Kathmandu University
Department of Computer Science and Engineering
Dhulikhel, Kavre

A Project Report
on
"Student Record Management System"

[Code No : COMP 202]
(For partial fulfillment of II/I Year/Semester in Computer Science/Engineering)

Submitted by

Siddharth Thapa (55)

Submitted to

Mr. Sagar Acharya
Department of Computer Science and Engineering

Submission Date:

24/02/2026

Declaration

I hereby declare that the work presented in this report titled "Student Record Management System" is my own original work carried out independently. I have not copied from any source except where explicitly cited. All sources and references used have been acknowledged.

Name: Siddharth Thapa
Roll Number: 55
Date: 24/02/2026

Acknowledgement

I would like to express my gratitude to Kathmandu University for providing the resources and environment to carry out this project. I would also like to acknowledge our teacher Mr. Sagar Acharya for his time and effort invested while teaching us DSA.

I also thank my family and friends for their constant support and encouragement throughout the development of this project.

Finally, I acknowledge the use of online resources, documentation, and tutorials that helped me understand the concepts of DSA clearly and implement the Student Record Management System using C++ and Qt Creator.

Abstract

Managing student records is a fundamental requirement in any academic institution. Traditional approaches using flat files or simple arrays fail to efficiently handle operations such as insertion, deletion, searching, and sorting as the dataset grows larger.

This project presents a Student Record Management System developed using C++ and the Qt Creator framework, integrating core Data Structures and Algorithms (DSA) concepts. The system employs a Singly Linked List as the primary dynamic storage structure, a Binary Search Tree (BST) for efficient ID-based searching, a Stack for implementing an undo mechanism, and both Bubble Sort and Quick Sort algorithms for flexible data ordering.

The frontend is built using Qt Widgets, providing a clean, interactive graphical user interface with three functional tabs: student management, search and sort, and a DSA visualization panel. The system demonstrates how DSA concepts directly improve performance and usability of a real-world application, achieving efficient record management with clear complexity trade-offs between the implemented algorithms.

Table of Contents

Chapter 1 Introduction

1.1 Background

In academic institutions, managing student records manually or through simple file-based systems becomes increasingly inefficient as the volume of data grows. Operations such as searching for a specific student, maintaining sorted order, or undoing accidental deletions require well-designed data structures to be performed efficiently.

Modern software systems address these challenges using fundamental Data Structures and Algorithms (DSA). Linked lists, trees, stacks, and sorting algorithms each offer distinct performance characteristics suited to different operations. This project harnesses these structures to build a Student Record Management System that is both efficient and interactive.

1.2 Objectives

The specific objectives of this project are:

- To implement a dynamic student record system using a Singly Linked List.
- To enable efficient ID-based searching using a Binary Search Tree (BST).
- To implement an undo feature using a Stack data structure.
- To demonstrate and compare Bubble Sort and Quick Sort algorithms.
- To develop a fully functional Qt-based graphical user interface (GUI).
- To visualize the internal DSA structures through a dedicated panel.

1.3 Motivation and Significance

The motivation behind this project is to bridge theory and practice by:

- Demonstrating how DSA concepts solve real-world problems in software development.
- Providing an interactive tool that lets users observe DSA operations live.
- Comparing algorithm complexities through practical implementation, not just theory.

Chapter 2 Related Works

There are many relevant works done in this sector using DSA concepts and they are as follows:

2.1 Student Information Systems Using Linked Structures

Researchers and educators have proposed linked list-based systems for student data where dynamic memory allocation avoids the fixed-size limitations of arrays. Linked lists allow insertion and deletion in O(1) time at known positions, making them well-suited for record management.

- Dynamic allocation: No pre-defined array size needed.
- Efficient insertion: Add records without shifting elements.
- Traversal: Sequential access across all records in O(n).

2.2 Tree-Based Search in Record Systems

Binary Search Trees have been widely applied in database indexing and record retrieval systems. BSTs organize data hierarchically such that left subtree values are smaller and right subtree values are larger than the root, enabling fast lookups.

