Project Title: Performance analysis of Insertion-Merge sort for real time transaction logs

Member [1] Awindrela Roy Dristy (2030226) Member [2] Nur Akib Al Muid (2120272) Member[3] Md. Saikat Khan (2310144) Member [4] Ahanaf Shahriar (2310546) Member [5] Sanjana Rahman (2330807)

Department of Computer Science and Engineering
Independent University, Bangladesh
Dhaka, Bangladesh.
{[1]2030226@iub.edu.bd,[2]2120272@iub.edu.bd,[3]2310144@iub.edu.bd,[4]2310546@iub.edu.bd,[5]2330807@iub.edu.bd

Abstract

This project focuses on improving the efficiency of sorting real-time transaction logs by using a hybrid algorithm that combines Insertion Sort and Merge Sort. Since most transaction data is already sorted with only small updates, our approach adapts by using Insertion Sort for smaller or nearly sorted sections, and Merge Sort for larger parts. We implemented and tested the algorithm using Python and evaluated its performance on various dataset types. The results show that our hybrid method performs significantly faster than standard Merge Sort, especially in real-world scenarios like semi-sorted transaction data.

Introduction

In real-world systems like e-commerce platforms and online banking, transaction logs are constantly updated. These logs typically contain a large amount of sorted data, with small updates inserted in real-time. Traditional sorting algorithms like Merge Sort treat all data the same whether it's sorted or not leading to unnecessary processing. Our project aims to solve this by designing a hybrid sorting algorithm that combines the strengths of Insertion Sort and Merge Sort. This hybrid approach adapts based on the input size and characteristics, making it especially efficient for transaction logs that are nearly sorted.

Rationale for the Algorithm's Selection

In our project, we focused on sorting real-time transaction logs, which are continuously updated with new entries. In most cases, this kind of data is already mostly sorted, with only a few recent transactions inserted out of order.
A standard sorting algorithm like Merge Sort treats all data the same—it doesn't take advantage of the fact that the list might already be almost sorted. That's why we chose a hybrid sorting algorithm that combines Insertion Sort and Merge Sort.
Insertion Sort is simple and works really well on small or nearly sorted data, like real-time logs.
Merge Sort is powerful for large and unordered datasets, offering stability and predictable performance.
By combining the two, our hybrid algorithm adapts to the data:
It uses Insertion Sort for smaller or sorted chunks (less overhead, faster),
And Merge Sort for larger or unsorted parts (ensures efficiency on bigger data).
This makes it a perfect match for our use case, as it balances speed, memory usage, and adaptability for sorting logs that change frequently but aren't fully disorganized.

Methodology

Our hybrid sorting algorithm is applied to real-time transaction logs, where most of the data is already sorted, and only a few new entries are inserted during updates. Instead of re-sorting the entire dataset every time, the algorithm adapts based on the size and order of the data.
It checks whether the current portion of the data is small (e.g., 32 elements):

- If yes, it uses Insertion Sort for quick, efficient sorting.
-
- If no, it applies Merge Sort recursively to maintain overall structure.

This makes it perfect for time-sensitive systems like online banking and e-commerce, where fast, adaptive sorting is crucial.

