# School Management System - Submission Report

Course: Data Structures and Algorithms

Instructor: Dismas Kitaria

Due Date: 24/10/2025

Authors: CT204/119531/24 DENZEL KAMANDE

CT204/119092/24 HELLEN KASYIMA

CT204/119579/24 IAN KIPKORIR

CT/204/120930/24 LIONEL MWANGI

## 1. Abstract

This submission contains a modular Python prototype of a School Management System demonstrating core data structures: hash table, queue, binary search tree, hash map + stack behavior, and heap for analytics. The system includes sample data and a README for running the demo.

## 2. Architecture Overview

The system uses a central controller (main.py) which demonstrates interactions among modules. Each module is implemented as a separate Python file under the "modules" package. Communication occurs via simple method calls and shared IDs (student IDs).

## 3. Module Designs and Data Structure Justification

**Student Registry:** Implemented with a Python dictionary (hash table) for O(1) average lookup, insertion and deletion by student ID.

**Course Scheduler:** Uses a queue (collections.deque) to ensure FIFO allocation of course seats, providing fairness based on registration order.

**Fee Tracker:** A Binary Search Tree stores transactions keyed by (amount, transaction\_id). This enables in-order traversal to produce sorted reports by amount. For production, an AVL or balanced BST should be used to guarantee O(log n) operations.

**Library System:** Inventory is a hash map (dict) mapping ISBN to metadata and copies. Borrow history per student acts like a stack to model "most recent borrowed returned first".

**Performance Analytics:** A max-heap (implemented via negation) quickly extracts top performers. Additional matrix or graph structures can be added for correlations across courses.

## 4. Sample Data & How to Run

Run `python main.py` to execute a demo using provided sample data found in sample\_data.py. The README contains the same instructions.

## 5. Performance Report (summary)

Time complexities (average/worst where applicable):

- Student lookup (hash table): O(1) average, O(n) worst (hash collisions).

- Course enqueue/dequeue (queue): O(1).

- Fee insert/search (BST): O(h) where h is tree height; balanced tree gives O(log n).

- Library operations (dict + stack): O(1) average.

- Top-k extraction (heap): O(k log n) to extract k items; O(log n) per insertion.

## 6. Ethical Reflection

Privacy: The prototype uses simple in-memory storage. For real deployment, encrypt sensitive fields and enforce access controls.

Fairness: Course allocation uses FIFO; consider lottery or priority rules to ensure equity for special cases.

Transparency: Keep logs for allocation and fee operations to allow audits.

## 7. Submission Checklist

- Design document (this file)

- Functional code (folder: SchoolManagementSystem)

- Sample data (sample\_data.py)

- Performance analysis and ethical reflection