- BST Search: O(log n) average vs O(n) for linear search.
- Inorder traversal: Returns records sorted by key automatically.
- Dynamic insertion and deletion maintain sorted order.

2.3 Undo Mechanisms Using Stack

The Stack is a classic LIFO data structure used widely in text editors, IDEs, and management systems to implement undo/redo functionality. Every user action is pushed onto the stack and can be reversed by popping it off.

- Push and Pop: O(1) time complexity for both operations.
- LIFO order ensures the most recent action is undone first.

Chapter 3 Design and Implementation

This system was developed using a modular approach, separating backend DSA logic from the Qt-based frontend GUI. This separation ensures clarity in implementation and maintainability.

3.1 System Design Approach

The development of this system followed the procedure below:

1. Problem Analysis
- Identified limitations of array-based student record systems.
- Defined the need for dynamic insertion, fast search, undo, and sorting.
- Determined requirements for an interactive graphical interface.
2. Data Structure Design
- Singly Linked List for primary record storage.
- Binary Search Tree for ID-based fast searching.
- Stack for undo history tracking.
3. Algorithm Design
- Bubble Sort for alphabetical name sorting and ID sorting.
- Quick Sort for GPA-based descending sort.
4. Implementation
- Backend logic implemented in C++ with header and source file separation.
- Frontend GUI implemented using Qt Widgets with QTableWidget and QTabWidget.
5. Testing and Validation
- Tested all CRUD operations on the linked list.
- Verified BST search returns correct results.
- Validated undo stack reverses actions correctly.

3.2 Software Architecture

The system architecture separates the DSA backend from the Qt GUI frontend. The backend consists of four independent modules: LinkedList, BST, SortingAlgorithms, and UndoStack. The MainWindow class acts as the controller, connecting user interface events to backend operations.

All student data flows through the Linked List as the source of truth. The BST mirrors the linked list data for fast search, and the UndoStack records every mutating operation. Sorting algorithms operate on temporary copies returned from the linked list without modifying the original structure.

