COMP0005 Group Coursework
Academic Year 2024-25

Group Number: 11

1 Overview of Experimental Framework

1.1 Framework Design/Architecture

The experimental framework is designed to evaluate and compare the insertion and search performance of three data structures (**AVL Tree, Red-Black Tree, and Scapegoat Tree**) across seven different types of synthetic data.

1.1.1 Data Generation

We use a **TestDataGenerator** to create seven types of datasets: **Random (R), Sorted Ascending (SA), Sorted Descending (SD), Almost Sorted (AS), Alternating (A), Only Duplicates (OD), and Half Duplicates (HD).**

1.1.2. Algorithm Evaluation:

For each dataset, the framework runs multiple trials for accuracy. Insertion and search times are calculated and plotted on cumulative box plots to show how they differ across different data distributions and algorithms. Line graphs track performance trends for each algorithm as dataset size increases.

1.1.3. Statistical Analysis: Kruskal-Wallis & Dunn's Test:

Since dataset distributions are not necessarily normal, we use **non-parametric statistical tests** to identify significant differences.

- **Kruskal-Wallis Test** is a rank-based test that determines if at least one dataset type behaves differently, but doesn't specify which. It uses the H statistic which determines whether one/more datasets behave differently, with higher values indicating greater differences. The test also uses Degrees of Freedom (dF = k-1) as a statistic, representing independent comparisons; higher dF makes the test stricter.

- **Dunn's Test**, which compares dataset types pairwise, was then used to identify which specific dataset/algorithm differs in performance. It uses the Z statistic which measures dataset differences; larger values mean stronger differences. Bonferroni-adjusted p-value is also used as a statistic to prevent false significance; $p<0.05$ confirms real difference.

These tests help us verify whether certain data distributions impact tree performance differently and allow for precise comparisons between the tested dataset types.

1.2 Hardware/Software Setup for Experimentation

Hardware Setup: The tests have been run on a **MacBook Air (2024)**, with *Apple M3 Chip* and *16GB RAM*. Due to the performance limitations of our machine, we selected datasets of the following sizes for our experiments: 10, 50, 100, 500, 1000, 2500, 5000, and 10,000 elements. These sizes are sufficient to thoroughly analyze and compare the performance of the data structures across a range of input sets.

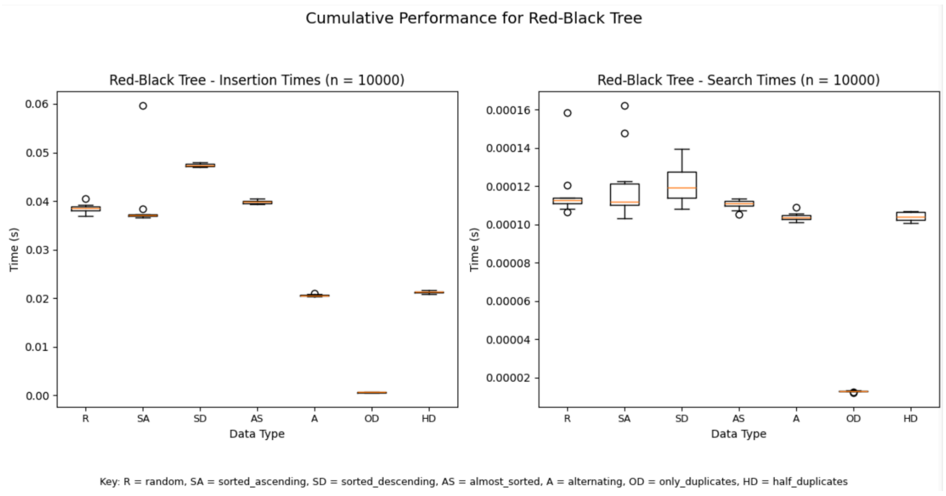
2 Performance Results

2.1 Left-Leaning Red-Black Tree

To better understand the performance of our Left-Leaning Red-Black Tree (LLRB) on the different datasets, it's important to understand the key operations that impact its efficiency:

- Tree Traversal: Move left or right until the key is found.
- Left Rotation: Balances right-leaning red links to maintain structure.
- Right Rotation and Colour Flip: Fixes consecutive left red links, propagating balance.

