Habib
University

Spring 2024

SKIPLIST VS AVL TREES

Prepared By
Irfan Sohail **Mustafa Hussain** **Sarah Faisal** **Sabahat Zahra**

Course: Data structures II

Instructor: M. Mobeen Muvania

TABLE OF CONTENTS

PROJECT OVERVIEW

Objectives

This project investigates the efficiency of two data structures, AVL trees and skip lists, for managing a car dataset containing 100,000 entries. Each car record encompasses essential details like ID, Brand, Model, Year, Color, Mileage, Price, and Condition.

Focus

The primary objective is to compare the performance of AVL trees and skip lists in terms of time complexity for three crucial operations: insertion, searching, and deletion. We will concentrate on the time taken (in milliseconds) by each data structure to execute these operations on the car dataset, focusing on the ID field as the key for searching and operations.

Methodology

1. Data Structure Implementation: We will implement both AVL trees and skip lists in C++. AVL trees will utilize standard techniques for maintaining balance during insertions and deletions. For skip lists, we will employ a probabilistic approach using a coin-flipping strategy to determine the level of an element during insertion.
2. Performance Evaluation: We will leverage the pb_plots library to generate visual representations of the performance data. These graphs will depict the relationship between the dataset size (number of cars) and the time taken (in milliseconds) by each data structure to perform insertions, searches, and deletions.

INTRODUCTION SKIPLIST

Skip lists, a probabilistic data structure introduced by William Pugh in 1989, present a compelling alternative that prioritizes efficiency and simplicity. Skip lists are a probabilistic variant of sorted linked lists. They achieve efficient search, insertion, and deletion operations with an average time complexity of O(log n), where n represents the number of elements in the list. This performance rivals that of balanced search trees but with a simpler implementation.

The core concept of a skip list lies in its layered structure. It consists of a bottom layer, which acts as a standard sorted linked list, and a set of optional upper layers. Each element in the skip list resides in the bottom layer and has a certain chance of being included in the higher layers. These upper layers serve as "express lanes," allowing for faster traversal by skipping over a larger number of elements in the lower layers. The decision to elevate an element to a particular upper layer is typically made probabilistically, often using a coin-flipping strategy. This probabilistic approach forms the crux of the skip list's efficiency. Elements are not uniformly distributed across layers. Frequently accessed elements are more likely to be present in higher layers, facilitating quicker searches.

INTRODUCTION SKIPLIST

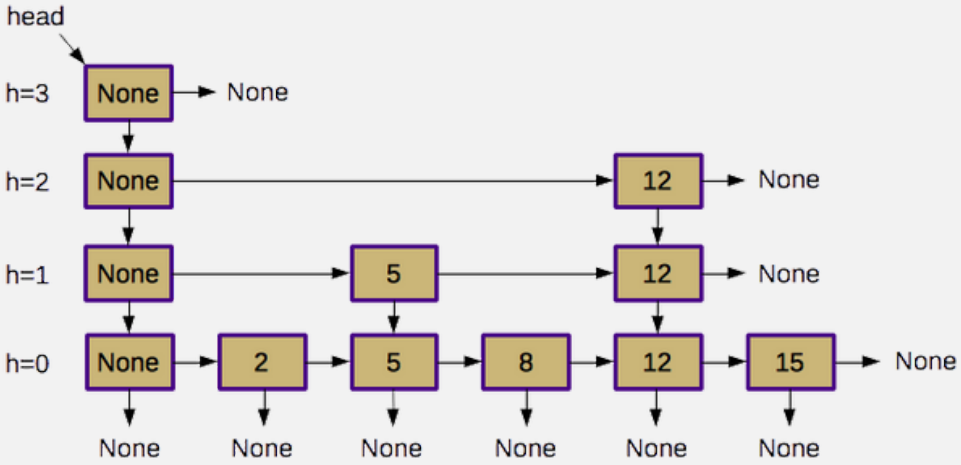

Figure 1: Basic structure of skiplist

Beyond their core functionality, skip lists offer several advantages. Their probabilistic nature makes them efficient for dynamic data sets where insertions and deletions are frequent. Additionally, their simpler structure avoids the complex rebalancing operations required by balanced search trees, leading to potentially faster insertions and deletions in specific scenarios.

However, it is essential to acknowledge the probabilistic nature of skip lists. While the average time complexity for operations remains logarithmic, the worst-case scenario can deviate from this, potentially reaching linear time. This trade-off might not be optimal for applications demanding guaranteed performance.

INTRODUCTION AVL TREES

The realm of data structures is a constant quest for efficient management of organized information. In this pursuit, balanced search trees stand out as a powerful tool for maintaining sorted data and facilitating rapid search, insertion, and deletion operations. Among these balanced search trees, AVL trees, named after their inventors Adelson-Velsky and Landis, hold a prominent position.

An AVL tree is a self-balancing binary search tree. It upholds the core principle of a binary search tree, where each node has at most two child nodes (left and right), and the key value of a node is greater than all elements in its left subtree and less than all elements in its right subtree. This characteristic ensures a sorted order when traversing the tree in order.
However, AVL trees go beyond the basic binary search tree structure by introducing a crucial concept: the balance factor. The balance factor of a node is calculated as the difference between the heights of its left and right subtrees. An AVL tree enforces a strict balance constraint – the absolute value of the balance factor for any node must be less than or equal to 1. This constraint necessitates self-balancing operations, specifically rotations, whenever insertions or deletions disrupt the balance of the tree.

While AVL trees offer guaranteed logarithmic time complexity for most operations, their self-balancing mechanisms introduce some overhead compared to simpler data structures. In scenarios where absolute predictability is less critical and slight performance variations are acceptable, skip lists, a probabilistic data structure, can offer an alternative approach.

