Group 17: Lucy Cui, Aadhik Easwar, Vayk Mathrani, Ashley Tam

Theoretical Analysis

Sequential Search: Search and insert operations have a time complexity of $O(n)$ for average and worse-case scenarios, where n is the number of elements in the set. Search space complexity is $O(1)$ for all cases as one variable is required to store the index of the current element being checked. Insert would be $O(n)$for all cases, to store the list as a new element may be appended to the end.

Binary Search Tree: In the worst case, the tree is a linked list with height n and space complexity $O(n)$ for all operations. In the average case, the balanced tree has height $log\ n$ with space complexity $O(log\ n)$ as the number of nodes on each level doubles. The time complexity of a binary search tree is $O(n)$ for the worst case and $O(log\ n)$for the average case, where n is the height of the tree. $O(1)$ is the time complexity for inserting a node.

Balanced Search Tree (Left-Leaning Red-Black BST): The worst-case space complexity of a red-black tree is $O(n)$, where n is the number of nodes in the tree. The actual space required depends on key distribution, therefore the average-case space complexity is less than $O(n)$, simplified to $O(n)$. A LLRB BST has a maximum height of $2log_2(n+1)$ for n nodes.In the worst case, with completely unbalanced trees, all operations have a time complexity of $O(log\ n)$. The average case is slightly less than $O(log\ n)$, depending on the key distribution.

Bloom filter: The Bloom filter has a constant time complexity of $O(k)$ for both insertion and search operations, where k is the number of hash functions used in the algorithm. Both operations require k hash function evaluations and k bit array lookups.
The space complexity of the filter is $O(m)$, where m is the size of the bit array.

Experimental analysis

Real data

As the size of the test files increases, the time complexity of operations using sequential search increases exponentially. However, the search operations performed by the other algorithms are not significantly affected.