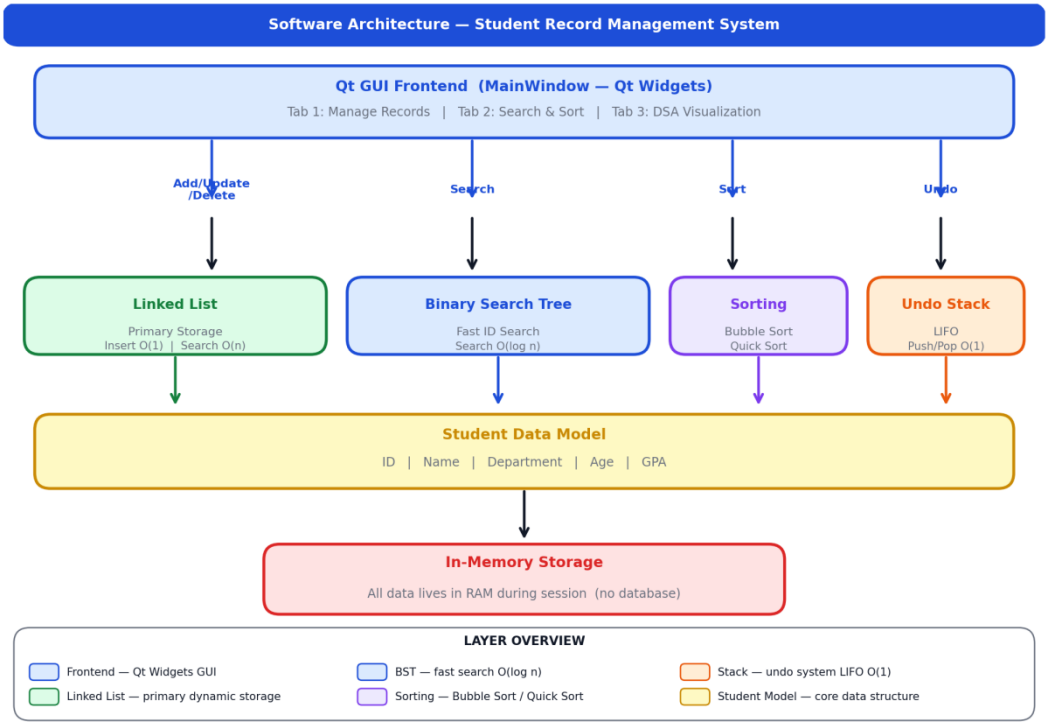

Fig 3.2.1 : Software Architecture of Student Record Management System

3.3 Algorithm Design

3.3.1 Linked List Insert at End

6. Create a new Node with the student data.
7. If list is empty, set head to the new node.
8. Else traverse to the last node and set its next pointer to the new node.
9. Increment the size counter.

3.3.2 BST Search Algorithm

Objective: Find a student by ID in O(log n) average time.

10. Start at the root node.
11. If target ID equals current node ID, return the node.
12. If target ID is less, recurse into the left subtree.
13. If target ID is greater, recurse into the right subtree.
14. If node is null, return null (not found).

3.3.3 Bubble Sort (by Name / ID)

Time Complexity: $O(n^2)$ | Space Complexity: O(1)

15. Iterate outer loop from i = 0 to n-1.
16. Inner loop from j = 0 to n-i-1.
17. If adjacent elements are out of order, swap them.
18. Use swapped flag to stop early if no swaps occur in a pass.

3.3.4 Quick Sort (by GPA)

Time Complexity: O(n log n) average | Space Complexity: O(log n)

19. Choose the last element as the pivot.
20. Partition so elements >= pivot go left (descending order).
21. Recursively apply Quick Sort to left and right partitions.
22. Base case: array of size 0 or 1 is already sorted.

3.3.5 Stack Undo Mechanism

Every user action is recorded as one of three types:

- ADD action: stores added student for deletion on undo.
- DELETE action: stores deleted student for re-insertion on undo.
- UPDATE action: stores old and new data for state restoration on undo.

When undo is triggered, the top action is popped and reversed in O(1) time.

3.4 Flowchart

The following flowchart describes the complete operational flow of the Student Record Management System, covering all six operations: ADD, DELETE, UPDATE, SEARCH, SORT, and UNDO.