INTRODUCTION AVL TREES

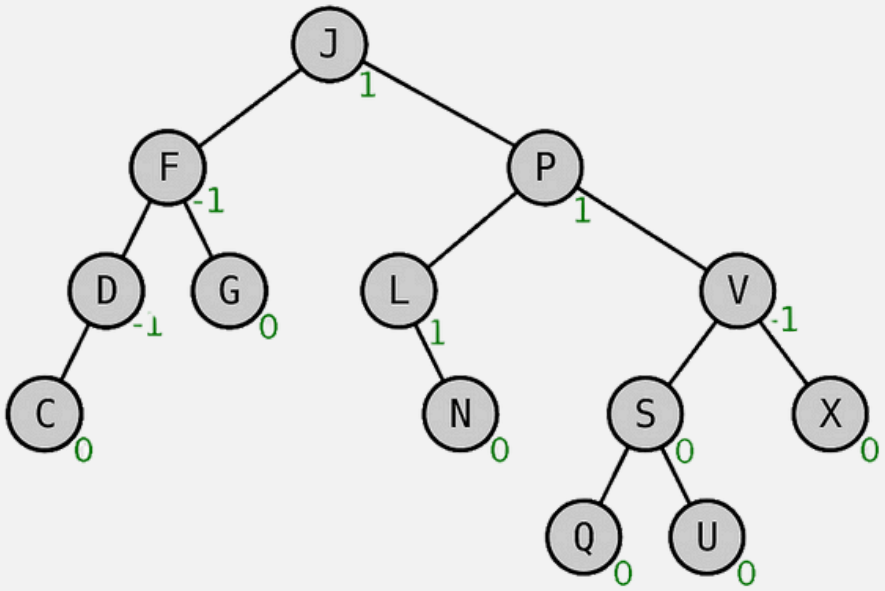

Figure 2: Basic structure of AVL tree

Rotations for Balance:

AVL trees employ various rotation techniques to maintain the balance factor after insertions or deletions. These rotations involve rearranging the subtrees around a specific node to restore the balance constraint. There are four primary rotation types:

- Left Rotation: Used when the balance factor of a node is +2 and the imbalance originates in its right subtree.
- Right Rotation: Employed when the balance factor of a node is -2 and the imbalance stems from its left subtree.
- Left-Right Rotation: A combination of a left rotation followed by a right rotation, used in specific scenarios.
- Right-Left Rotation: A combination of a right rotation followed by a left rotation, used in specific scenarios.
- These rotations ensure that the height of the AVL tree remains logarithmic in relation to the number of elements, even with numerous insertions and deletions. This logarithmic height translates into efficient time complexity for search, insertion, and deletion operations, all averaging O(log n).