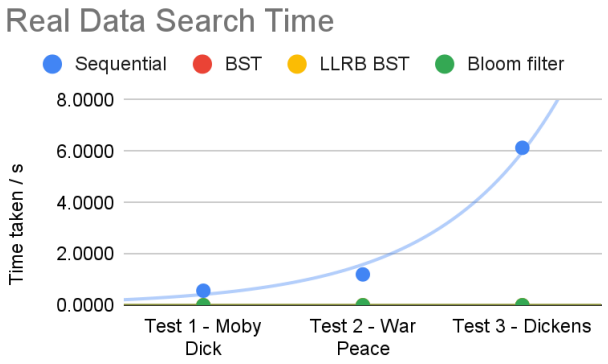


Figure 1.1

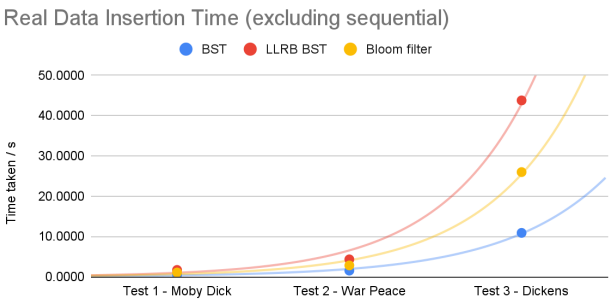


Figure 1.2

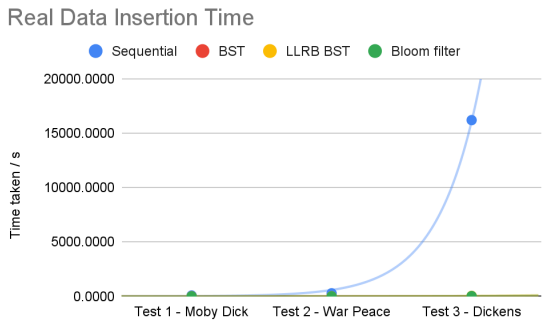


Figure 1.3

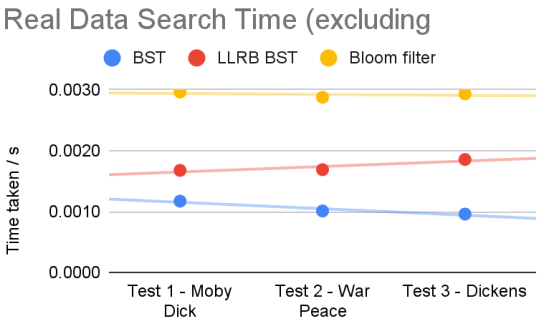


Figure 1.4

Synthetic data

We conducted experiments with each algorithm by manipulating 5 aspects of generated synthetic data, then measuring the time required for insertion and searches in each data set. The purpose of the experiments is to analyze how different aspects of the data used affect the performance time of the algorithms.

Synthetic data 1 - number of strings:

Random strings of 5-10 characters are generated for search and insert operations. The number of strings used increases for each experiment to observe time growth.

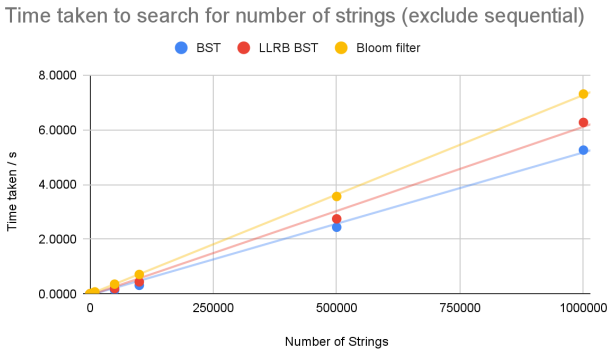


Figure 2.1

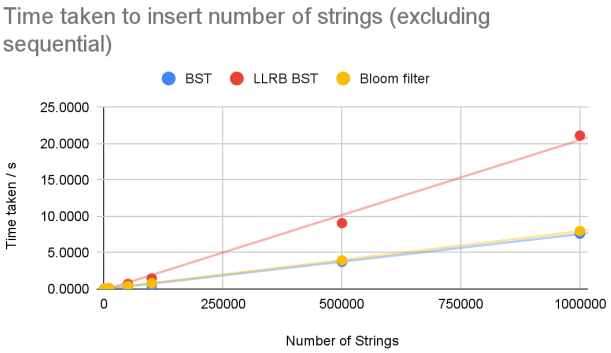


Figure 2.2

The results of this experiment are consistent with the theoretical analyses. Figure 2.2, shows the clear advantage binary search trees and bloom filters have over balanced search trees in terms of time complexity for insertion operations. The time complexity for operations using sequential search is illustrated in Figures 2.3 and 2.4 in the appendix.

Synthetic data 2 - Insertion Order:

The order in which data is inserted can impact algorithm behaviour, particularly for binary search trees. We insert the same data in ascending, descending, and random order to study this impact.

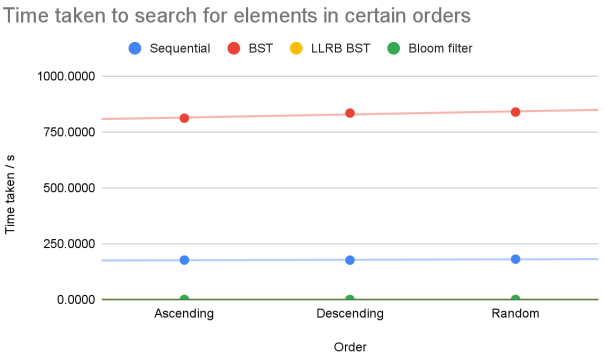


Figure 3.1

The time complexity for binary search trees is worse than even the sequential search when the data is ascending or descending. This is because it results in a highly unbalanced tree. However, more analysis is required to understand why this trend continues when data is randomised. The time complexity for inserting elements is similar to searching, illustrated in Figure 3.2 under the appendix.

Synthetic data 3 - String Length:

Strings of increasing length are generated to investigate whether longer strings lead to longer processing times due to increased complexity in comparisons.

