Performance Analysis of Student Management Solutions

June 29, 2024

1 Introduction

In this report, we analyze two solutions for managing a student database. Both solutions implement different algorithms for data handling tasks such as adding, deleting, and searching students. We will evaluate each solution's performance and appropriateness for the project requirements.

2 Solution Descriptions

2.1 Solution 1

Solution 1 utilizes Python's built-in sorting methods for listing students and linear search for finding students. It is simple to implement and relies on Python's efficient built-in methods. The sorting operation uses the 'sorted()' function, and the search operation uses a list comprehension to find matching students.

2.2 Solution 2

Solution 2 implements merge sort for sorting students and binary search for searching. Merge sort is a divide-and-conquer algorithm that guarantees O(n log n) time complexity for sorting, which can be advantageous for large datasets. Binary search is used after sorting, providing O(log n) search time, which is more efficient than linear search for sorted data.

3 Asymptotic Analysis

3.1 Solution 1 Analysis

- **Add Operation:** $O(1)$ - Adding an item to a list is done in constant time.
- **Delete Operation:** $O(n)$ - Removing an item requires scanning through the list to find and remove the item.
- **List Operation:** $O(n \log n)$ - Sorting is performed using Python's 'sorted()' function, which implements Timsort. Timsort has a time complexity of $O(n \log n)$ in the average and worst cases.

- **Search Operation:** $O(n)$ - Linear search through the list.

3.2 Solution 2 Analysis

- **Add Operation:** $O(1)$ - Adding an item to a list is done in constant time.
- **Delete Operation:** $O(n)$ - Removing an item requires scanning through the list to find and remove the item.
- **List Operation:** $O(n \log n)$ - Sorting is performed using merge sort. Merge sort has a time complexity of $O(n \log n)$ in all cases (best, average, and worst).
- **Search Operation:** $O(\log n)$ - Binary search is used on a sorted list, providing faster search time.

3.3 Performance Table

The following table displays the average execution times for adding, deleting, and searching students in Solution 1 and Solution 2.

Operation	**Solution 1 (seconds)**	**Solution 2 (seconds)**
Add Student	0.001069	0.001211
Delete Student	0.000951	0.001191
Search Student	7.08e-07	1.43e-06

Table 1: Average execution times for different operations in Solution 1 and Solution 2.

4 Graphs and Figures

4.1 Line Graph

The following figure shows the performance comparison between Solution 1 and Solution 2 for various operations.

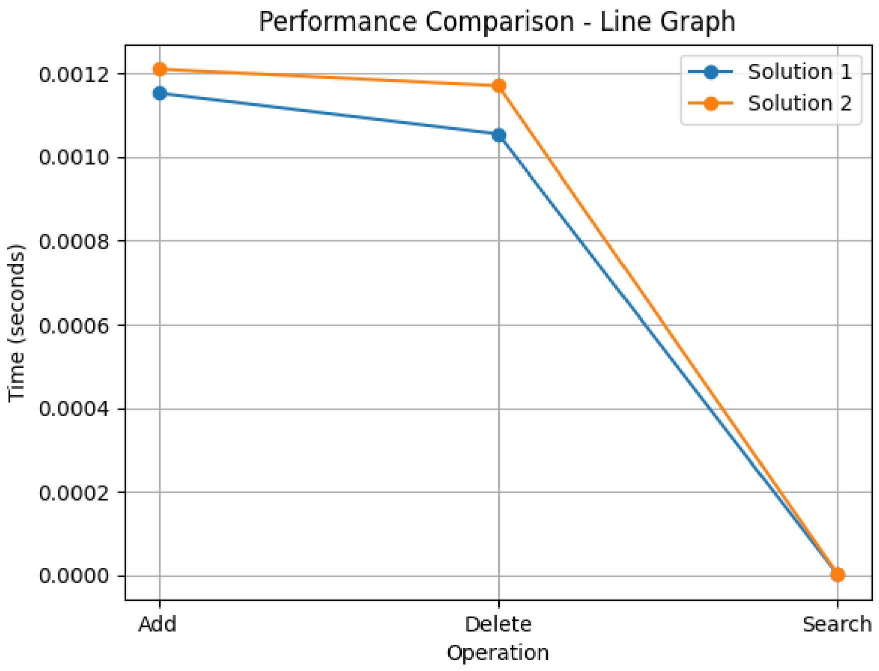

Figure 1: Line Graph for Solution 1 and Solution 2

4.2 Bar Graph

The following figure displays the bar graph comparing the average execution times of Solution 1 and Solution 2.