ANALYSES: INSERTION

SkipList

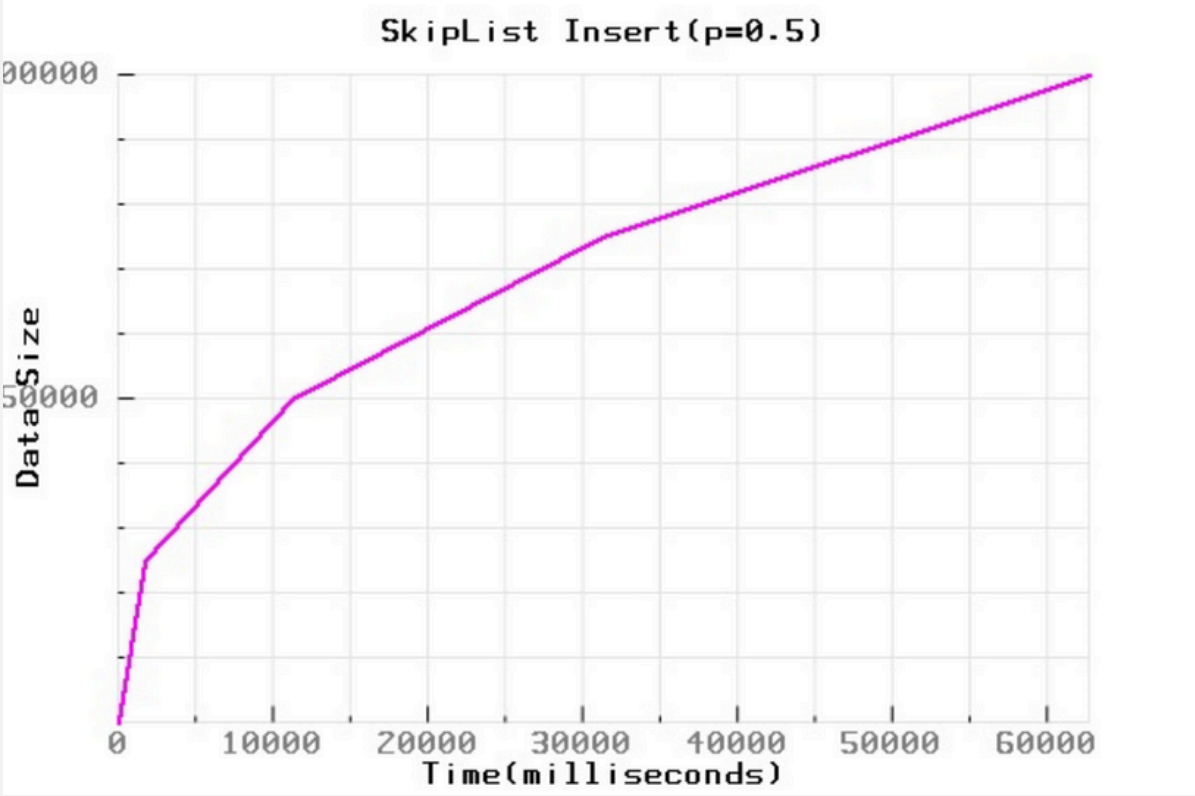

Figure 3:Insert graph for skiplist

ANALYSES: INSERTION

AVL Tree

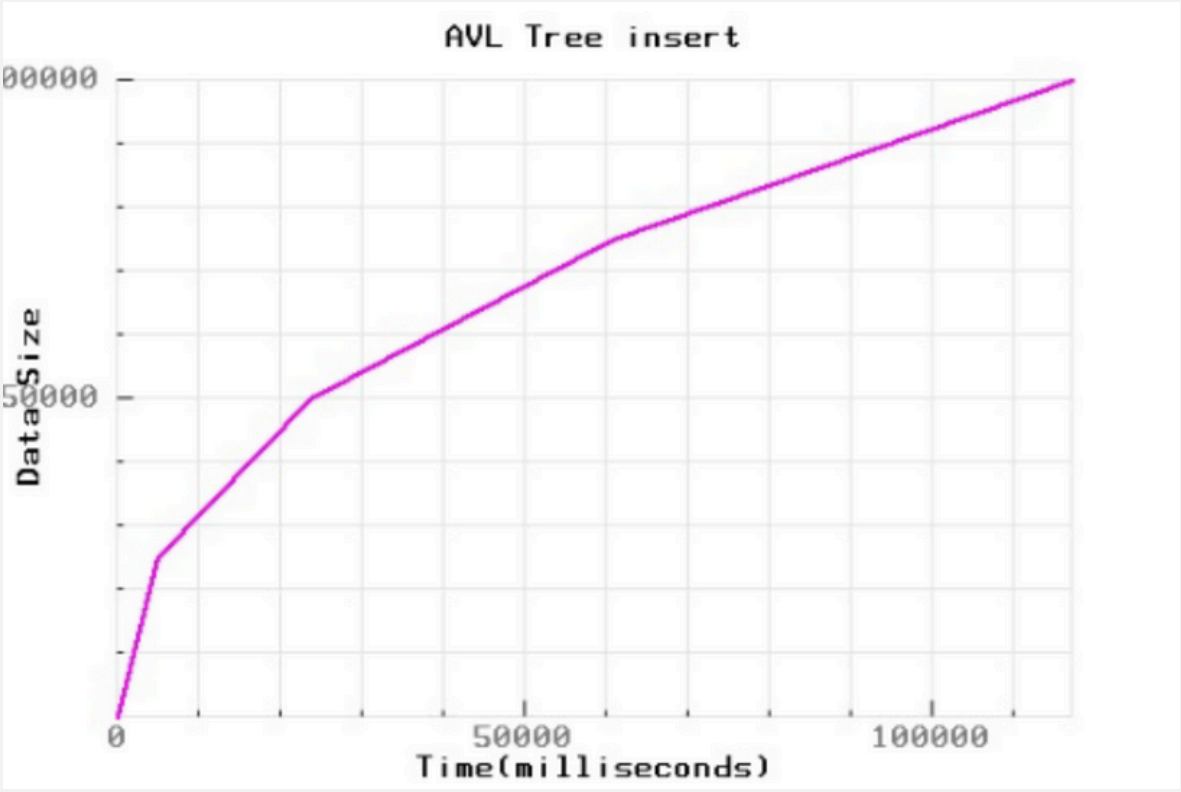

Figure 4:Insert graph for AVL tree

COMPARATIVE ANALYSES

Insertion

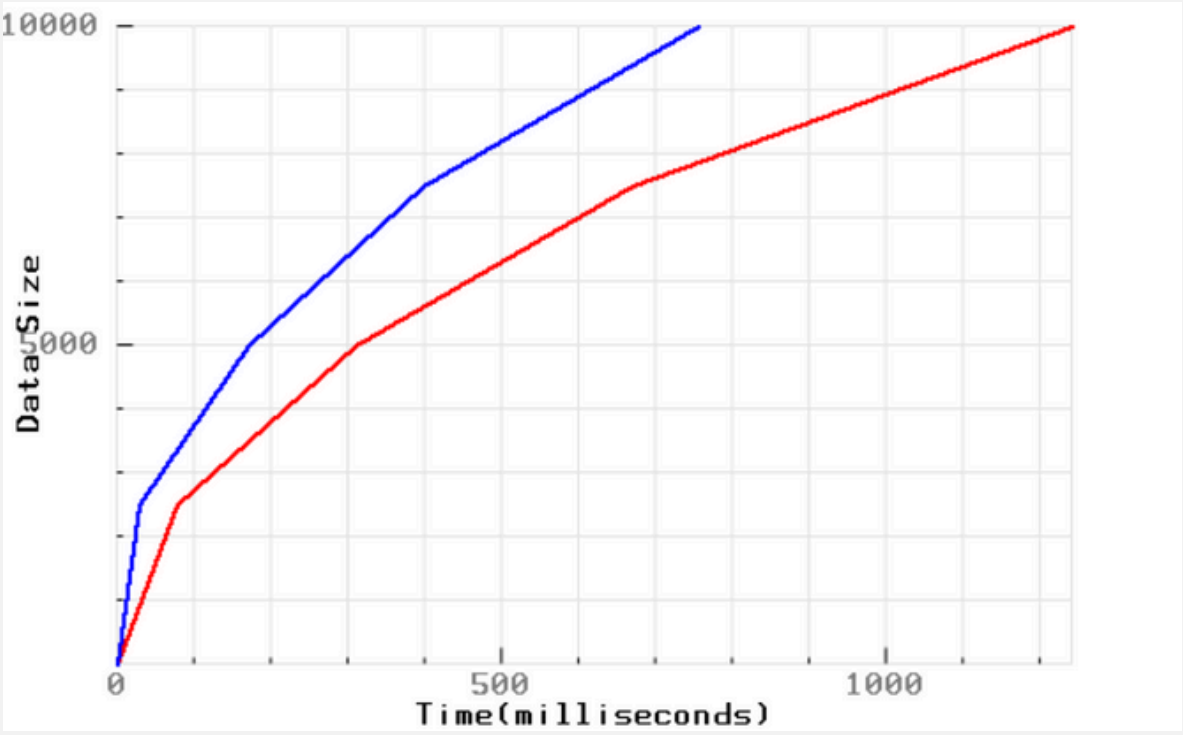

Figure 5: AVL vs Skiplist insertion comparison

Remarks:

- As we can observe in the above graph the skiplist outperforms the AVL, as the insertion time of the skiplist is faster than AVL. The Skiplist's probabilistic structure allows for faster insertions due to its skip levels, which reduce the number of comparisons needed during insertion.