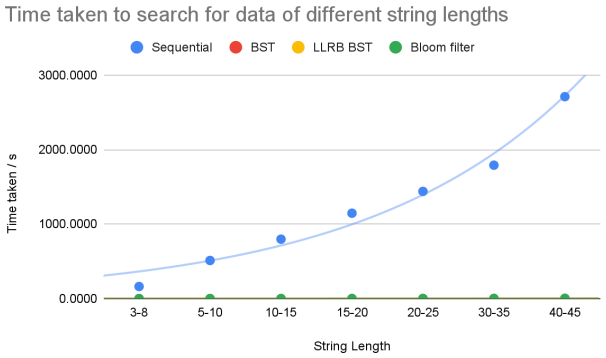


Figure 4.1

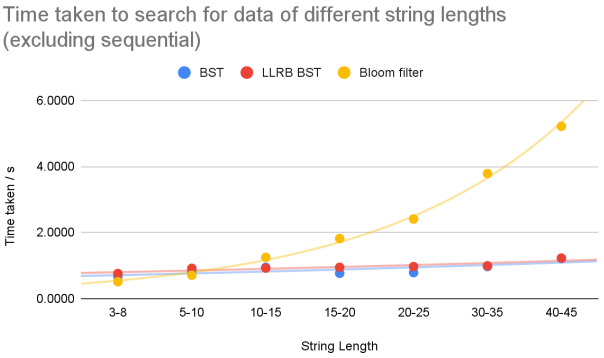


Figure 4.2

Time complexity of the bloom filter increases as the string length grows due to the hash functions of the bloom filter iterating through each character in the string before generating a hash. However, more analysis is required to understand why the sequential search shows a similar trend. Figures 4.3 and 4.4 in the appendix show the time complexities for each algorithm to perform insertion operations, which are similar to the trends in search operations.

Synthetic data 4 - Proportion of Duplicates:

Data with varying proportions of duplicates are generated by thoroughly shuffling repeated data. Since sets do not contain duplicate keys, this can result in smaller data structures and potentially reduce processing time.

The experiments disprove the theory that a high proportion of duplicates in a dataset lowers the time complexity of its operations. Figure 5.1 shows that there is lack of correlation between the proportion of duplicates in a dataset and the time complexity of search operations. A similar conclusion can be drawn from Figure 5.2 in the appendix, which shows the same for insert operations.

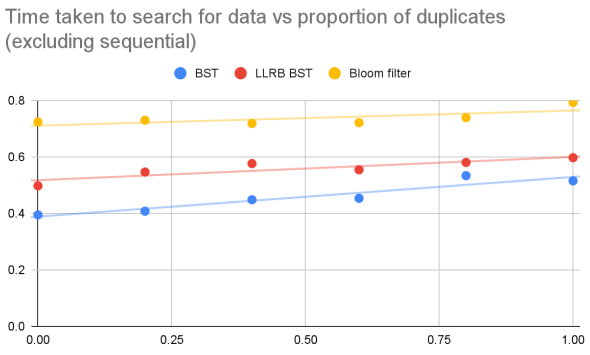


Figure 5.1

Synthetic data 5: Number of absent values

To test whether searching for non-existent values in the set takes longer than searching for values present in the set, datasets with varying proportions of search values are generated using a different set of random strings from the insertion ones.

The datasets are thoroughly shuffled to minimize the probability of overlap between the inserting and search values. Although there is a small chance that the same random strings may appear in both sets, this is negligible. Only search time is measured for this experiment.