The Kruskal Wallis Test resulted in an H-statistic of 64.2292 for insertion, and 53.6083 for search, with a degree of freedom in 6. These results show there is a large variance in the insert results for different datasets, which we can clearly observe in the following boxplot:



We observe a significant variance in insertion times across different datasets. Inserting duplicate elements results in notably faster performance, as the tree detects existing keys and skips the insertion process, thereby avoiding any rebalancing operations. In contrast, the highest insertion time occurs with the **AS** dataset, which is expected. In this case, each new element is inserted into the rightmost path, violating the left-leaning property and triggering at least one balancing operation per insertion, such as left rotations and colour flips. As the tree grows increasingly skewed to the right before each fix, the number of rebalancing steps accumulates, resulting in higher overall insertion time.

These observations are further supported by Dunn's Test results, where z-values greater than 3 and the Bonferroni-adjusted p value lesser than 0.05 in the comparisons of SA, SD, and AS vs. OD emphasizing significant performance differences seen in the insertion graph. This suggests a notable variance in efficiency for the OD dataset. On the other hand, the relatively low z-values for search excluding OD imply that search performance remains consistent across most datasets, with the OD dataset being the primary outlier.

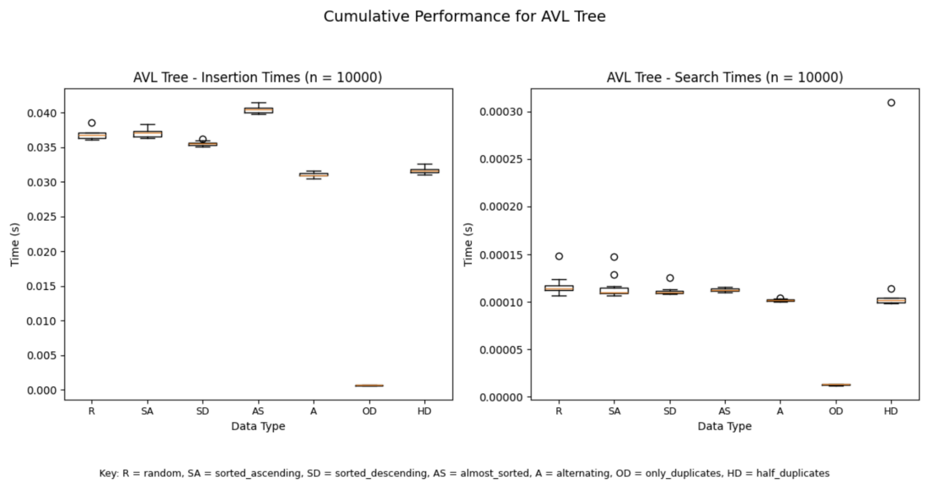
2.2 AVL Tree

To understand the performance of our AVL Tree implementation, we should first explain its key properties:

- Balance Factor: Height difference between the left and right subtrees.
- Right Rotation: Fixes imbalance from two consecutive left children.
- Left Rotation: Fixes imbalance from two consecutive right children.
- Left-Right Rotation: Left rotation followed by right rotation to rebalance.
- Right-Left Rotation: Right rotation followed by left rotation to rebalance.

For the Kruskal-Wallis Test for insertion and search times, the H-statistic is 65.8324 and 47.9125 respectively, with degrees of freedom being 6 each. A high H-statistic suggests that insertion and search times significantly differ across the data types. We now perform Dunn's test to compare each dataset.

For the Dunn's Test, we can look at the groups with a Bonferroni-adjusted p-value less than 0.05 ($p < 0.05$), ensuring that the difference is statistically significant. For insertion times, the data we obtained shows that there are significant differences in the groups R vs. A, R vs. OD, SA vs. A, SA vs. OD, SA vs. HD, SD vs. AS, SD vs. OD, AS vs. A, AS vs. OD and AS vs. HD. For search times, the data we obtained shows that there are significant differences in the groups R vs. A, R vs. OD, R vs. HD, SA vs. OD, SD vs. OD, AS vs. A and AS vs. OD. This means that insertion times and search times are significantly affected by datasets and that A, OD and HD usually show different performances than the rest of the datasets for both insertion and search times.



The boxplots for both insertion and search times further support the data obtained from our tests. In the boxplots for both insertion and search times, the median times for A, OD and HD are lower than those for other data types.

The difference in insertion times can be explained by the AVL tree's need to perform rotations upon detecting imbalances. The A, OD and HD datasets maintain better balances, reducing rotation frequency. The difference in search times is influenced by the AVL tree's balanced structure and how values are distributed within it. The A dataset leads to a more evenly distributed tree structure, which results in shorter search paths. The OD and HD data sets allow for quick lookups since the number of unique values is limited.

