Analysis of Data Structures and Algorithms for our Restaurant Reservation System

For each data structure used, a detailed explanation of the complexity of the structure and the algorithm is provided in the implementation. These details provided here are an overview.

**1. Table Management: Binary Search Tree (BST)**

Data Structure Choice

Binary Search Tree (BST) is chosen to represent table availability based on seating capacity. The reasons behind this choice include:

- Efficient Search Operations: BST provides logarithmic time complexity (O(log n)) for search operations. This is crucial for quickly identifying available tables based on their seating capacity.

- Dynamic Scalability: As the number of tables increases, the BST structure allows for efficient scaling without significant performance degradation.

Operations and Complexity

Insertion:

The insert operation in a BST involves recursively traversing the tree to find the appropriate position for the new node. The time complexity is O(log n) in the average case, ensuring efficient insertion.

Deletion:

Deleting a node from a BST requires finding the node and handling three cases: a node with no children, a node with one child, and a node with two children. The time complexity is O(log n) on average.

Search:

Searching in a BST involves recursively navigating the tree based on the comparison of the target value with the values in the nodes. The time complexity is O(log n) on average.

**2. Reservation Bookings: Hash Table**

Data Structure Choice

Hash Table is employed for managing reservations. The reasons for selecting a hash table are as follows:

- Constant Time Complexity: Hash tables offer constant average time complexity (O(1)) for insertion and retrieval operations.

- Quick Access by Key: Reservations can be directly accessed using the customer's name as a key, ensuring swift retrieval.

Operations and Complexity

* Insertion:
  + Inserting a reservation into the hash table has an average time complexity of O(1).
  + This allows for real-time booking without significant delays.
* Retrieval:
  + Retrieving reservation details based on the customer's name also has an average time complexity of O(1).
  + This ensures quick access to reservation information.

**3. Waitlist Management: Queue**

Data Structure Choice

Queue is chosen for managing the waitlist. The reasons for selecting a queue include:

- FIFO Order: Queue follows a First-In-First-Out order, which aligns with the principle of managing a waitlist based on the order of customer reservation requests.

- Constant Time Complexity for Enqueue and Dequeue: Enqueuing and dequeuing operations in a queue have a constant time complexity of O(1).

Operations and Complexity

* Enqueue:
  + Adding a customer to the waitlist is a constant-time operation (O(1)).
  + This ensures that customers are efficiently added to the waitlist in real-time.
* Dequeue:
  + Removing a customer from the waitlist is also a constant-time operation (O(1)).
  + This facilitates the quick processing of available tables for customers on the waitlist.

4. Occupancy Overview: Data Structures Integration

To provide a comprehensive visual display of the current occupancy status, a combination of data structures is integrated:

- Binary Search Tree (BST): This structure aids in efficiently retrieving tables based on their capacity for display purposes.

- Hash Table: Quick access to reservation details enables the system to display information about reserved tables and customers.

- Queue: The queue is utilized to showcase customers on the waitlist, maintaining the order in which they joined.