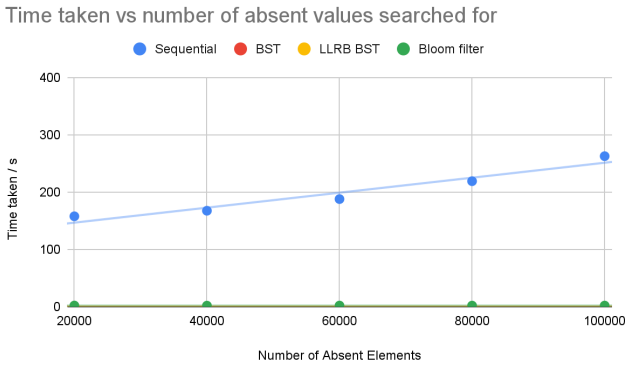


Figure 6.1

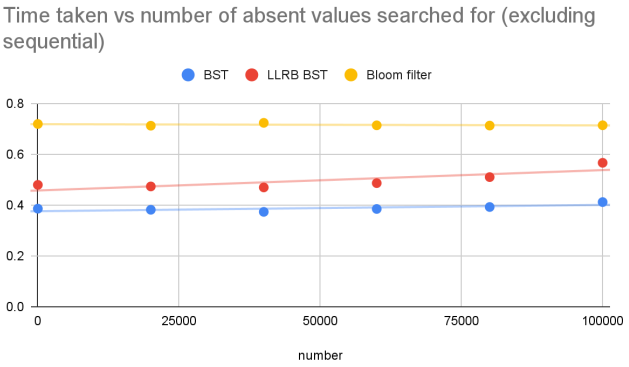


Figure 6.2

As the number of absent values searched for increases, the sequential search algorithm shows an obvious rise in the time taken since each element in the set is iterated over to find the missing value. Similarly, the time taken by the balanced and binary search trees shows a slight increase for the same reasons.

Advantages, Disadvantages and Best Scenarios

Sequential search is easy to implement and search operations have O(1) space complexity. However, its high time complexity poses a huge disadvantage in real-world situations. It is only suitable for a small-scale set-membership problems where a quick implementation is needed.

The Binary Search tree typically offers the lowest time complexity when data is in randomized order and preserves element orders. However in the worst-case situation of an unbalanced tree, time complexity could increase a lot, therefore it is most suitable for medium-sized data sets.

The Balanced Search Tree is best suited for scenarios where the number of insertions is relatively low compared to the number of searches required. It guarantees log(n) time complexity for search operations and performs better than the Binary Search Tree when the data is ordered. While it is slower than the Bloom Filter, it offers higher accuracy.

Bloom filters guarantee constant lookup times and require only a small amount of storage space. However, it is slower than the binary search tree and there is a risk of false positives. Therefore bloom filters are most effective in situations where memory storage is more limiting than time, and where accuracy is not an important factor, such as determining the uniqueness of a new username on a website.

Bloom Filter Accuracy

The bloom filter was designed with an estimated false positive rate of 1%. The accuracy of the bloom filter in the experimental and real data was 100% given the hashing functions used and the size of the bitarray used to store data. Unfortunately, due to time constraints, our group was unable to set up an experimental framework to test the relationship between accuracy and the aforementioned factors.

