**School Management System — Performance Analysis Section**

**Course:** Data Structures and Algorithms

## 1. Overview

This performance report analyzes the efficiency of each module in the School Management System based on the **modules and main program** previously created. The focus is on time complexity, space complexity, and trade-offs for each data structure.

## 2. Module Performance Analysis

### StudentRegistry (HashMap)

* **Operations:** add, remove, lookup students by ID
* **Time Complexity:** O(1) for add, remove, lookup (average case)
* **Space Complexity:** O(n), where n is the number of students
* **Trade-offs:** HashMap allows **fast access**, but does not maintain sorted order. If sorted order was required, a BST could be used, but lookups would be slower (O(log n)).

### CourseScheduler (Queue)

* **Operations:** enqueue students, dequeue for allocation
* **Time Complexity:** O(1) enqueue/dequeue
* **Space Complexity:** O(n), for n students in the queue
* **Trade-offs:** Queue preserves **first-come-first-served** order. Using a list could allow random access but would complicate the fair allocation process.

### FeeTracker (Binary Search Tree)

* **Operations:** add payment, in-order traversal for reports
* **Time Complexity:** O(log n) insert/search, O(n) traversal
* **Space Complexity:** O(n), for n fee records
* **Trade-offs:** BST maintains **sorted payments**, making reporting easier. Using a HashMap would allow faster access but require extra sorting when generating reports.

### LibrarySystem (Stack + HashMap)

* **Operations:** borrow book, return book, check availability
* **Time Complexity:** O(1) for borrow/return/lookup
* **Space Complexity:** O(n), for n books
* **Trade-offs:** Stack keeps a **borrow/return history** in LIFO order. HashMap enables **instant lookup** by ISBN. Using only a list would increase lookup time to O(n).

### PerformanceAnalytics (Heap / Graph)

* **Operations:** add performance record, retrieve top performer, traverse subject dependencies
* **Time Complexity:** Heap insert O(log n), retrieve top O(1); Graph traversal O(V+E)
* **Space Complexity:** O(n + e), n = students, e = edges for dependencies
* **Trade-offs:** Heap allows **efficient top performer detection**. A simple sorted list could also work but would require O(n log n) sorting after every insert. Graph provides subject dependency insights; without it, relationship analysis is limited.

## 3. Notes

* All complexities assume **average case**.
* Space complexity considers the **number of objects stored** in each data structure.
* The chosen data structures are **optimal for the module tasks** while keeping the implementation simple and student-friendly.