ANALYSES: SEARCH

SkipList

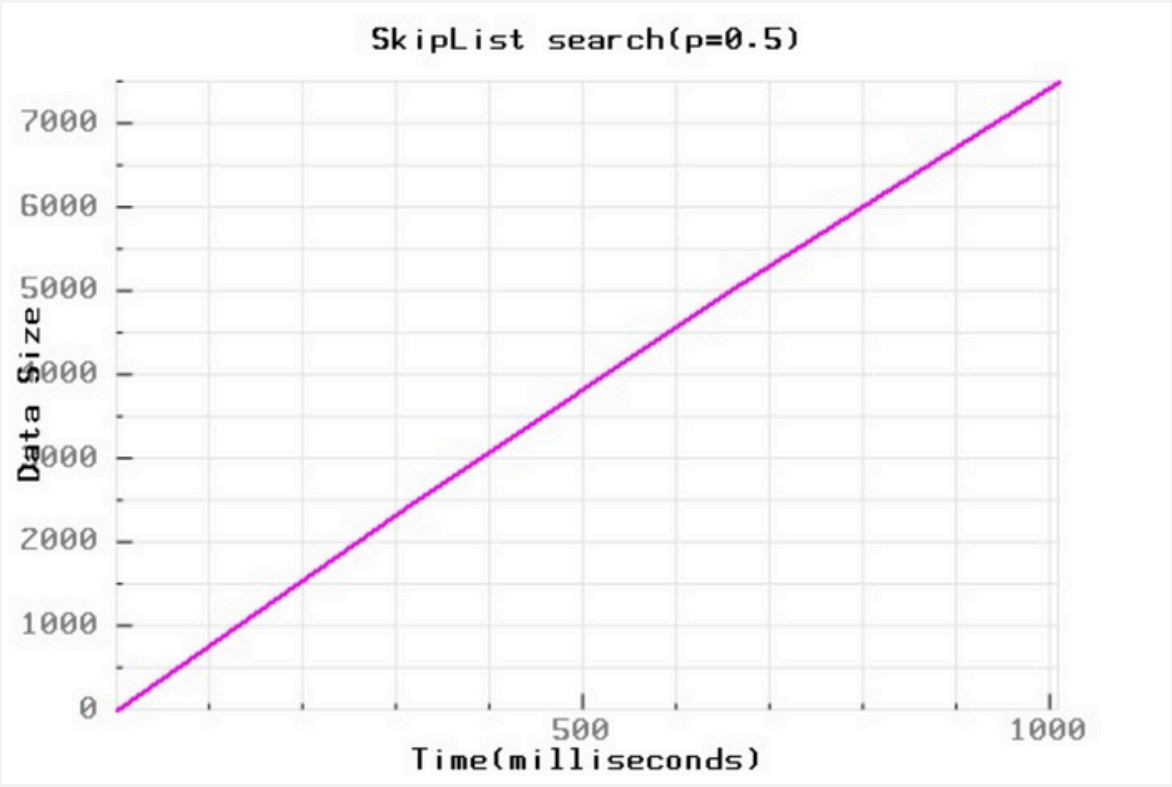

Figure 6: Search graph for Skiplist

ANALYSES: SEARCH

AVL Tree

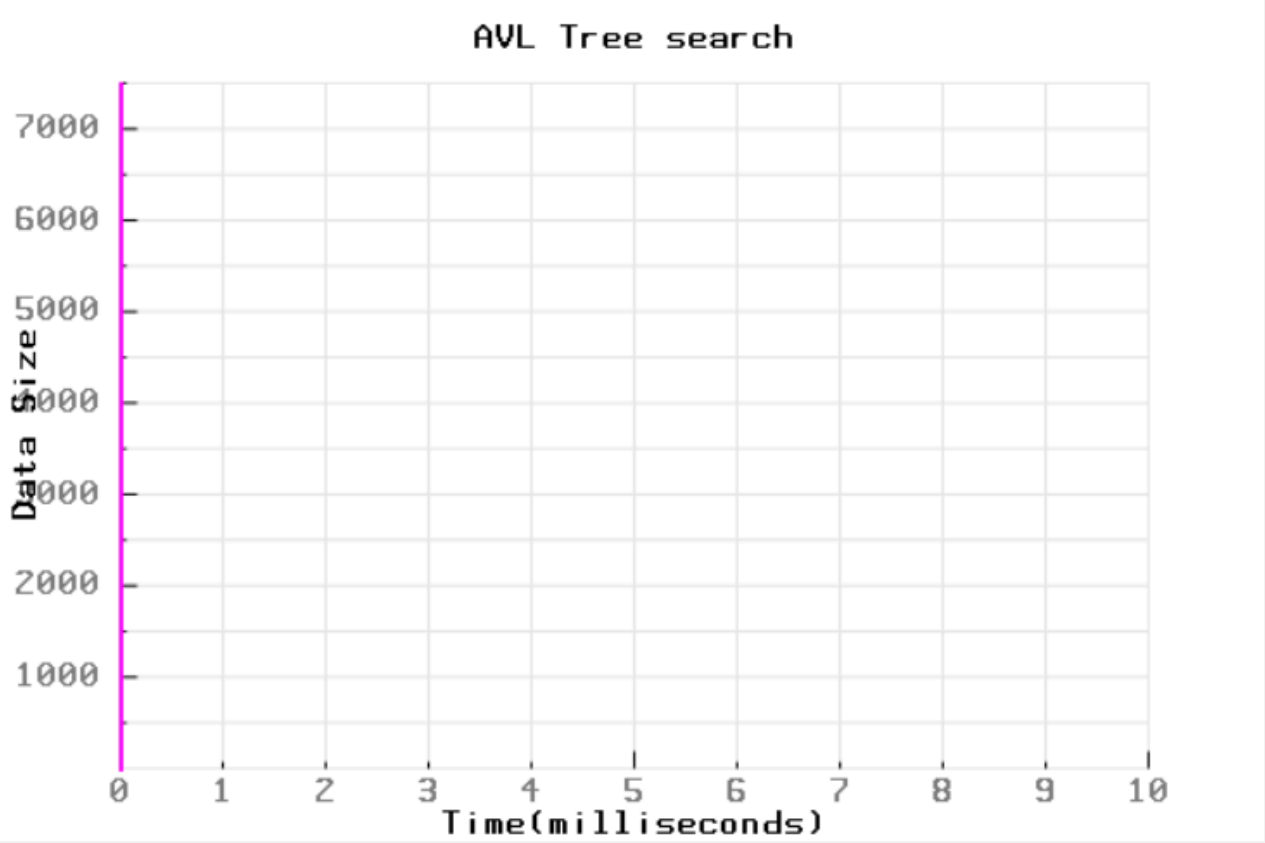

Figure 7: Search graph for AVL tree

COMPARATIVE ANALYSES

Searching

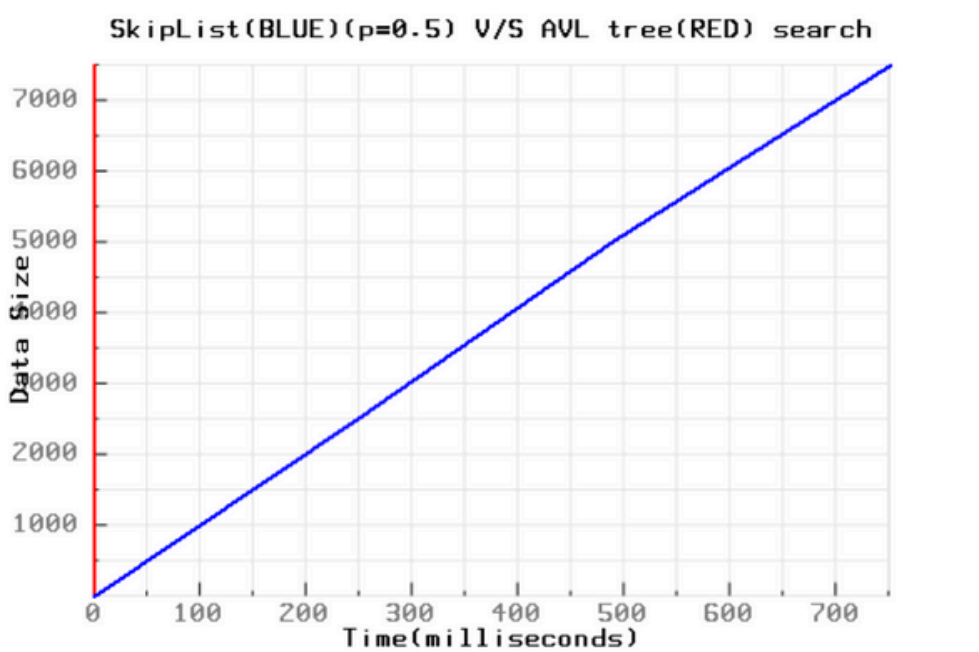

Figure 8: AVL vs Skiplist search comparison

Remarks:

- Our analysis of search operations reveals a clear distinction between the performance of skiplists and AVL trees. The AVL tree significantly outperforms the skiplist in terms of search speed. This is reflected in the graphs, where the AVL tree exhibits a flat line across most of the dataset size range, indicating consistently fast searches. This efficiency stems from the AVL tree's guaranteed logarithmic time complexity, ensured by its self-balancing properties. Even with small datasets, searches on the AVL tree are negligible due to its small height. Conversely, the skiplist exhibits a near-linear trend in its search time graph. While skiplists offer consistent performance in both best and worst-case scenarios, their search time becomes noticeably slower for larger datasets compared to the AVL tree.

