Student Record Management System (Merge Sort Algo)

Submitted by

Arshdeep Kaur 25MCI10333

25MAM-5(B)

Submitted To

Dr. Javed Alam (E17984)

Assistant Professor

in partial fulfilment for the award of the degree of
Master of Computer Application

Chandigarh University

Aug 2025 – Nov 2025

ACKNOWLEDGEMENT

I would like to express my sincere gratitude to all those who contributed to the successful completion of this case study on Library Management System. First and foremost, I extend my heartfelt thanks to [Chandigarh University] for providing the resources and academic environment that enabled this research. I am deeply grateful to my mentor/supervisor for their invaluable guidance, constructive feedback, and continuous support throughout the development of this study.

I am also sincerely thankful to my institution and faculty members for providing me with the opportunity to work on this project and for their continuous motivation and support. Their lectures and resources helped me understand core concepts of Python programming, file handling, and database connectivity, which were essential for developing this system.

Finally, I thank my peers, friends, and family for their encouragement and patience during the course of this project. Their support has been instrumental in helping me stay focused and motivated.

This case study is a reflection of collective effort, and I am truly grateful to everyone who played a part in its realization. This project has been a significant learning experience, helping me develop not only technical skills but also problem-solving, logical thinking, and time management abilities.

A special thanks to my friends and classmates for their constant motivation, collaboration, and insightful discussions that helped refine the project further. Finally, I am deeply grateful to my family for their unconditional love, patience, and support during the entire duration of this work.

ARSHDEEP KAUR

UID – 25MCI10333

CHANDIGARH UNIVERSITY

JAVED ALAM

ASSISTANT PROFESSOR

CHANDIGARH UNIVERSITY

INTRODUCTION

In modern data-driven environments, efficient management and processing of records are crucial for organizations, educational institutions, and businesses. A Record Management System (RMS) provides a structured way to store, retrieve, update, and process data efficiently, reducing manual errors and saving time. Traditional methods of record-keeping, such as maintaining paper-based registers, are not only time-consuming but also prone to mismanagement and data loss.

The project incorporates the Merge Sort algorithm for sorting records. Merge Sort is a stable, efficient, and comparison-based sorting algorithm based on the divide-and-conquer approach. It ensures that records with equal marks retain their relative order in the database, a property known as stability, which is essential in real-world record management where the original entry sequence often matters.

The graphical interface built with Tkinter makes the application user-friendly, allowing non-technical users to interact with the system intuitively. Users can easily add new student records, view the existing records, sort them by marks, or reset the database with just a few clicks.

This project demonstrates the integration of GUI-based applications, database management, and efficient algorithms to create a functional and reliable system for managing student records, highlighting the importance of both data structure principles and practical software design in developing real-world applications.

PROBLEM DEFINITION

Managing large sets of student records manually or using basic digital tools can be time-consuming, error-prone, and inefficient. Traditional methods of maintaining student data, such as paper registers or simple spreadsheets, often face several challenges:

1. **Data Organization Issues:** As the number of records grows, it becomes difficult to maintain proper order and quickly retrieve specific information.
2. **Error-Prone Processes:** Manual data entry and sorting may lead to mistakes, duplicates, or loss of records.
3. **Inefficient Sorting:** Sorting student records by marks or other criteria can be slow and unstable using simple methods, making it hard to preserve the original sequence of entries with equal values.
4. **Limited Interactivity:** Non-technical users may find it difficult to navigate or perform operations on records without a graphical interface.
5. **Lack of Data Persistence:** Without a database, records may be lost when the application closes or the system shuts down.

The core problem addressed by this project is to develop a stable, efficient, and user-friendly Record Management System that allows easy storage, retrieval, and sorting of student records while preserving the integrity of the data. By combining a graphical interface (Tkinter), persistent storage (SQLite), and a stable sorting algorithm (Merge Sort), the system aims to simplify record management, reduce errors, and improve efficiency in educational or organizational settings.

OBJECTIVES

The primary objective of this project is to develop an efficient, reliable, and user-friendly Record Management System for managing student records. The specific objectives of the system are as follows:

1. **Efficient Data Storage:** To provide a system that stores student records, including names and marks, in a structured and persistent manner using a database (SQLite).
2. **Stable Sorting of Records:** To implement the Merge Sort algorithm to sort student records by marks in an efficient and stable manner, ensuring that the original order of entries with equal marks is preserved.
3. **User-Friendly Interface:** To design a graphical user interface (GUI) using Tkinter that allows users to easily add, view, sort, and reset records without requiring technical knowledge.
4. **Data Retrieval and Display:** To enable quick and organized viewing of records so that users can easily access and review student information.
5. **Data Management Operations:** To provide functionalities for adding new records, sorting existing records, and resetting/deleting all records to maintain data integrity and flexibility.
6. **Error Handling and Validation:** To ensure the system validates user input, preventing incorrect or incomplete data entry and maintaining accuracy and reliability of the stored information.