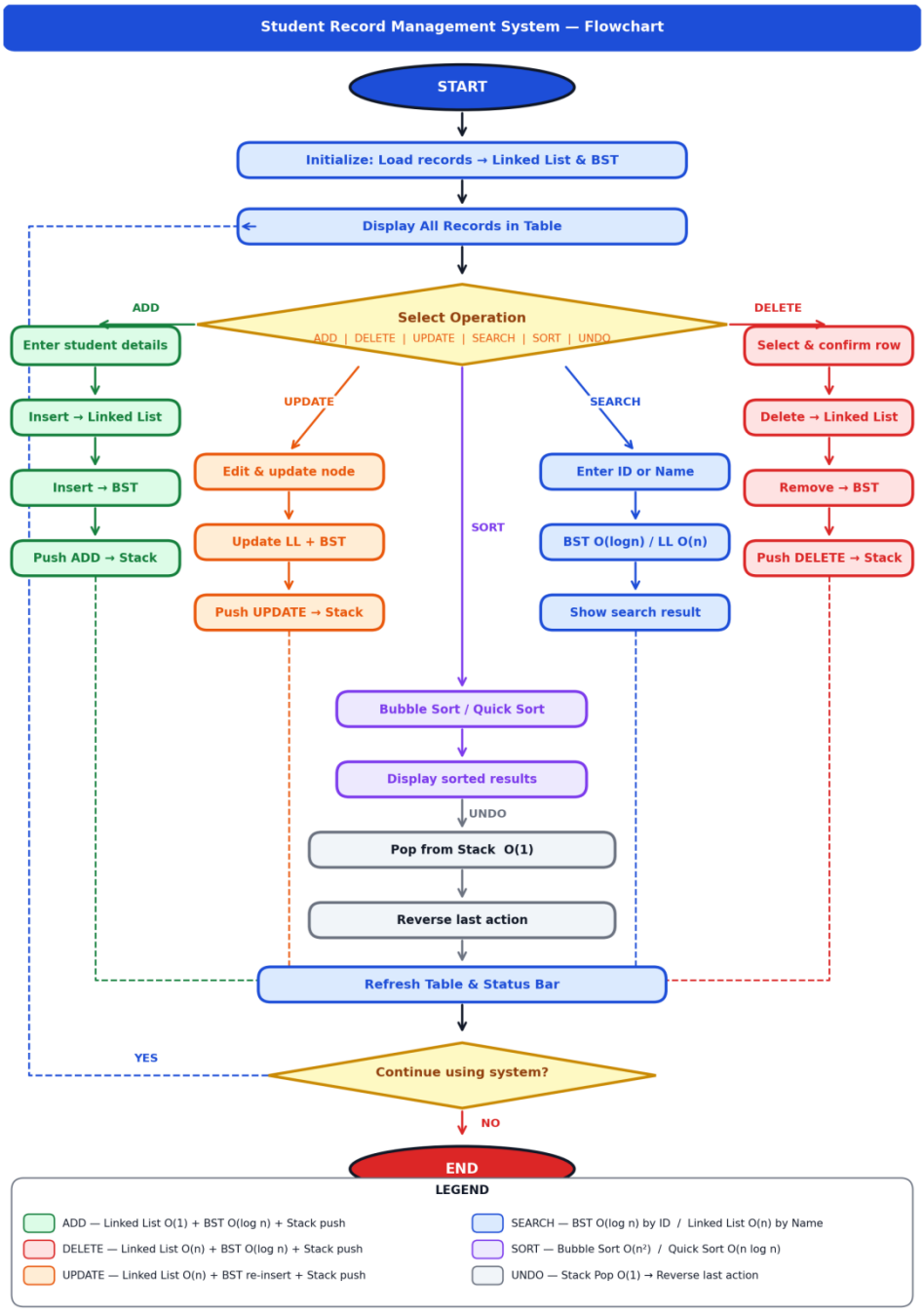

Fig 3.4.1 : Flowchart of Student Record Management System

3.5 Software Specifications

3.5.1 Functional Requirements

The functional requirements of the system are:

- Add, update, and delete student records dynamically.
- Search students by ID using BST (O(log n)) or by name using Linked List (O(n)).
- Sort by Name using Bubble Sort, by GPA using Quick Sort, or by ID using Bubble Sort.
- Undo the last operation using Stack-based undo supporting up to 20 actions.
- Visualize Linked List, BST inorder traversal, and Stack contents in a dedicated tab.

- Pre-load seven sample records on application startup.
- Color-code student rows by GPA level in the table display.

3.5.2 Non-Functional Requirements

The non-functional requirements of the system are:

- Performance: All operations complete in real time with no visible lag.
- Reliability: Data integrity maintained across all CRUD and undo operations.
- Usability: Qt interface is intuitive with labeled buttons, tabs, and status feedback.
- Maintainability: Code is modular with each DSA concept in its own files.

3.5.3 Software Dependencies

The following are the software dependencies:

- C++ Compiler: MinGW (Windows) or GCC (Linux/Mac).
- Qt Framework: Qt 5.x or Qt 6.x with Qt Widgets module.
- Qt Creator IDE: For building and running the project using the .pro file.

3.6 Hardware Specifications

The minimum hardware requirements are:

- Processor: Intel i3 or equivalent.
- RAM: 2 GB minimum, 4 GB recommended.
- Storage: 500 MB free space for Qt installation and build output.
- Display: Any monitor supporting 1280x720 resolution or higher.

Chapter 4 Discussion on the Achievements

The primary objective of this project was to build a functional Student Record Management System that meaningfully integrates DSA concepts into a real graphical application. This project successfully demonstrates dynamic record management using a Singly Linked List, efficient ID-based search using a Binary Search Tree, undo functionality using a Stack, and comparative sorting using Bubble Sort and Quick Sort. The Qt-based frontend provides a clean, interactive experience with real-time feedback on every operation.