ANALYSES: DELETE

SkipList

Figure 9: Delete graph for Skiplist

ANALYSES: DELETE

AVL Tree

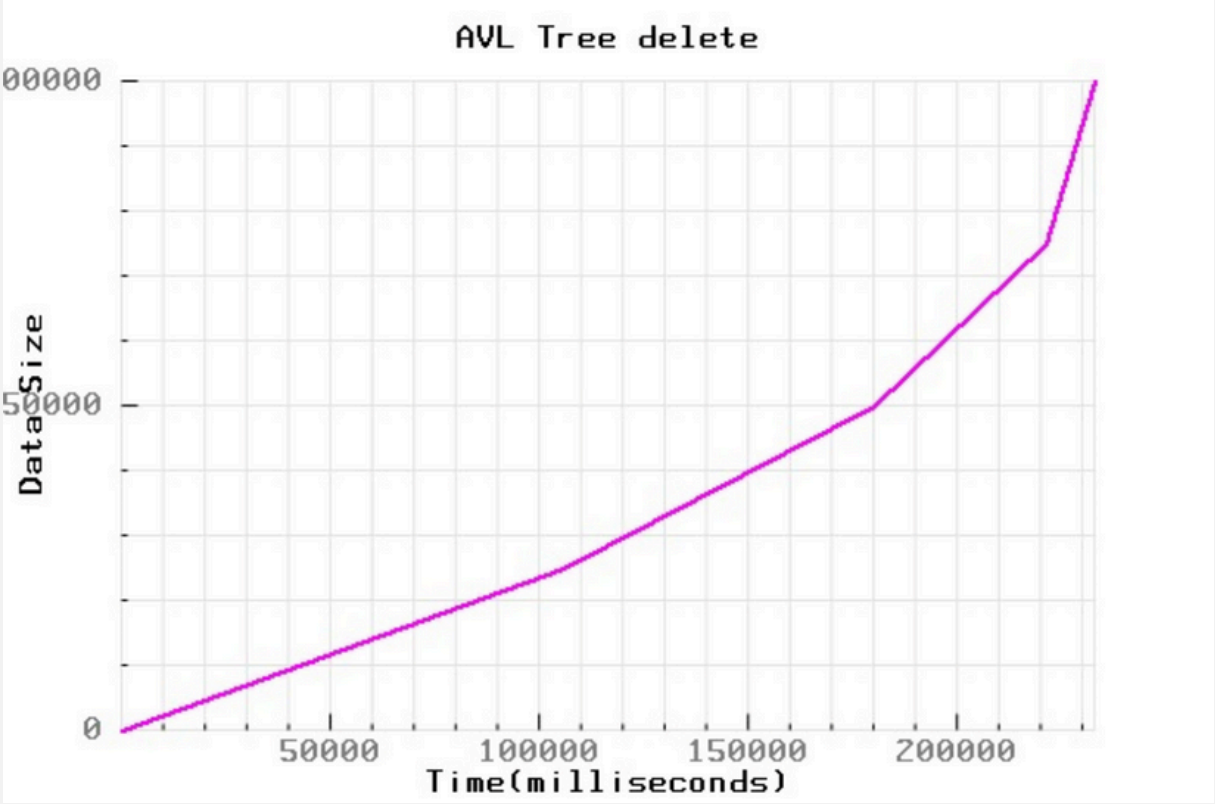

Figure 10: Delete graph for AVL tree

COMPARATIVE ANALYSES

Deletion

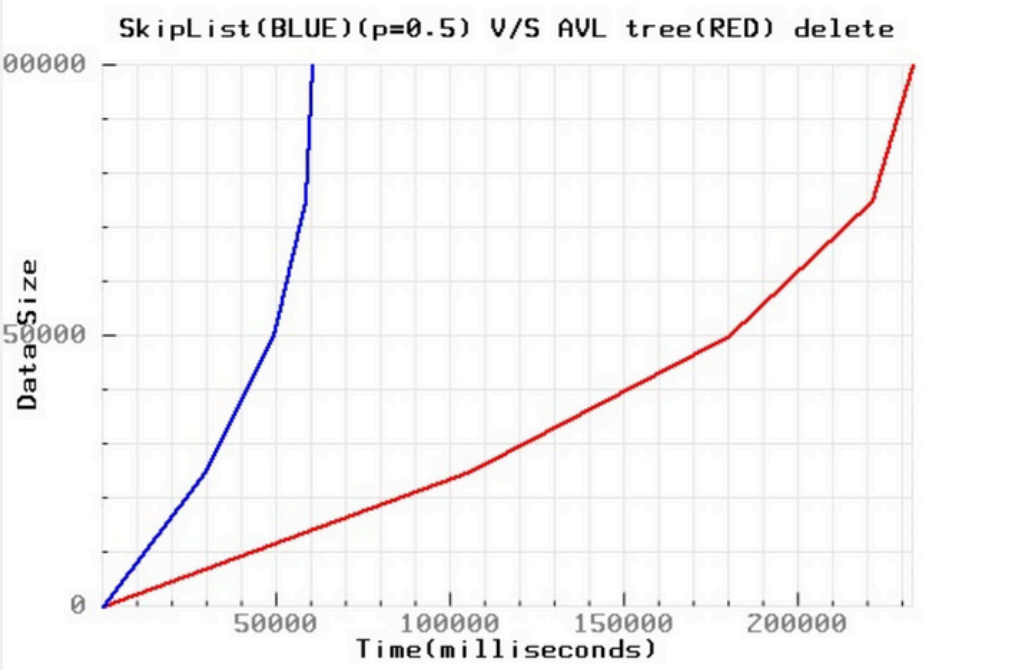

Figure 11: AVL vs Skiplist Delete comparison

Remarks:

- Initially, the skiplist appears significantly faster than the AVL tree for deletions. However, both data structures exhibit a logarithmic trend in their time complexity as the dataset size increases. This seemingly slower performance of the AVL tree stems from its rebalancing mechanism. Maintaining balance in the AVL tree becomes more time-consuming with larger datasets, impacting deletion speed. Conversely, the skiplist's probabilistic approach avoids such overhead. However, as the dataset shrinks due to deletions, the AVL tree's rebalancing burden diminishes, leading to faster deletion times for remaining elements.

OPTIMIZATION OF SKIPLIST

The inherent probabilistic nature of skiplists offers opportunities for performance optimization. A key factor influencing skiplist efficiency is the probability distribution used to determine the level (layer) an element is inserted into. The default approach often employs a coin-flipping strategy, where the probability of inclusion in a higher level is typically set to 0.5. While this approach offers a balance between average search time and space complexity, it might not be optimal for all scenarios.