LITERATURE REVIEW

A thorough review of existing literature reveals various approaches to record management systems and the importance of efficient sorting algorithms in managing large datasets.

1. **Traditional Record Management Systems:** Traditionally, educational institutions and organizations relied on manual record-keeping, using paper registers to store student data. These systems were prone to human errors**,** data loss, and time-consuming retrieval processes. With the growth in student populations and data volume, the limitations of manual systems became increasingly evident.
2. **Database-Driven Systems:** The advent of Database Management Systems (DBMS) allowed digital storage of records, improving data reliability, persistence, and retrieval speed. Systems using databases like MySQL, SQLite, or PostgreSQL provided structured storage and querying capabilities, which eliminated many errors associated with manual systems.
3. **(GUI) in Record Management:** Integrating GUI frameworks like Tkinter in Python enhances the usability of record management systems, allowing non-technical users to add, view, sort, and delete records intuitively**.** Studies have shown that GUI-based systems improve user efficiency and reduce the learning curve compared to command-line interfaces.
4. **Sorting Algorithms in Data Management:** Sorting is a fundamental operation in data management. Algorithms such as Quick Sort, Bubble Sort, and Merge Sort are widely studied in literature. Merge Sort is particularly noted for its stability and efficiency, especially in applications requiring preservation of relative order of records with equal keys. Several researchers have emphasized that stable sorting is essential in academic record management, where the sequence of entry may carry significance.
5. **Stable Sorting in Real-World Applications:** In educational and administrative software, sorting records by marks, dates, or names is common. Using a stable sorting algorithm like Merge Sort ensures that entries with equal keys maintain their original input order**,** which is critical for fairness and data integrity. Studies in software engineering literature highlight Merge Sort's O(n log n) time complexity and its suitability for database-driven GUI applications**.**

REQUIREMENTS

The development and implementation of the Record Management System involve both hardware and software requirements, as well as functional and non-functional requirements to ensure the system works efficiently and meets user needs.

1. Hardware Requirements

- Processor: Intel i3 or equivalent (or higher)
- RAM: 4 GB or more
- Storage: Minimum 100 MB free disk space
- Display: Minimum resolution 1024 x 768 for GUI display
- Input Devices: Keyboard and mouse

2. Software Requirements

- Operating System: Windows 7/8/10 or Linux/MacOS
- Programming Language: Python 3.x
- GUI Framework: Tkinter (Python built-in library)
- Database Management System: SQLite (lightweight, file-based DB)
- IDE/Editor: Any Python IDE like PyCharm, VS Code, or IDLE

3. Functional Requirements

- Record Addition: Users must be able to add student records, including name and marks.
- Record Viewing: Users should be able to view all existing records in a structured format.
- Sorting Records: The system should allow sorting of records by marks using stable Merge Sort, preserving the order of records with equal marks.
- Reset Records: Users should be able to delete all records from the database with a confirmation prompt.
- Input Validation: The system must validate user input to prevent errors (e.g., marks must be numeric, name cannot be empty).
- Error Handling: Display appropriate error messages for invalid actions or inputs.

4. Non-Functional Requirements

- Performance: Sorting should be efficient, even for large numbers of records (Merge Sort: O(n log n) time complexity).
- Usability: The system should have an intuitive and user-friendly GUI for non-technical users.
- Reliability: Data should be stored persistently in the SQLite database, preventing loss of records.
- Maintainability: The system should be easy to update or extend (e.g., adding new fields or multi-field sorting).
- Portability: The application should run on multiple operating systems that support Python and Tkinter.

RESULT AND VISUALIZATION

The proposed Record Management System was implemented successfully using Python, Tkinter, SQLite, and Merge Sort**.** The system allows users to add, view, sort, and reset student records through a graphical interface. The following results were observed during testing and operation:

1. Adding Records

- Users can enter the name and marks of students through the GUI.
- Input validation ensures that marks are numeric and names are not empty.
- After submission, records are stored persistently in the SQLite database.

