COMP0005 Algorithms Coursework

Ramit Bag, Yunsheng Xu, Ajay Mahendrakumaran, Robin Stewart

1 Theoretical Analysis

1.1 Data Structures and Algorithms

AVL trees were chosen as the main data structure to store the transactions, with 12 in total being used for each bank / stock name. To prevent duplicates, multiple transactions with the same trade value were stored in the same node.

AVL trees are self-balancing binary search trees with the property that the heights of any two child nodes differ by at most one. This balancing provides a logarithmic performance guarantee of look-up and insertion operations, as well as a better performance guarantee in the worst-case scenario. If instead all transactions were entered in ascending order of stock value in an unbalanced tree, the time complexity of search and insertion would be O (n).

AVL tree was selected over other competing self-balancing trees (such as red black) as it has especially fast look-up speeds due to its perfect balancing. However, red black trees do have a faster insertion speed than AVL due to their relaxed balancing property. For this project where there were many operations that required look-up, AVL trees were ideal.

To optimise the time and space complexity, the transactions with the lowest and highest trade values are stored as a property of the tree. This means that the time and space complexity of the minTransaction and maxTransaction is constant. The 12 different AVL trees were stored inside a list. This is because an index can be assigned to each of the stock names (and therefore each AVL tree) which makes the look up constant time. This means that it will not affect the time complexity of the other functions.

Lists were also used to store each transaction. The lists contain five properties for each transaction (stock name, price per stock, quantity, timestamp and total trade value).

1.2 Time and Space Complexity

n is used to denote the number of logged transactions per stock. This is because separate AVL trees are used for all 12 stocks - so the time taken to complete a transaction operation for one stock will not affect the time for another. This same principle applies to space complexity.

1.2.1 Time Complexity

logTransaction
Average: Θ (log n) for insertion + Θ (log n) for rotations $\sim$ O (log n) overall
Worst: O (log n) for insertion at bottom of tree + O (log n) for rotations $\sim$ O (log n) overall

sortedTransactions
Average & Worst: O (n) for traversing all nodes in the tree and outputting n transactions

minTransactions & maxTransactions
Average & Worst: O (1) due to the values being stored as a property of the AVL tree

floorTransactions & ceilingTransactions
Average & Worst: O (log n) for traversing from the root to the leaf

rangeTransactions
p is used to define the number of transactions within range of the input parameter in the AVL tree, and h is used to denote the height of the tree.

Average: Θ (p + h) for traversing only the nodes in range and outputting p transactions, and verifying there were no more leaves within the range
Worst: O (n) for traversing all nodes in the tree and outputting n transactions

1.2.2 Space Complexity

All functions apart from minTransactions and maxTransactions have been implemented recursively, compromising space complexity - as the call stack has more functions being pushed on it. However, this is intentional as recursive functions are easier to understand and therefore the code is far more readable.

The space complexity of each AVL tree is O (n) in average and worst case. When the space complexity is referred to below, it is purely just for the functions, which only store the tree node it traverses. Hence the stack space refers to the number of nodes required to be stored per function call.

logTransaction
Average & Worst: O (log n) for stack space + O (1) for the current transaction $\sim$ O (log n)

sortedTransactions
Average Worst: O (log n) for stack space + O (n) for the list to be outputted $\sim$ O (n)

minTransactions & maxTransactions
Average Worst: O (1) due to no traversal required and only 1 item returned each time.

floorTransactions & ceilingTransactions
Average Worst: O (log n) for stack space + O (1) for the transaction to be outputted $\sim$ O (log n)

rangeTransactions
p is used to define the number of transactions within range of the input parameter in the AVL tree

Average: Θ (log n) for stack space + Θ (p) for the transactions in range to be outputted $\sim$ Θ (p)
Worst: O (log n) for stack space + O (n) for all the transactions in the tree to be outputted $\sim$ O (n)

2 Experimental Analysis

2.1 Methodology

The run time of the program is affected not only by the system independent factors (algorithms used, input data) but also by system factors. These are due to:

- Hardware (CPU, memory, cache)
- Software (python programming language, compiler)