This section delves into the potential for optimizing skiplist performance by adjusting the probability distribution for element placement within the skiplist's layers. Our investigation focused on three distinct probability values for inclusion in higher levels: 0.5 (default), 0.25, and 0.125.

Impact of Probability Distribution:

- Higher Probability (0.5): This setting, often the default, promotes a balanced approach. Elements have a higher chance of residing in upper levels, leading to faster average search times but potentially requiring more space due to increased levels in the skiplist.
- Lower Probability (0.25, 0.125): By decreasing the probability of inclusion in upper levels, the skiplist becomes more compact. However, this optimization might come at the cost of slightly slower average search times. Fewer elements in higher levels can increase the number of comparisons needed during searches.

INSERTION WITH VARYING PROBABILITIES

Insertion

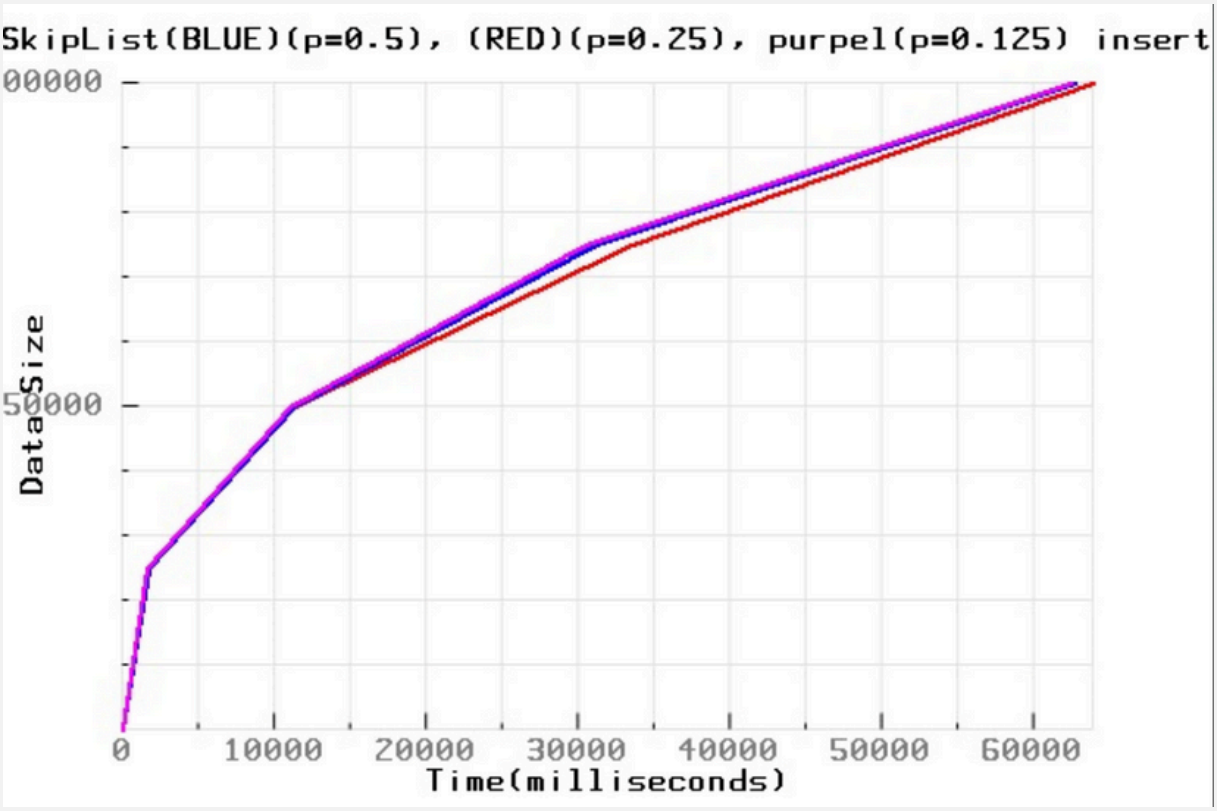

Figure 12: Insertion in skiplist of varying probability

Remarks:

- The analysis of insertion time reveals an interesting trend. For smaller datasets, the probability assigned to elements during skiplist insertion appears to have minimal impact on the time complexity of the operation. However, as the dataset size grows (as evidenced by the graph), the skiplist configured with a probability of 0.125 demonstrates superior performance in terms of insertion time compared to those using probabilities of 0.5 and 0.25.

