UNIVERSITI
TEKNOLOGI
PETRONAS

TFB2023/TEB1113:Algorithm and Data Structure - May 2025

Bubble Sort vs Merge Sort: A Comparative Analysis

Lecturer's Name: Shashi Bhushan

NO.	NAME	STUDENT ID	PROGRAM
1.	MUHAMMAD AIMAN ZAKWAN BIN MASRI	24003466	COMPUTER ENGINEERING
2.	MUHAMMAD IDRIS BIN MOHD YUSRI	24003318	COMPUTER ENGINEERING
3.	EMIR AZIMIL AKBAR BIN FAUZI	24003510	COMPUTER ENGINEERING
4.	MUHAMMAD HARITH ISKANDAR BIN MAHATHIR	24003426	COMPUTER ENGINEERING

A. Introduction

In this project, we compare two popular sorting algorithms: Bubble Sort and Merge Sort. Bubble Sort is known for its simplicity and intuitive design, making it a common introductory algorithm for beginners. In contrast, Merge Sort is a more advanced algorithm that efficiently handles large datasets through a divide-and-conquer approach. Understanding the differences between these two algorithms is essential for choosing the right tool based on the requirements of efficiency, memory usage, and scalability.

B. Algorithm Explanation

1. Bubble Sort

Working Principle: Bubble Sort repeatedly steps through the list, compares adjacent elements, and swaps them if they are in the wrong order. This process continues until no more swaps are needed.

Step-by-Step Example (Input: [5, 2, 9, 1]):
- Pass 1: [2, 5, 1, 9]
- Pass 2: [2, 1, 5, 9]
- Pass 3: [1, 2, 5, 9] (Sorted)

Time Complexity:
- Best Case: $O(n)$
- Average Case: $O(n^2)$
- Worst Case: $O(n^2)$

Space Complexity: $O(1)$

Pros:
- Easy to understand and implement
- Requires no extra memory

Cons:
- Very inefficient for large datasets
- Slow due to repetitive swapping

2. Merge Sort

Working Principle:

Merge Sort follows the divide-and-conquer strategy. It recursively divides the list into halves until each sublist contains a single element, then merges them in sorted order.

Step-by-Step Example (Input: [5, 2, 9, 1]):

- Split: [5, 2] and [9, 1]
- Split further: [5], [2], [9], [1]
- Merge step 1: [2, 5], [1, 9]
- Merge step 2: [1, 2, 5, 9] (Sorted)

Time Complexity:

- Best Case: O(n log n)
- Average Case: O(n log n)
- Worst Case: O(n log n)

Space Complexity: O(n)

Pros:

- Very efficient on large datasets
- Stable and consistent performance

Cons:

- Requires additional memory
- Slightly complex to implement

C. Side-by-Side Comparison Table

Criteria	Bubble Sort	Merge Sort
Working Principle	Repeatedly swaps adjacent elements	Divides list and merges sorted sublists
Time Complexity	$O(n^2)$	O(n log n)
Space Complexity	O(1)	O(n)
Number of Steps	High (especially for n > 10)	Moderate
Best Use Cases	Small datasets, educational use	Large datasets, real-world applications

D. Use Case Comparison

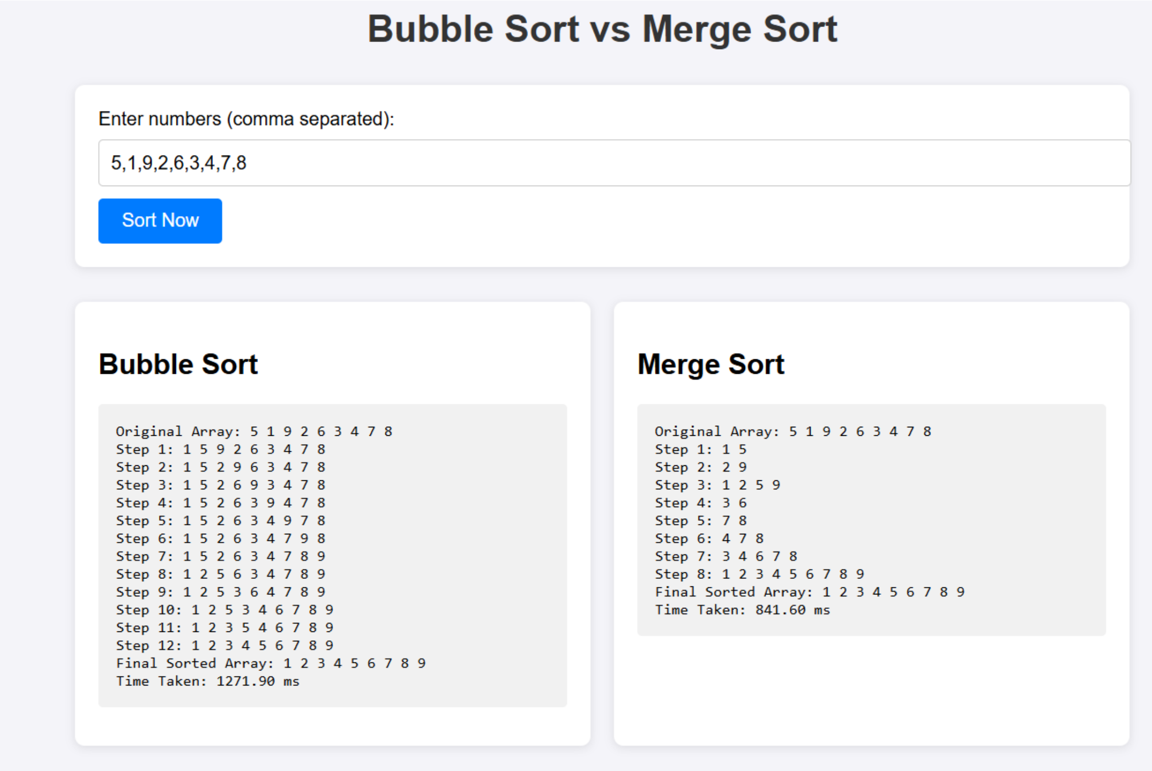
Scenario 1: Sorting a small list of student grades (e.g., < 10 items)