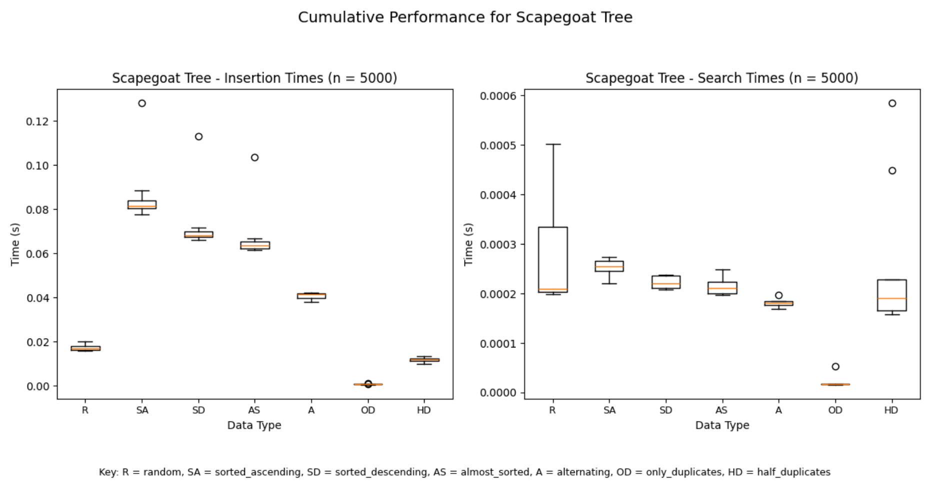
2.3 Scapegoat Tree

The Scapegoat Tree's performance is characterised by its unique balance mechanism, featuring several key operations:

- Lazy Rebalancing: Rebalances only when imbalance exceeds α-threshold; 0.6 used for the purpose of this test.
- Scapegoat Selection: Finds the nearest ancestor violating balance.
- Complete Rebuilding: Fully reconstructs the scapegoat's subtree.

Our statistical analysis using the Kruskal-Wallis Test showed significant variance across datasets with an H-statistic of 65.7721 for insertion and 52.9064 for search operations (both with 6 degrees of freedom). This high variance is expected, since the insertion times vary based on the frequency of imbalance, and the value for search is skewered by OD.

For insertion times, Dunn's Test revealed significant differences ($p < 0.05$) between multiple dataset pairs, notably R vs. SA, R vs. SD, SA vs. A, SA vs. OD, SA vs. HD, SD vs. OD, SD vs. HD, AS vs. OD, AS vs. HD, and A vs. OD. The boxplot confirms these findings, showing HD dataset with the highest median insertion time, while SA demonstrated the lowest. These



patterns emerge because HD frequently triggers the rebuilding mechanism, while SA benefits from the Scapegoat Tree's lazy rebalancing approach for sequential insertions.

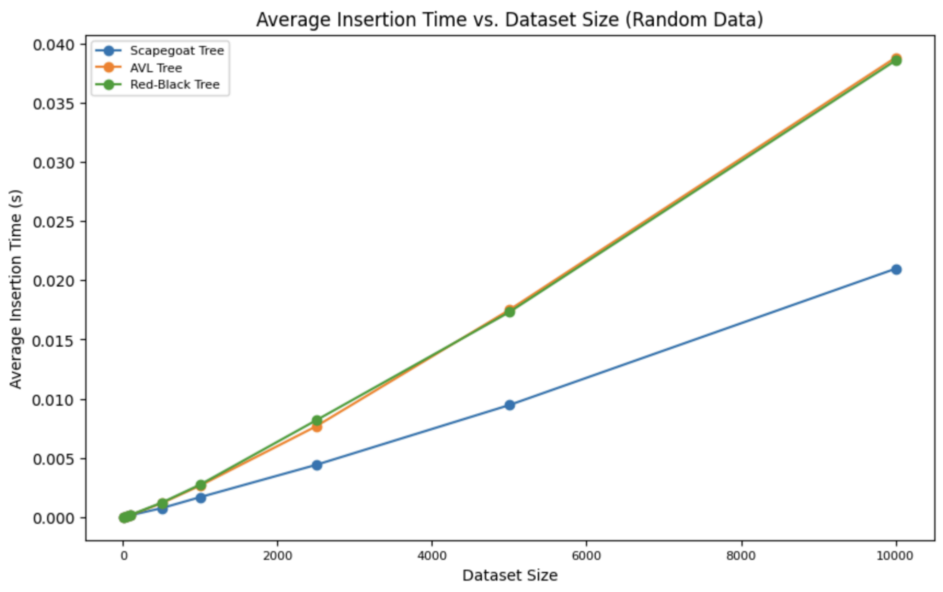
For search operations, significant differences were primarily observed between several pairs: R vs. A, R vs. OD, R vs. HD, SA vs. A, SA vs. OD, SA vs. HD, and SD vs. OD. The boxplot shows that OD has markedly faster search times, while A dataset demonstrates higher median search times with greater variability. The Scapegoat Tree’s search algorithm is the same as that of a BST, so these results are expected.