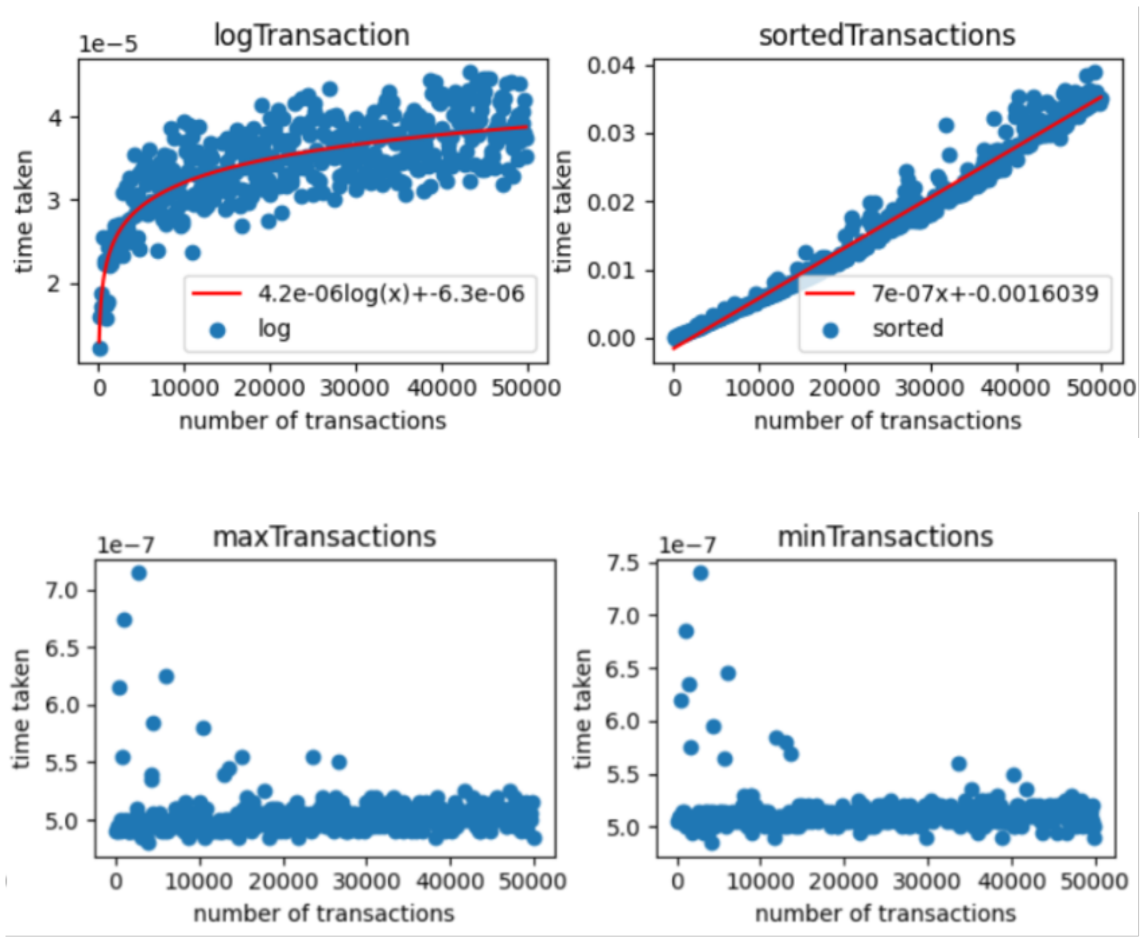
- Systems (Operating system and other running apps)

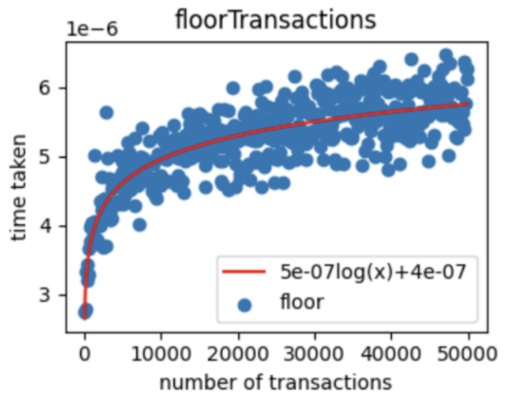
Therefore, the test was conducted with all the background apps closed on the computer to minimise the effect of the system dependent factors.