SEARCH WITH VARYING PROBABILITIES

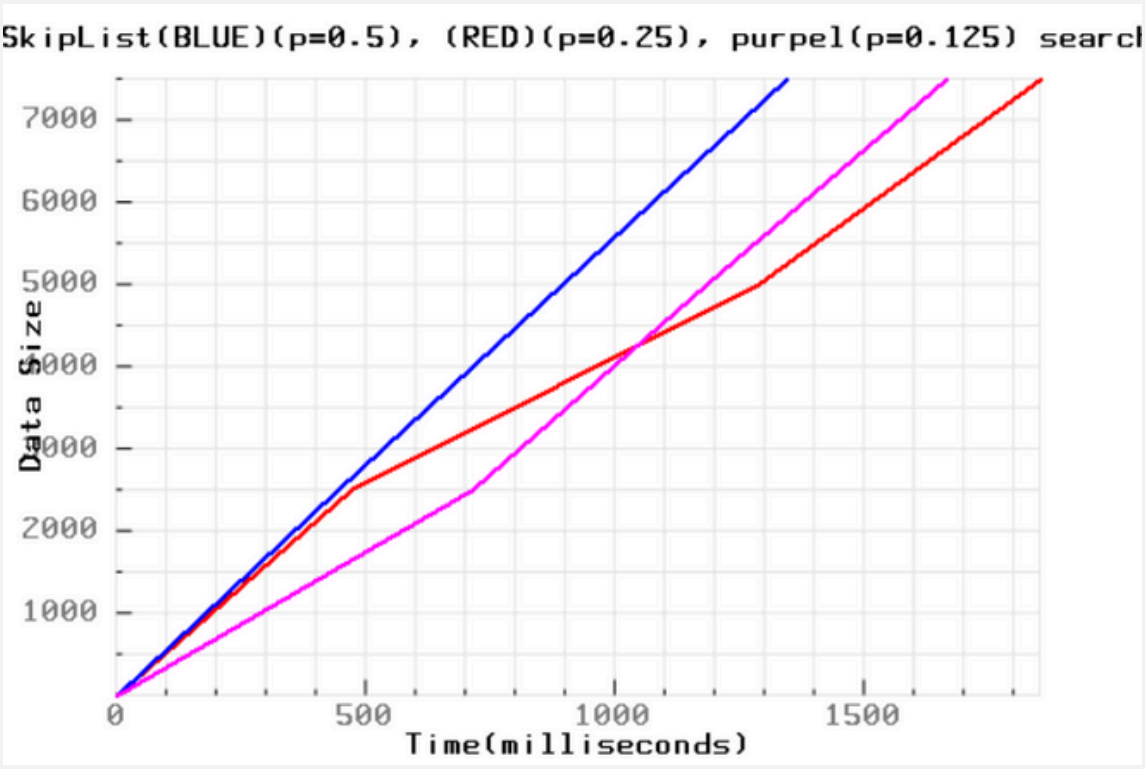

Figure 13: Search in skiplist of varying probability

Remarks:

- The analysis of search time indicates that the skiplist with a probability of 0.5 exhibits the fastest average search performance among the tested configurations. This can be attributed to the balanced approach this probability distribution fosters. By having a 50% chance of inclusion in a higher level, elements are more likely to be distributed across various layers, leading to faster searches on average. While lower probabilities (0.25 and 0.125) might create a more compact skiplist, they also decrease the number of elements in higher levels, potentially requiring more comparisons during searches, thus increasing search time.

DELETION WITH VARYING PROBABILITIES

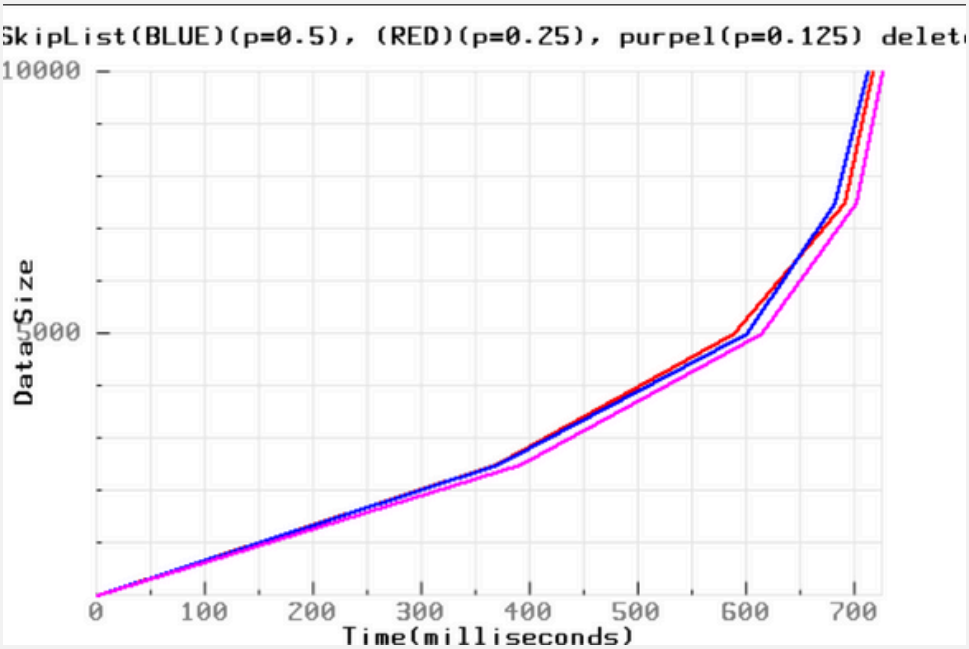

Figure 14: Deletion in skiplist of varying probability

Remarks:

- For smaller datasets, the difference in deletion time between various probabilities seems negligible. However, as the dataset grows, the skiplist with a probability of 0.5 demonstrates a performance advantage for deleting elements. This can be explained by the trade-off between search efficiency and space complexity in skiplists. A higher probability (0.5) increases the likelihood of elements residing in upper layers. While this might require slightly more space, it also facilitates faster deletions. Upper layers allow for skipping over more elements during deletions, leading to faster overall deletion times for larger datasets.