2. Viewing Records

- Clicking the "View Records" button fetches all stored data from the database and displays it in the GUI text area.
- The records are displayed in the order they were entered, which helps track the original sequence of entries.

3. Sorting Records

- Records can be sorted by marks using the Merge Sort algorithm.
- Merge Sort ensures stability, i.e., records with equal marks retain their original relative order.

4. Resetting Records

- The "Reset All Records" button deletes all records from the database after a confirmation prompt.
- The GUI output is cleared, and the system confirms the action to the user.

5. Graphical Interface

- Text boxes for data entry (name and marks)
- Buttons for Add, View, Sort, and Reset
- A Text widget to display records
- Color-coded buttons for better usability

6. Performance

- The system handles multiple records efficiently.
- Merge Sort's O(n log n) time complexity ensures fast sorting, even for larger datasets.
- Input validation and error messages improve reliability.

CODE OF THE PROJECT

library sys.py | sortapp.py

C: > Users > ARSHDEEP KAUR > visual code > sortapp.py > merge_sort_records

```
import tkinter as tk
from tkinter import messagebox
import sqlite3
# Database Setup
conn = sqlite3.connect("students.db")
cursor = conn.cursor()
cursor.execute("""
CREATE TABLE IF NOT EXISTS students (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    name TEXT NOT NULL,
    marks INTEGER NOT NULL
)
""")
conn.commit()
```

