Note:

The sample input & sample output for the Search Event by Range: 31/12/2023 12:00:00 - 01/01/2024 13:00:00 should not give the output for event no. 2(O/P - 2 - 01/01/2024 13:00:00 - 01/01/2024 13:30:00 - Keynote Address)

Design Document: Event Management System

a. Data Structure Model

For this Event Management System, the chosen data structure is the AVL Tree. An AVL Tree is a self-balancing binary search tree, where the height difference between left and right subtrees (balance factor) of any node is at most 1. Here are the justifications for choosing AVL Tree:

Efficient Searching: AVL Trees provide efficient searching with an average time complexity of O(log n). This is crucial for event management systems where quick retrieval of events is required.

Balanced Structure: AVL Trees automatically balance themselves after insertion or deletion operations, ensuring that the tree remains balanced. This helps in maintaining optimal performance even after multiple modifications.

Ordered Storage: Events can be stored in the AVL Tree based on their start times, allowing for efficient range-based queries and retrieval of events in sorted order.

Support for Range Queries: AVL Trees allow for efficient range queries, making it suitable for searching events within a specified time range.

b. Details of Operations

Insertion (insert function):

Time Complexity: O(log n)

Justification: AVL Trees maintain balance during insertion by performing rotations if necessary. This ensures that the height of the tree remains logarithmic, leading to efficient insertion.

Deletion (delete function):

Time Complexity: O(log n)

Justification: Similar to insertion, AVL Trees maintain balance during deletion. Even after removing a node, the tree remains balanced, leading to efficient deletion.

Searching by ID (search function):

Time Complexity: O(log n)

Justification: AVL Trees support efficient searching by maintaining the binary search tree property. The tree's balanced structure ensures that the search operation traverses only a logarithmic number of nodes.

Searching within a Time Range (search_events_within_range function):

Time Complexity: O(log n + k), where k is the number of events within the specified range.

Justification: By performing an inorder traversal within the specified time range, AVL Trees efficiently retrieve all events falling within that range. The logarithmic time complexity arises from searching for the start and end times of the range, while k represents the number of events found within the range.

Initiate Event Management System (initiateEventManagementSystem function):

This function reads commands from the input file, processes them, and returns the AVL tree containing the events.

Time Complexity: Depends on the number of commands and the size of the input data.

Justification: While the time complexity of this function depends on the number of commands and the input data size, the operations within the function (such as parsing commands, adding events to the AVL tree, etc.) are generally O(log n) due to the AVL tree operations.

c. Alternate Modeling with Cost Implications

An alternate way of modeling the problem could be using a simple sorted list or array to store events ordered by their start times. Here's how it would affect the cost:

Insertion:

Time Complexity: O(n)

Justification: Inserting an event into a sorted list/array requires finding the correct position to maintain the sorted order, which involves shifting elements if necessary. This operation has a linear time complexity, making it less efficient compared to AVL Trees.

Deletion:

Time Complexity: O(n)

Justification: Similar to insertion, deleting an event from a sorted list/array requires finding the event and removing it, which may involve shifting elements. This operation also has a linear time complexity.

Searching by ID:

Time Complexity: O(log n) (for binary search in a sorted list/array)

Justification: Searching for an event by ID in a sorted list/array can be done using binary search, which has a logarithmic time complexity. However, maintaining the sorted order after insertion/deletion operations may offset this advantage.

Searching within a Time Range:

Time Complexity: O(k), where k is the number of events within the specified range.

Justification: While searching within a time range may still be efficient (linear time complexity), the overall performance may degrade due to the lack of balancing. If the list/array is not kept sorted, searching within a time range may require traversing all events, resulting in a linear time complexity.

In summary, although using a simple sorted list/array may seem straightforward, it lacks the efficiency and balance that AVL Trees offer. AVL Trees provide consistent performance for insertion, deletion, and searching operations, making them a better choice for this event management system.