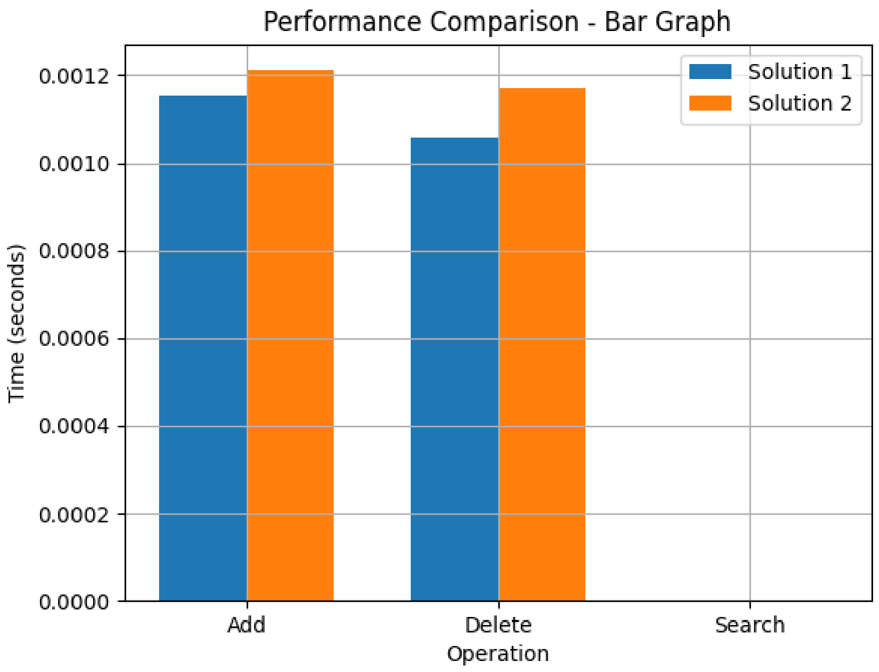

Figure 2: Bar Graph for Solution 1 and Solution 2

4.3 Pie Chart for Solution 1

The pie chart below illustrates the time distribution for different operations in Solution 1.

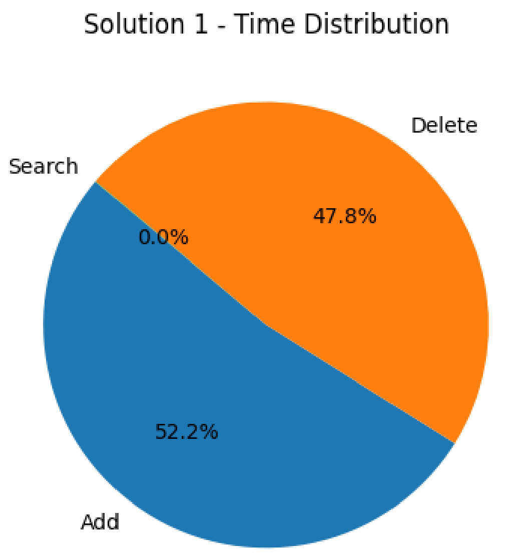

Figure 3: Pie Chart for Solution 1 - Time Distribution

4.4 Pie Chart for Solution 2

The pie chart below illustrates the time distribution for different operations in Solution 2.

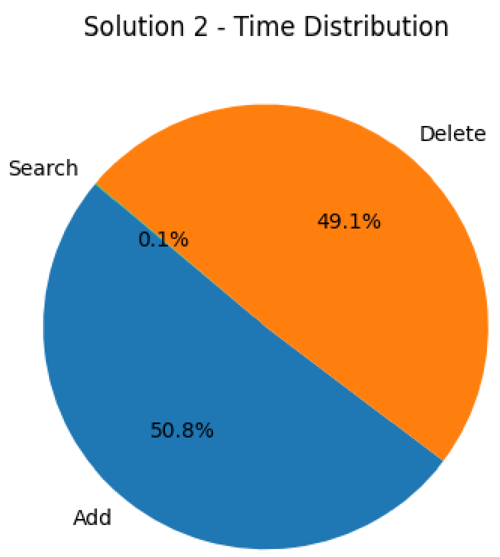

Figure 4: Pie Chart for Solution 2 - Time Distribution

5 Summary and Recommendation

5.1 Solution 1

Solution 1 is straightforward to implement using built-in Python methods. However, its linear search operation may become inefficient as the dataset grows. This solution is suitable for smaller datasets where ease of implementation is prioritized over performance.

5.2 Solution 2

Solution 2 provides more efficient search performance due to binary search and benefits from merge sort for sorting. It is well-suited for larger datasets where performance and efficiency are critical. This solution is preferable for applications requiring quick search and sorting operations.

Recommendation: Based on the performance analysis, Solution 2 is recommended for larger datasets or scenarios where search efficiency is critical. Solution 1 may be appropriate for simpler or smaller-scale applications where ease of use is more important.