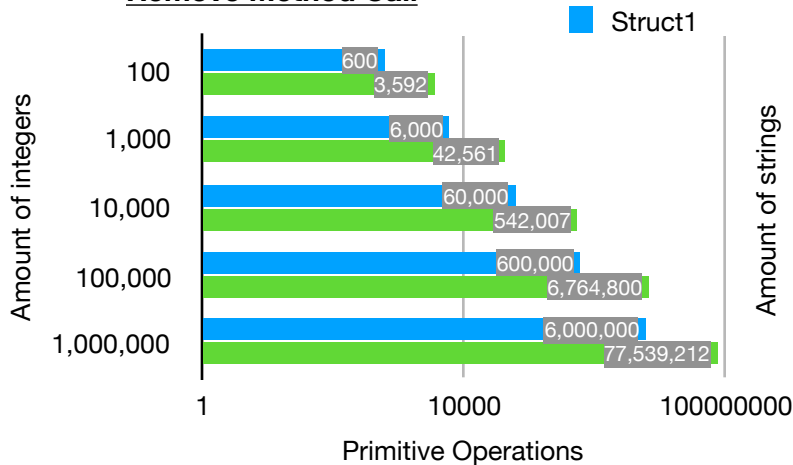
Experiments

To test my data structures I developed an app class that uses 5 sets of integers and strings to test the difference in primitive operations. My 5 sets were (100, 1000, 10,000, 100,000, and 1,000,000). I outputted the data into separate text files to keep my results organized. I measured the amount of data processed, amount of primitive operations and the runtime for each method call in milliseconds.



DataResults_1.txt	DataResults_2.txt	DataResults_3.txt	DataResults_4.txt	DataResults_5.txt
Inserted integers into s1: 100 integers 501 operations 1ms	Removed strings from s2: 1000 integers 42561 operations 1ms	Removed strings from s2: 10000 integers 542007 operations 3ms	Removed strings from s2: 100000 integers 6764000 operations 16ms	Inserted integers into s1: 1000000 integers 8276657 operations 138ms
Removed integers from s1: 100 integers 600 operations 0ms	Inserted strings into s1: 1000 strings 8017 operations 1ms	Inserted strings into s1: 10000 strings 75429 operations 3ms	Inserted strings into s1: 100000 strings 704641 operations 9ms	Removed integers from s1: 1000000 integers 6000000 operations 16ms
Inserted integers into s2: 100 integers 4919 operations 1ms	Removed strings from s1: 1000 strings 6000 operations 0ms	Removed strings from s1: 10000 strings 60000 operations 1ms	Removed strings from s1: 100000 strings 600000 operations 1ms	Inserted integers into s2: 1000000 integers 145010728 operations 645ms
Removed strings from s2: 100 integers 3592 operations 0ms	Inserted strings into s2: 1000 strings 72334 operations 3ms	Inserted strings into s2: 10000 strings 955061 operations 9ms	Inserted strings into s2: 100000 strings 12551310 operations 64ms	Removed strings from s2: 1000000 integers 77539212 operations 89ms
Inserted strings into s1: 100 strings 501 operations 1ms	Removed strings from s2: 1000 strings 44437 operations 0ms	Removed strings from s2: 10000 strings 525690 operations 7ms	Removed strings from s2: 100000 strings 6367162 operations 14ms	

The exported data

Insert Method Call**Remove Method Call**

The results more or less conform to my expectations, with struct2 exponentially increasing with the amount of data passed while struct1 had a more linear growth proportional to the amount of data. I believe changing the implementation from array to linked nodes or vice versa would have little effect, as the amount of operations is mainly effected by the structure itself. For example the fact that the BST scales and compares to find the correct spot contributes the most to the algorithmic scaling.

Challenges

The biggest challenge I faced was counting the primitive operations. To figure out what qualified as a primitive operation through all the comparisons and recursions I wanted to represent the most accurate result without overcomplicating the process.

Although not particularly difficult I found developing the CreateData class a fun challenge as I've never programmed a class that generates so much data let alone outputting it to a external file. Also I learned how to set up the main method to accept arguments from the terminal.