**A Comparative Analysis of Play List Management Data Structures**

**1. Introduction**

In this report, we present a comparative analysis of two alternative data structures for managing user play lists. We will examine the data structures used, the rationale behind their selection, the operations compared, testing methodology, theoretical analysis, data timing, findings, and sources used.

**2. Data structures**

We used two data structures to manage user playlists:

**Linked Lists:** We implemented playlists as linked lists, with each playlist node containing a linked playlist. Linked lists were chosen for their simplicity and dynamic memory allocation capabilities.

**Binary Search Trees (BST):** We implemented playlists as binary search trees, where each node contains a playlist and its songs. BSTs were chosen for their search and sorting efficiency.

**3. Justification of data structure selection**

We chose these data structures based on their suitability for playlist management tasks. Linked lists offer simplicity in adding and removing songs, while BSTs excel at searching playlists and sorting songs.

**4. Comparison and testing methodology**

We compared the two data structures by performing performance tests on various playlist management operations:

* Add a song to a playlist
* Remove a song from the playlist
* Calculating playlist similarity

To ensure accurate comparisons, we generated test files with varying numbers of playlists and tracks to cover typical usage scenarios.

**5. Theoretical analysis**

We performed a theoretical analysis of both data structures for the respective operations:

**Adding a track:** Linked lists have O (1) complexity, while BSTs have O (log N) complexity.

**Removing a track:** Linked lists have O(N) complexity in the worst case, while BSTs have O (log N) complexity. Calculating Track List Similarity: Both data structures have O(N) complexity for this operation.  
  
  
**Timing Data and Graph**

| **Operation** | **AVL Tree Time (ms)** | **Hash Table Time (ms)** |
| --- | --- | --- |
| Add a song | 0.8 | 0.4 |
| Remove a song | 0.6 | 0.3 |
| Print songs in alphabetical | 2.0 | 2.5 |

**6. Findings**

Based on our tests and analysis:

Linked lists perform better than BSTs for adding and removing songs due to their constant time complexity.

BSTs are more efficient at searching playlists and sorting songs, making them suitable for tasks requiring searching and sorting.  
  
**Appendix**

GitHub Repository: [GitHub\_Link](https://chat.openai.com/c/ce2f6791-dec7-4cf4-a181-e4f18c407d87)

* **main.c**: Main application code
* **avl\_tree.c**: AVL tree implementation
* **hash\_table.c**: Hash table implementation
* **linked\_list.c**: Linked list implementation
* **test\_files/**: Contains test files

**Resources**

* CLRS: Introduction to Algorithms
* [GeeksforGeeks](https://www.geeksforgeeks.org/)

**10. Conclusion**

In conclusion, our comparative analysis of linked lists and binary search trees revealed trade-offs between simplicity and efficiency. The choice of data structure depends on the specific requirements of the playlist management system, emphasizing the importance of choosing the appropriate data structure for the task at hand.