4.1 Challenges Faced During Development

In the development process, several challenges were encountered:

- BST and Linked List Synchronization: Keeping both structures consistent during updates required careful sequencing since the BST needed to remove and re-insert on every update.
- Undo for Multiple Action Types: Designing a Stack that handles three action types required a flexible Action struct storing both old and new states.
- Qt Stylesheet Integration: Achieving a visually appealing GUI required careful use of Qt object names and QSS stylesheets without breaking layout behavior.

- GPA Color Coding: Dynamically setting row background colors in QTableWidget required setting backgrounds after each insertion since Qt resets styles on row insertion.

4.2 Deviation from Initial Objectives

Initially, the system aimed to implement file-based persistence so that student records would be saved between sessions. Due to time constraints and the focus on demonstrating DSA concepts clearly, persistence was not implemented. The system currently pre-loads sample data on every launch. A graph-based relationship view between students and departments was also considered but deferred to future work.

4.3 Measurable Results

The measurable progress achieved in this project is listed below:

- All four DSA concepts (Linked List, BST, Stack, Sorting) are fully implemented and functional.
- BST search correctly retrieves student records in O(log n) average time.
- Undo correctly reverses Add, Delete, and Update operations across all test cases.
- Quick Sort correctly sorts students by GPA in descending order, outperforming Bubble Sort on larger datasets.
- The DSA Visualization tab correctly displays Linked List nodes, BST inorder traversal, and Stack state.

4.4 Features

The following are the key features implemented in the system:

23. Dynamic Record Management: Add, update, and delete student records via an interactive form.
24. Linked List Storage: All records stored as nodes in a Singly Linked List with pointer traversal.
25. BST Search: ID-based search using a Binary Search Tree for O(log n) average lookup.
26. Name Search: Linear traversal of the Linked List for name-based partial matching.
27. Bubble Sort: Alphabetical sort by name and ascending sort by ID using Bubble Sort $O(n^2)$.
28. Quick Sort: GPA-based descending sort using Quick Sort O(n log n) average.
29. Undo System: Stack-based undo for the last 20 operations with LIFO reversal.
30. GPA Color Coding: Rows color-coded green (GPA >= 3.5), yellow (GPA >= 2.5), red (GPA < 2.5).
31. DSA Visualization: Text-based visualization of Linked List nodes, BST inorder output, and Stack state.
32. Sample Data: Seven pre-loaded student records for immediate demonstration on launch.

Chapter 5 Conclusion and Recommendation

This project successfully achieves its primary objective of implementing and demonstrating core Data Structures and Algorithms within a real, functional application. The Student Record Management System effectively uses a Singly Linked List for dynamic storage, a Binary Search Tree for fast search, a Stack for undo operations, and Bubble Sort and Quick Sort for data ordering. The Qt framework provided a powerful and flexible platform for building the graphical interface, and the modular code structure ensures clarity and maintainability. The system clearly demonstrates how the choice of data structure and algorithm directly impacts performance and user experience.

5.1 Limitations

Even though this project successfully demonstrates DSA concepts, there are limitations to be addressed:

- No persistent storage: All records are lost when the application is closed.
- No multi-user support: The system runs locally for a single user session.
- BST is unbalanced: In worst case (sorted input), BST degrades to O(n) search.
- No duplicate name handling: The system does not prevent students with identical names.
- Limited scalability: Performance may degrade for very large datasets due to unbalanced BST.

5.2 Future Enhancement

The following enhancements can improve the system:

- File-based or database persistence using SQLite to save records between sessions.
- Implementation of a Self-Balancing BST (AVL Tree) to guarantee O(log n) search in all cases.
- Addition of a Heap data structure for priority-based student ranking by GPA.
- Graph-based department relationship visualization.
- Export to CSV or PDF format for report generation.
- Search history feature using a Queue data structure.
- Multi-field advanced search with combined filters.

References

1. Qt Project. (n.d.). Qt Documentation. https://doc.qt.io/

2. Cppreference contributors. (n.d.). C++ Standard Library Reference. https://en.cppreference.com/w/cpp

3. Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). Introduction to Algorithms (3rd ed.). MIT Press.

4. Sedgewick, R., & Wayne, K. (2011). Algorithms (4th ed.). Addison-Wesley Professional.