- Better Algorithm: Bubble Sort
- Reason: Easier to implement with minimal overhead. Fast enough for small size.

Scenario 2: Sorting large e-commerce product listings (>1000 items)

- Better Algorithm: Merge Sort
- Reason: Merge Sort performs consistently regardless of initial order and scales efficiently.

E. Visual Aid



Bubble Sort vs Merge Sort

Enter numbers (comma separated):

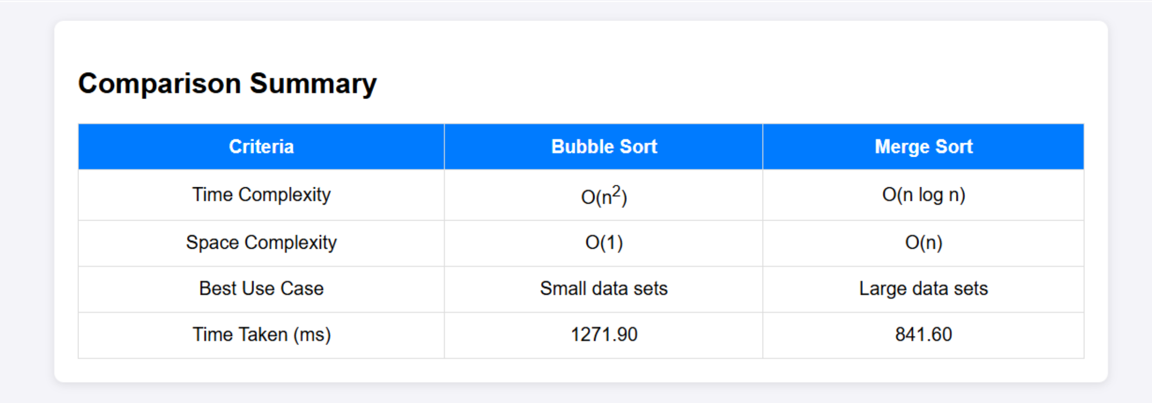
5,1,9,2,6,3,4,7,8

Bubble Sort

```
Original Array: 5 1 9 2 6 3 4 7 8
Step 1: 1 5 9 2 6 3 4 7 8
Step 2: 1 5 2 9 6 3 4 7 8
Step 3: 1 5 2 6 9 3 4 7 8
Step 4: 1 5 2 6 3 9 4 7 8
Step 5: 1 5 2 6 3 4 9 7 8
Step 6: 1 5 2 6 3 4 7 9 8
Step 7: 1 5 2 6 3 4 7 8 9
Step 8: 1 2 5 6 3 4 7 8 9
Step 9: 1 2 5 3 6 4 7 8 9
Step 10: 1 2 5 3 4 6 7 8 9
Step 11: 1 2 3 5 4 6 7 8 9
Step 12: 1 2 3 4 5 6 7 8 9
Final Sorted Array: 1 2 3 4 5 6 7 8 9
Time Taken: 1271.90 ms
```

Merge Sort

```
Original Array: 5 1 9 2 6 3 4 7 8
Step 1: 1 5
Step 2: 2 9
Step 3: 1 2 5 9
Step 4: 3 6
Step 5: 7 8
Step 6: 4 7 8
Step 7: 3 4 6 7 8
Step 8: 1 2 3 4 5 6 7 8 9
Final Sorted Array: 1 2 3 4 5 6 7 8 9
Time Taken: 841.60 ms
```



Comparison Summary

Criteria	Bubble Sort	Merge Sort
Time Complexity	$O(n^2)$	O(n log n)
Space Complexity	O(1)	O(n)
Best Use Case	Small data sets	Large data sets
Time Taken (ms)	1271.90	841.60

F. Conclusion

In conclusion, while Bubble Sort is simpler and useful for understanding basic sorting logic, it becomes impractical for larger datasets. Merge Sort, although more complex, is significantly more efficient in terms of both time and scalability. In real-world applications where performance matters, Merge Sort is almost always the preferred choice.

GitHub Repository

https://github.com/hritthh/TEB1113_TFB2023_DSA.git