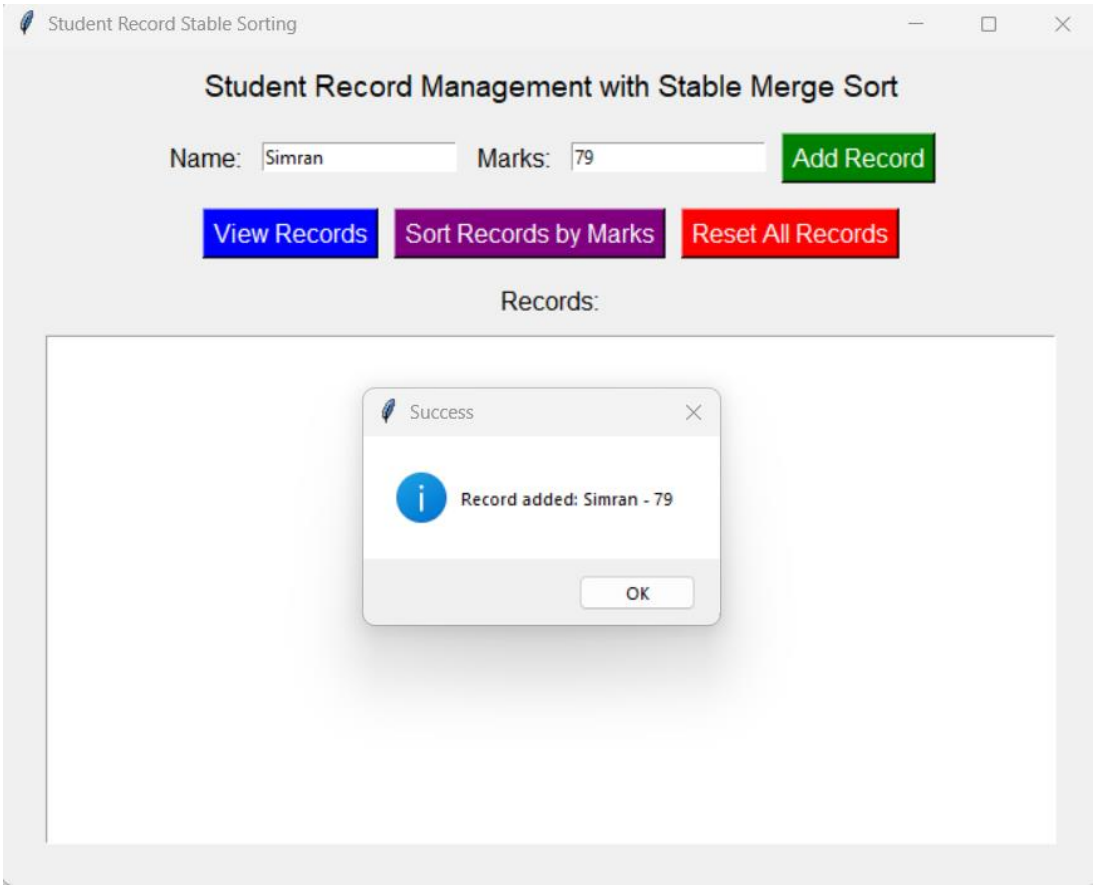
```python
# Merge Sort Implementation
def merge_sort_records(records):
    if len(records) > 1:
        mid = len(records) // 2
        left_half = records[:mid]
        right_half = records[mid:]
        merge_sort_records(left_half)
        merge_sort_records(right_half)
        i = j = k = 0
        while i < len(left_half) and j < len(right_half):
            if left_half[i][1] <= right_half[j][1]:
                records[k] = left_half[i]
                i += 1
            else:
                records[k] = right_half[j]
                j += 1
            k += 1

        while i < len(left_half):
            records[k] = left_half[i]
            i += 1
            k += 1

        while j < len(right_half):
            records[k] = right_half[j]
            j += 1
            k += 1
# GUI Application
class StableSortDBApp:
    def __init__(self, root):
        self.root = root
        self.root.title("Student Record Stable Sorting")
        self.root.geometry("700x550")
        # Heading
        tk.Label(root, text="Student Record Management with Stable Merge Sort", font=("Arial", 14)).pack(pady=10)
        # Add Record Frame
        frame = tk.Frame(root)
        frame.pack(pady=5)
        tk.Label(frame, text="Name:", font=("Arial", 12)).grid(row=0, column=0, padx=5, pady=5)
        tk.Label(frame, text="Marks:", font=("Arial", 12)).grid(row=0, column=2, padx=5, pady=5)
        self.name_entry = tk.Entry(frame)
        self.name_entry.grid(row=0, column=1, padx=5, pady=5)
        self.marks_entry = tk.Entry(frame)
        self.marks_entry.grid(row=0, column=3, padx=5, pady=5)
        tk.Button(frame, text="Add Record", command=self.add_record, bg="green", fg="white", font=("Arial", 12)).grid(row=0, column=4, padx=5)
```

```python
        tk.Button(frame, text="Add Record", command=self.add_record, bg="green", fg="white", font=("Arial", 12)).grid(row=0, column=4, padx=5)
        # Buttons to view, sort, reset
        btn_frame = tk.Frame(root)
        btn_frame.pack(pady=10)
        tk.Button(btn_frame, text="View Records", command=self.view_records, bg="blue", fg="white", font=("Arial", 12)).grid(row=0, column=0, padx=5)
        tk.Button(btn_frame, text="Sort Records by Marks", command=self.sort_records, bg="purple", fg="white", font=("Arial", 12)).grid(row=0, column=1, padx=5)
        tk.Button(btn_frame, text="Reset All Records", command=self.reset_records, bg="red", fg="white", font=("Arial", 12)).grid(row=0, column=2, padx=5)
        # Output Text Box
        tk.Label(root, text="Records:", font=("Arial", 12)).pack(pady=5)
        self.output_text = tk.Text(root, height=20, width=80)
        self.output_text.pack(pady=5)
    # Add record to database
    def add_record(self):
        name = self.name_entry.get().strip()
        marks = self.marks_entry.get().strip()

        if not name or not marks:
            messagebox.showerror("Input Error", "Please enter both name and marks.")
            return

        try:
            marks = int(marks)
        except ValueError:
            messagebox.showerror("Input Error", "Marks must be an integer.")
            return

        cursor.execute("INSERT INTO students (name, marks) VALUES (?, ?)", (name, marks))
        conn.commit()
        messagebox.showinfo("Success", f"Record added: {name} - {marks}")
        self.name_entry.delete(0, tk.END)
        self.marks_entry.delete(0, tk.END)

    # View records from database
    def view_records(self):
        cursor.execute("SELECT name, marks FROM students")
        records = cursor.fetchall()
        self.output_text.delete(1.0, tk.END)
        if records:
            for rec in records:
                self.output_text.insert(tk.END, f"{rec[0]}: {rec[1]}\n")
        else:
            self.output_text.insert(tk.END, "No records found.\n")
```