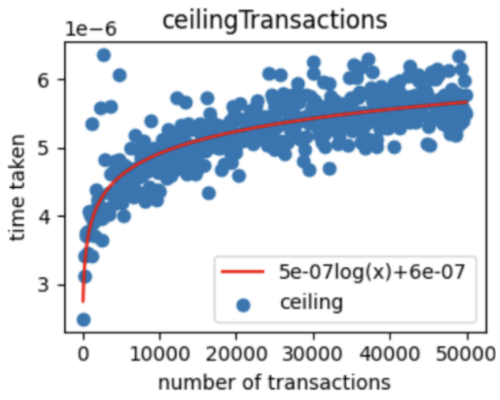
To generate the graph using the visualisation framework, 50,000 transactions were generated from the experimental framework, and used to produce seven graphs for each function. The cycle values for each of the operations were 20, except for the minTransactions and maxTransactions functions where 10 cycles were used. This was due to the consistency of these functions, allowing fewer repetitions of the experiment and therefore speeding up the completion time of the analysis.

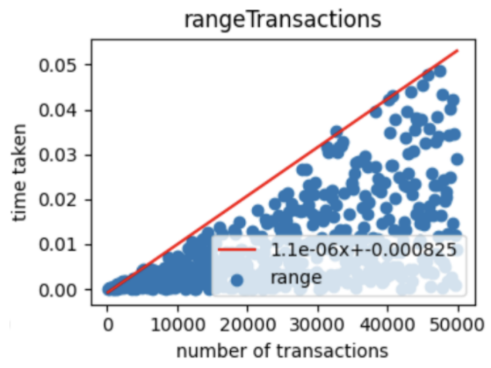
A line of best fit was calculated using the least squares regression method for the non-constant functions. This is shown by the red line plotted over the graph and this gives us the equation of the graph too. Another technique we used was removing the outliers. To achieve this, the means of the data points were calculated, and any points that had a standard deviation of over ±2 were excluded. To calculate the mean a moving average of 20 points was created for each of the operations - except for the minTransactions and maxTransactions functions where 10 were used.

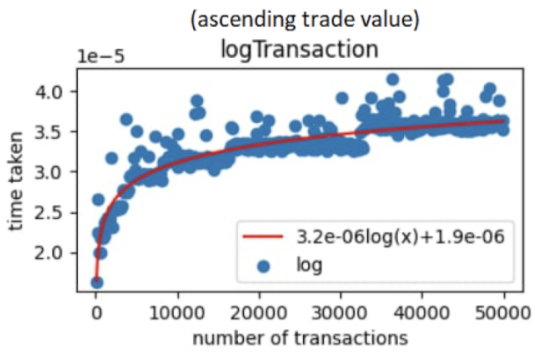
2.2 Results











2.3 Analysis of Results

- The time scales logarithmically for logTransactions, floorTransactions and ceilingTransactions, so the experimental results verify the theoretical analysis.
- In the worst case scenario for logTransactions where we enter the transactions in ascending order (to test the self balancing property of our AVL tree), the graph is not only tighter bound but also follows the logarithmic property. The stair-like shape can be attributed to the fact that when the tree height increases, the time taken in best case is capped.
- The time for minTransactions and maxTransactions is constant so the experimental results match our theoretical analysis (excluding the anomalies).
- The time for sortedTransactions is linear so the experimental results match our theoretical analysis.
- The worst case in terms of time for rangeTransactions is linear too, however, in average and best case, they tend to perform better than O (n). This is demonstrated by the triangular shape of the graph which means in best case it is O (1) or constant.

In conclusion, the graphs demonstrate that the functions operate with their expected order / scale of growth mentioned in the theoretical analysis. The graphs indeed contain anomalies, and this is expected due to the random variations of the system dependent factors.

However, excluding them and by plotting a line of best fit, we can see that the algorithm performs very well, executing the operations in microseconds. The only exceptions to this are sortedTransactions and rangeTransactions, which take roughly half a second to compute for the 50,000 transactions. This is predictable as linear functions do not scale well when compared to logarithmic and constant.

We can therefore say that for a large number of output queries the system is not ideal, yet for insertion and searching the system is optimal.