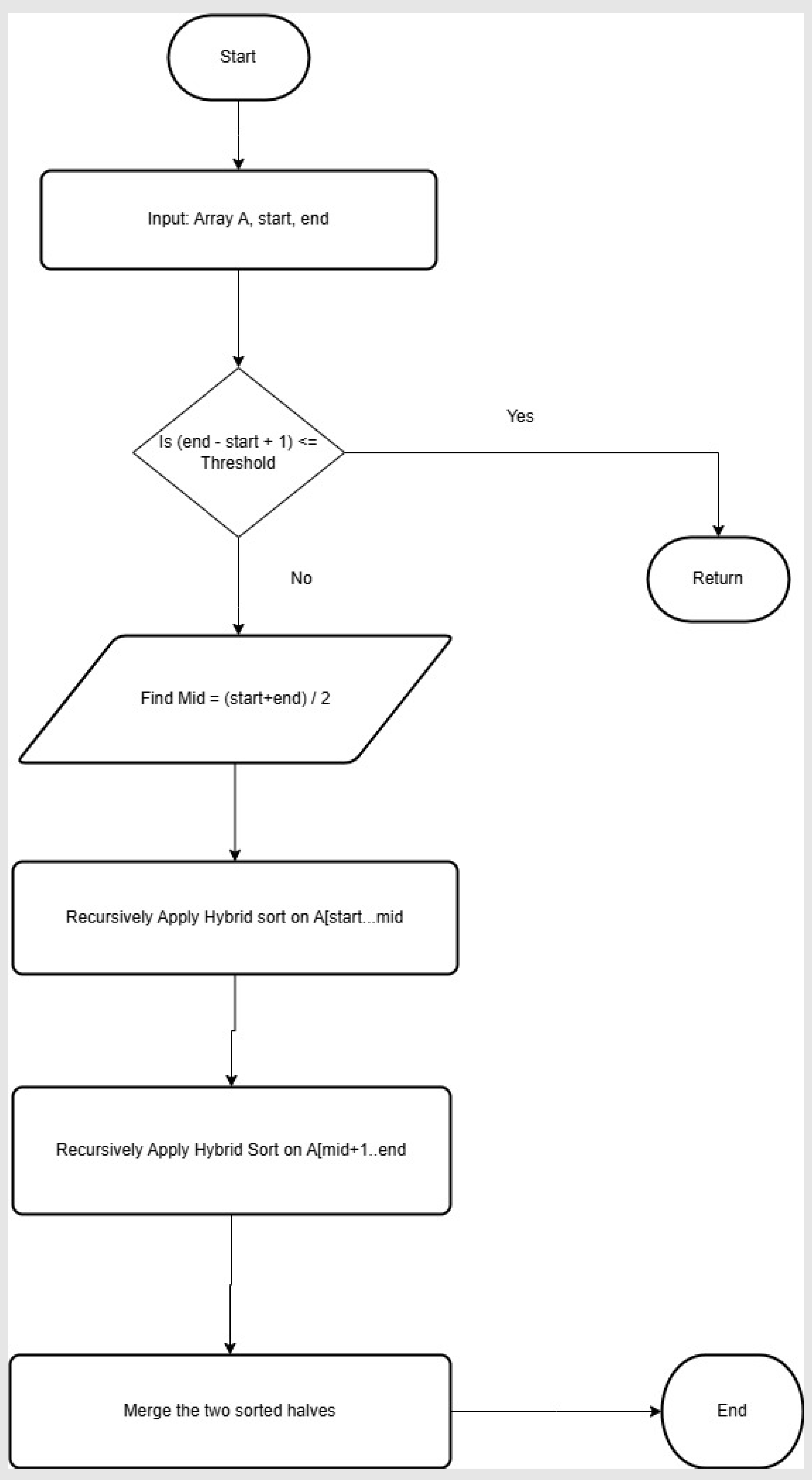
Start
Input: Array A, start, end
Is (end - start + 1) <= Threshold
Yes
Return
No
Find Mid = (start+end) / 2
Recursively Apply Hybrid sort on A[start...mid
Recursively Apply Hybrid Sort on A[mid+1..end
Merge the two sorted halves
End

Figure 1. Enter Caption

Substitute Use of the Algorithm

Yes, the hybrid sorting algorithm we used for real-time transaction logs can definitely be applied to other real-world scenarios, especially those involving semi-sorted or frequently updated data.
Reason: Because the algorithm is:

- Adaptive – It detects small sorted sections and avoids over-sorting
- Efficient – It reduces time and memory usage in dynamic environments
- Versatile – It works on both small and large datasets with consistent performance

It can be applied below:

- Real-Time Dashboards — to keep logs and data streams sorted on the fly
- In-Memory Databases — for fast, adaptive sorting without reprocessing everything
- Chat Systems / Feeds — to maintain order of messages as they arrive
- File Management Tools — for syncing or sorting files dynamically
- IoT Sensor Logs — to sort time-series data efficiently as it's collected

Alternative Solution

Yes. While our hybrid algorithm performs well, it can be replaced or enhanced by more advanced adaptive sorting techniques. Alternative Algorithms:

- Timsort – A real-world optimized hybrid used in Python, better for large, semi-sorted data.
- IntroSort – Combines QuickSort, HeapSort, and Insertion Sort to avoid worst-case performance

Possible Improvements:

1. Dynamic Thresholds – Adjust based on dataset behavior
2. Parallel Sorting – Use multiple threads to boost performance
3. Memory Optimization – Reduce extra space during merging

Conclusion

Our project successfully demonstrates that combining Insertion Sort and Merge Sort into a hybrid solution provides better performance for real-time data like transaction logs. This approach balances speed and stability and is especially suited for systems with continuous updates.
With further development, this algorithm could be optimized for even larger-scale applications, including real-time databases and live analytics dashboards.