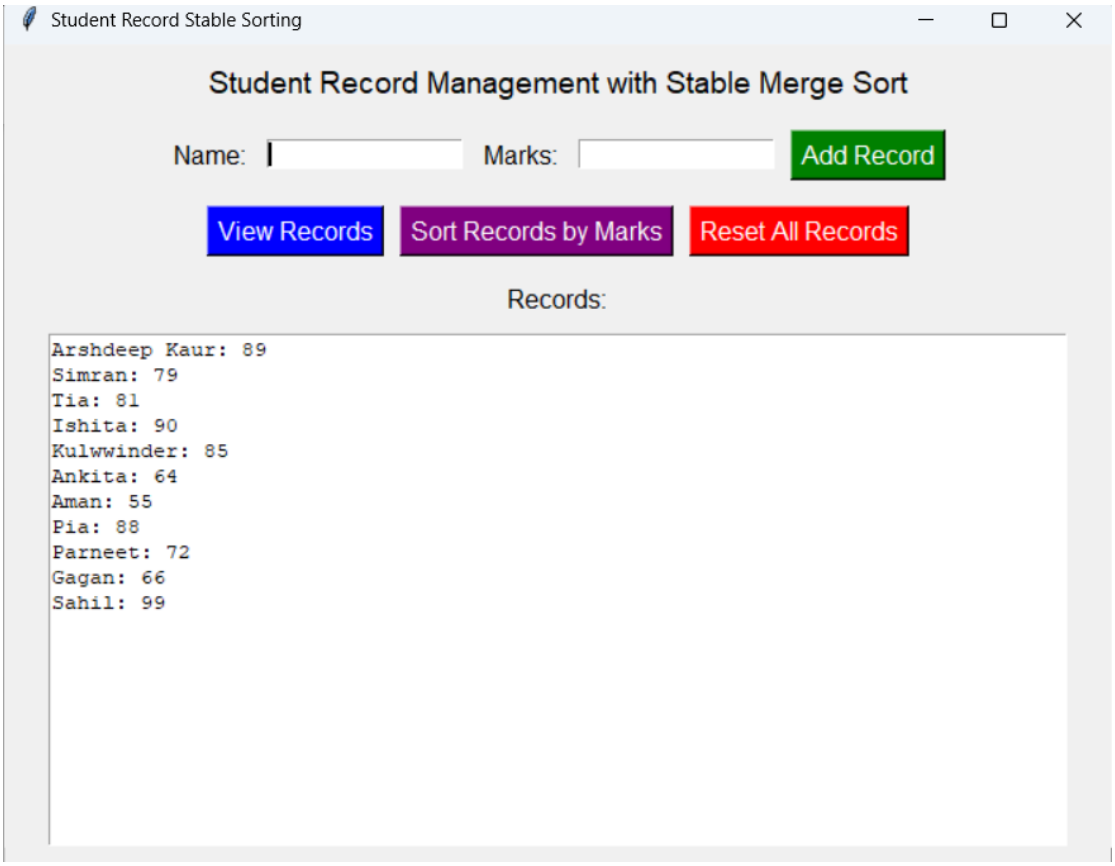
```
            self.output_text.insert(tk.END, "No records found.\n")

    # Sort records by marks using merge sort
    def sort_records(self):
        cursor.execute("SELECT name, marks FROM students")
        records = cursor.fetchall()
        if not records:
            messagebox.showinfo("Info", "No records to sort.")
            return

        records_list = list(records)
        merge_sort_records(records_list)  # sorted in-place

        self.output_text.delete(1.0, tk.END)
        for rec in records_list:
            self.output_text.insert(tk.END, f"{rec[0]}: {rec[1]}\n")

    # Reset all records from database
    def reset_records(self):
        confirm = messagebox.askyesno("Confirm Reset", "Are you sure you want to delete all records?")
        if confirm:
            cursor.execute("DELETE FROM students")
            conn.commit()
            self.output_text.delete(1.0, tk.END)
            self.output_text.insert(tk.END, "All records have been deleted.\n")
            messagebox.showinfo("Reset Complete", "All records have been cleared.")

# ========================
# Run Application
# ========================
if __name__ == "__main__":
    root = tk.Tk()
    app = StableSortDBApp(root)
    root.mainloop()
```