These results demonstrate how the Scapegoat Tree's performance varies substantially across different data patterns. The lazy rebalancing strategy proves efficient for sorted ascending data but performs poorly with mixed datasets containing duplicates that trigger frequent rebuilding operations.

3 Comparative Assessment

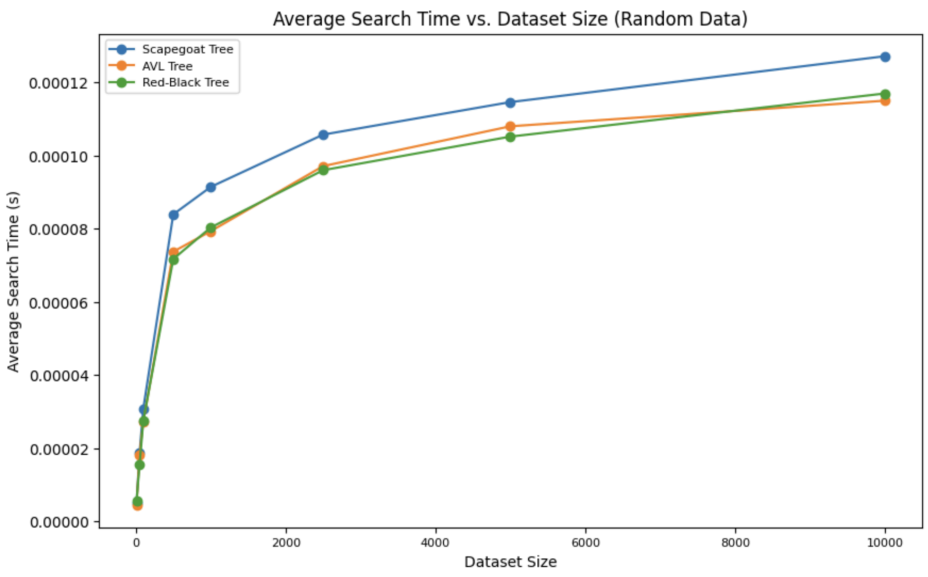
To compare the three algorithms, we can compare their insertion and search performance over the different data sets. While they trend similarly, there are notable differences in their operation times. As examples, line graphs for the random data type are presented.

Insertion Performance



Linear growth is observed in the three data structures, with **AVL** and **LLRB Trees** showing highly similar performance. This is a result of their balancing mechanisms which come into effect after every insertion operation, preventing extreme efficiencies. While the **Scapegoat Tree** is faster for insertion operations for random datasets, it has high overall variance and can become much slower depending on the data type being sorted. Note that the **AVL** and **LLRB Trees** have high consistency and will maintain high performance for data sets of any type and size.

Search Performance



We observe a clear logarithmic trend in search performance across all three data structures, which is a direct consequence of how the trees maintain balance. However, there are some *key differences*: **AVL Tree** and **LLRB Tree** consistently maintain the lowest search times. This is because both trees maintain a strict balance after every insertion, ensuring that the height remains at most log N. As a result, they require at most log N comparisons to reach a leaf node during a search. **Scapegoat Tree** performs slightly worse in search due to its lazy balancing approach, as there is some leniency in the total depth of the tree, denoted by the α-threshold. Decreasing this value would increase insertion times due to more frequent rebuild operations, but reduce search times as the tree would be shallower.

4 Team Contributions

Student Name	**Student Portico ID**	**Key Contributions**	**Share of work**[1]
Richik Mandal	24020740	Wrote the experimental framework as well as the specification for the AVL Tree.	25 %
Teo Montero	24020315	Wrote the specification and performance report for the LLRB tree as well as the Test Data generator.	25 %
Matthew Mok	24169216	Wrote the performance report for the AVL Tree as well as the Comparative analysis between the 3 algorithms used.	25 %
Eima Miyasaka	24158082	Wrote the specification and the performance report for the Scapegoat Tree and helped with the Comparative analysis.	25 %

[1] This should be a **percentage**. For example, in a group of 4 students, if all members contributed equally (i.e., the ideal scenario), their share of work would be 25% each.