KEY FINDINGS

Insertion Time:

- Skiplist vs AVL Tree: Skiplists generally outperform AVL trees in terms of insertion time, especially for larger datasets.
- Reason: Skiplists leverage their probabilistic structure with "skip levels" that reduce the number of comparisons needed during insertions.

Search Time:

- Skiplist vs AVL Tree: AVL trees significantly outperform Skiplists in search speed across most dataset sizes.
- Reason: AVL trees guarantee logarithmic time complexity for search operations due to their self-balancing properties. This ensures consistently fast searches even with small datasets.
- Skiplist Search: While Skiplists offer consistent performance in both best and worst-case scenarios, their search time becomes noticeably slower for larger datasets compared to AVL trees. This is because Skiplists rely on a probabilistic approach for element placement, leading to a near-linear trend in search time for larger datasets.

Deletion Time:

- Initial Performance: Initially, Skiplists appear significantly faster than AVL trees for deletions.
- Long-Term Performance: As the dataset size increases, both data structures exhibit a logarithmic trend in deletion time complexity.
- AVL Tree Deletion: The seemingly slower deletion performance of AVL trees for larger datasets stems from their rebalancing mechanism. Maintaining balance becomes more time-consuming with larger datasets, impacting deletion speed.
- Skiplist Deletion: Conversely, the Skiplist's probabilistic approach avoids such overhead. However, as the dataset shrinks due to deletions, the AVL tree's rebalancing burden diminishes, leading to faster deletion times for the remaining elements.

KEY FINDINGS

Impact of Probability Distribution in Skiplists:

- Minimal Impact for Small Datasets: The probability assigned to elements during Skiplist insertion has minimal impact on insertion time for smaller datasets.
- Impact on Larger Datasets: As the dataset size grows, the Skiplist configured with a probability of 0.25 demonstrates superior performance in terms of insertion time compared to those using probabilities of 0.5 and 0.125.

Optimal Probability for Skiplist Search:

- Balanced Approach: The analysis of search time indicates that the Skiplist with a probability of 0.5 exhibits the fastest average search performance among the tested configurations.
- Reasoning: This can be attributed to the balanced approach this probability distribution fosters. By having a 50% chance of inclusion in a higher level, elements are more likely to be distributed across various layers, leading to faster searches on average. While lower probabilities (0.25 and 0.125) might create a more compact Skiplist, they also decrease the number of elements in higher levels, potentially requiring more comparisons during searches, thus increasing search time.

Probability and Deletion Time:

- Negligible Impact for Small Datasets: For smaller datasets, the difference in deletion time between various probabilities in Skiplists seems negligible.
- Advantage of Probability 0.5 for Larger Datasets: However, as the dataset grows, the Skiplist with a probability of 0.5 demonstrates a performance advantage for deleting elements.
- Reasoning: This can be explained by the trade-off between search efficiency and space complexity in Skiplists. A higher probability (0.5) increases the likelihood of elements residing in upper layers. While this might require slightly more space, it also facilitates faster deletions. Upper layers allow for skipping over more elements during deletions, leading to faster overall deletion times for larger datasets.

CONCLUSION

This project has delved into the performance characteristics of AVL trees and Skiplists, two data structures well-suited for managing sorted data. Our analysis revealed their strengths and weaknesses, providing valuable insights into their appropriate use cases.

The choice between a Skiplist and an AVL tree depends on the specific needs of your application. When fast insertions and consistent performance for searches and deletions are paramount, Skiplists might be a compelling option. Conversely, if consistent and fast searches are an absolute necessity, even with slightly slower insertions, AVL trees are the preferred choice. Additionally, for applications with simple data structures and lower memory constraints, Skiplists offer an enticing alternative due to their simpler implementation.

Ultimately, the best data structure for your project depends on the specific trade-offs you are willing to make in terms of speed, simplicity, and space complexity. By considering the insights provided in this project, you can select the most appropriate data structure to optimize the performance and efficiency of your application.

REFERENCES

AVL Tree:

1. GeeksforGeeks (2023) AVL Tree Data Structure, GeeksforGeeks. Available at: https://www.geeksforgeeks.org/introduction-to-avl-tree/ (Accessed: 28 April 2024).
2. Claudiaa, S. (2023) Comparison between AVL, Skip List and Segment Tree, Medium. Available at: https://medium.com/@serbu.claudiaa/comparison-between-avl-skip-list-and-segment-tree-291e9b873074 (Accessed: 28 April 2024).
3. sgmmsgmm 59022 gold badges88 silver badges1313 bronze badges and rts1rts1 1 (1961) What is the downside to using an AVL tree?, Stack Overflow. Available at: https://stackoverflow.com/questions/33154475/what-is-the-downside-to-using-an-avl-tree (Accessed: 28 April 2024).

SkipLisk

1. (No date) Skip list: Implementation, optimization and web search. Available at: https://www.cscjournals.org/manuscript/Journals/IJEA/Volume5/Issue1/IJEA-45.pdf (Accessed: 28 April 2024).
2. GeeksforGeeks (2023b) Skip list: Set 1 (introduction), GeeksforGeeks. Available at: https://www.geeksforgeeks.org/skip-list/ (Accessed: 28 April 2024).
3. GeeksforGeeks (2023a) AVL Tree Data Structure, GeeksforGeeks. Available at: https://www.geeksforgeeks.org/introduction-to-avl-tree/ (Accessed: 28 April 2024).
4. Claudiaa, S. (2023a) Comparison between AVL, Skip List and Segment Tree, Medium. Available at: https://medium.com/@serbu.claudiaa/comparison-between-avl-skip-list-and-segment-tree-291e9b873074 (Accessed: 28 April 2024).

Library for plotting:
https://github.com/InductiveComputerScience/pbPlots