OUTPUT

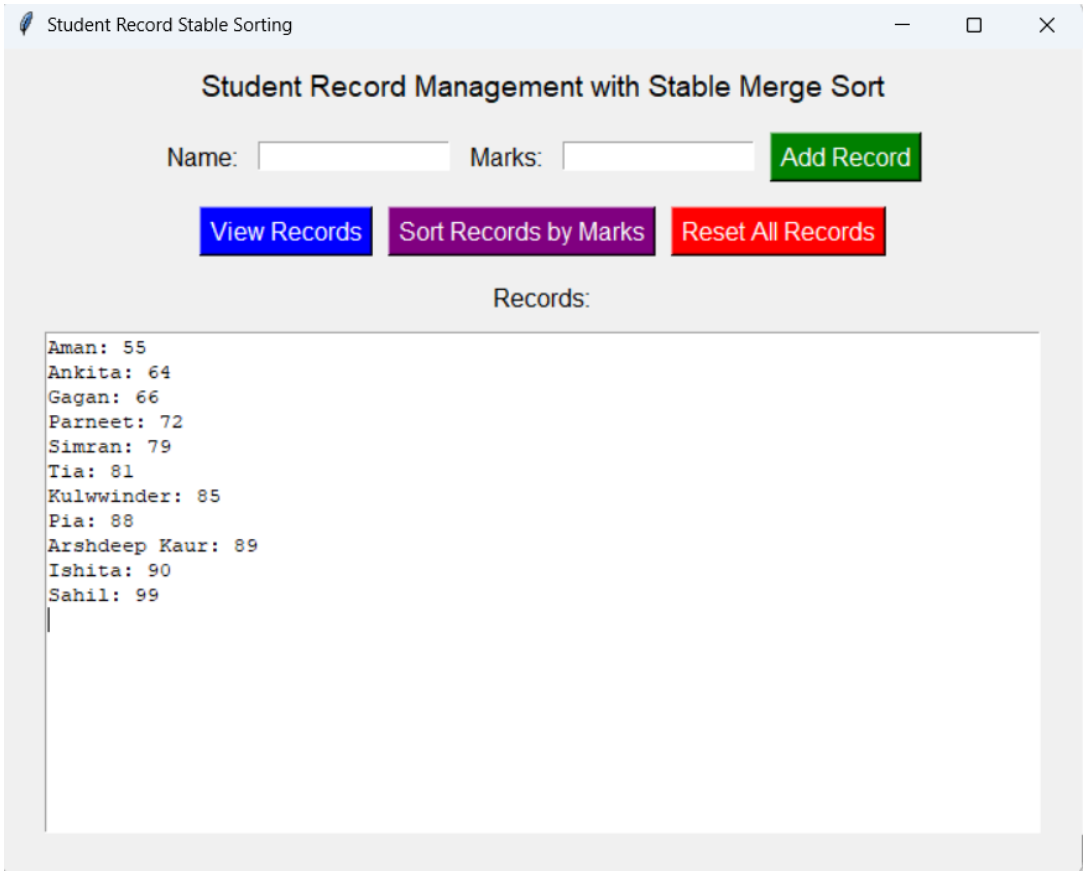
(Add Records)



(View Records)



Sort Records (Using Merge Sort)



CONCLUSION

The Record Management System developed using Python, Tkinter, SQLite, and Merge Sort successfully demonstrates a practical approach to managing student records in an efficient, reliable, and user-friendly manner.

The system addresses the common challenges of manual or basic digital record management, such as data loss, errors, and inefficient sorting, by providing a structured and persistent database-driven solution. The integration of Tkinter GUI ensures ease of use, allowing users to add, view, sort, and reset records with minimal effort, without requiring technical knowledge.

The implementation of the Merge Sort algorithm ensures stable and efficient sorting of records by marks, preserving the relative order of entries with equal values. This stability is crucial in educational and administrative contexts, where the sequence of records can carry significance.

Overall, this project highlights the effective integration of GUI, database management, and algorithms, demonstrating how theoretical computer science concepts like stable sorting can be applied in real-world applications. The system not only provides a functional tool for managing student records but also serves as an educational example of combining Python programming, data structures, and database management to create practical software solutions.

GITHUB LINK

https://github.com/arshdeepk1209-web/Student-Record-Management-using-Merge-Sort-Algo-.git

REFERENCES

1. Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). Introduction to Algorithms (3rd Edition). MIT Press.
2. Sweigart, A. (2015). Automate the Boring Stuff with Python. No Starch Press.
3. Richardson, L., & Ljung, L. (2018). Python GUI Programming with Tkinter. Packt Publishing.
4. SQLite Documentation. (2024). SQLite: SQL Database Engine. Retrieved from https://www.sqlite.org/docs.html
5. Goodrich, M. T., Tamassia, R., & Goldwasser, M. H. (2014**).** Data Structures and Algorithms in Python**.** Wiley.
6. GeeksforGeeks. (2024). Merge Sort Algorithm in Python. Retrieved from https://www.geeksforgeeks.org/merge-sort/