Appendix

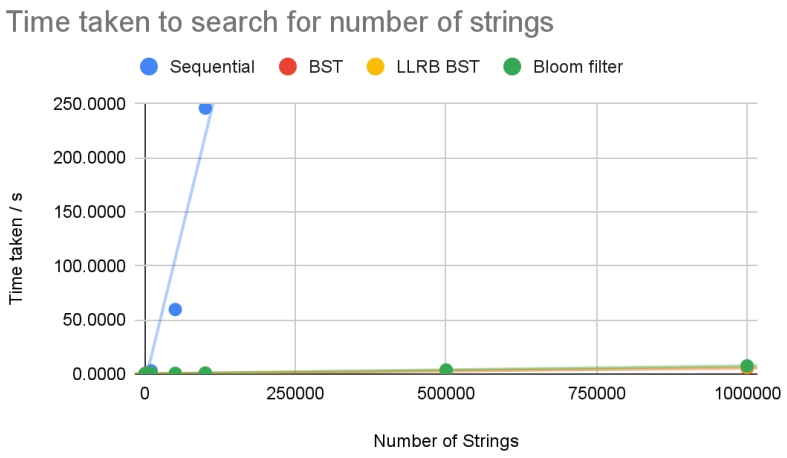


Figure 2.3

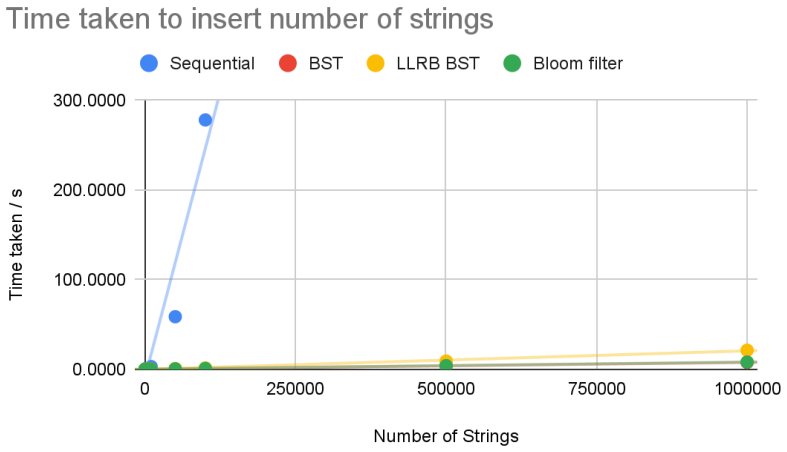


Figure 2.4

Figure 3.2

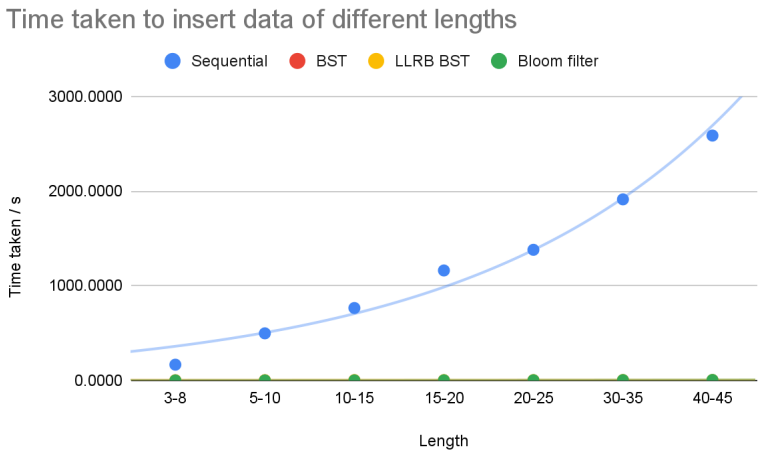


Figure 4.3

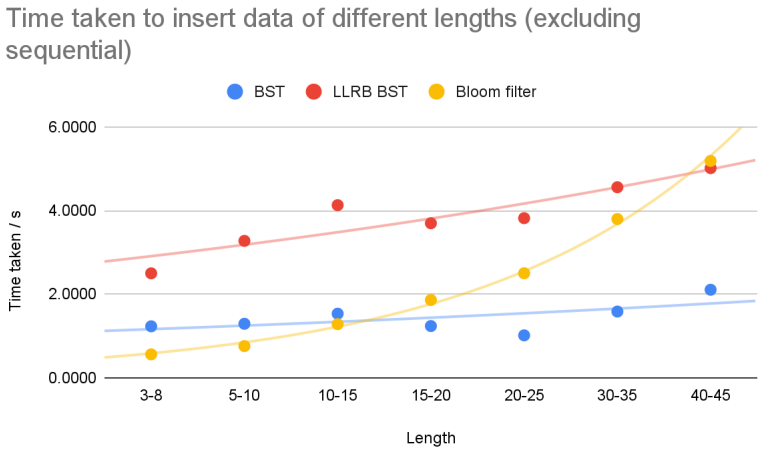


Figure 4.4

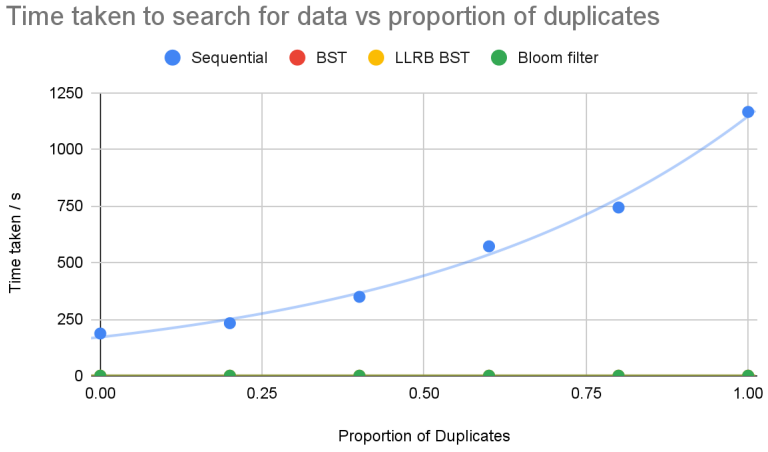


Figure 5.2

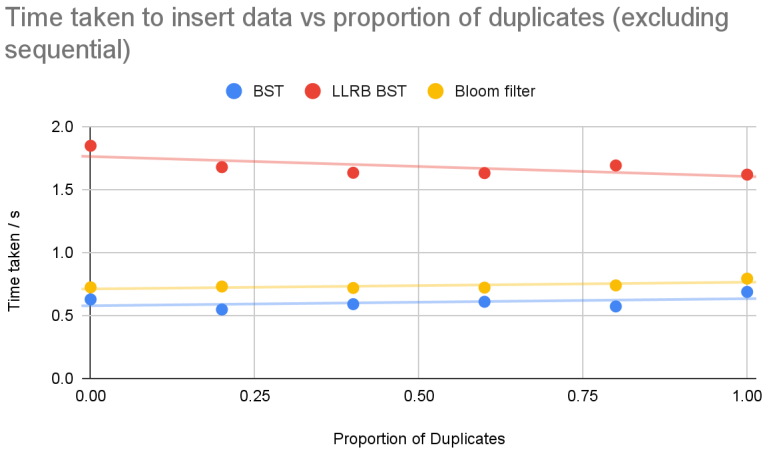
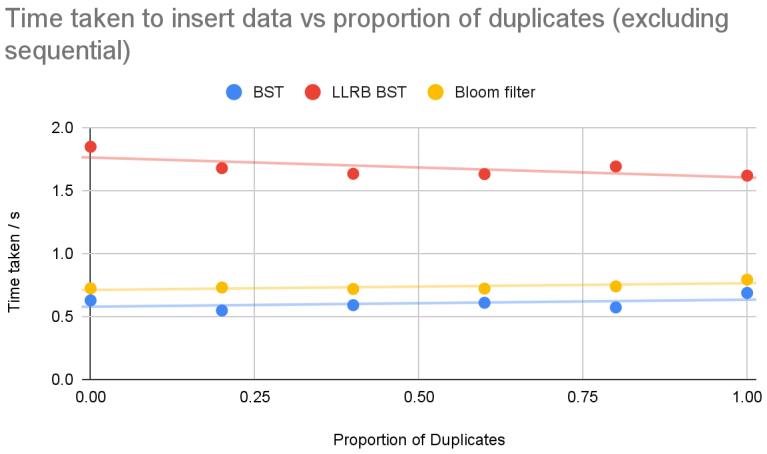

Time taken to insert data vs proportion of duplicates (excluding sequential)
BST
LLRB BST
Bloom filter
2.0
1.5
1.0
0.5
0.0
Time taken / s
0.00
0.25
0.50
0.75
1.00
Proportion of Duplicates

Figure 5.3


Time taken to insert data vs proportion of duplicates (excluding sequential)
BST
LLRB BST
Bloom filter
2.0
1.5
1.0
0.5
0.0
Time taken / s
0.00
0.25
0.50
0.75
1.00
Proportion of Duplicates