UNIVERSITI
TEKNOLOGI
PETRONAS

May Semester 2025

Course: TFB2023/TEB1113 (Algorithm and Data Structure)

Lecturer: Dr. Shashi Bhushan

Project Title: Sorting Algorithm Simulator & Comparator

Group Details: Group 1

Name	Student ID
Cheah Suet May	22010879
Loo Hong Sheng	22011393
Haridvarna Raman A/L Vathumalai	22011017
Rithish A/L Hari	22011356

GitHub: GitHub

Introduction

In this project, there will be two sorting algorithms presented in the form of desktop app. The sorting algorithms that are being compared are Bubble Sort and Merge Sort. These two sorting algorithms stand out as two of the most well-known algorithms, each representing their respective time complexity, space complexity, and performance spectrum. Bubble Sort is known for its simplicity and is often introduced early in programming education, while Merge Sort is recognized for its computation efficiency and widespread use in real-world applications. Comparing these two algorithms are important and interesting because it highlights the differences in terms of advantages and function such as whether it is a simple or complex implementation. It also provides valuable insights into algorithm design principles, such as brute-force versus divide-and-conquer methods, and helps beginners and developers understand when to use one algorithm over the other depending on the context, data size, and performance requirements.

Algorithm Explanation

Bubble Sort and Merge Sort are two basic sorting algorithms used to arrange data in a specific order, typically ascending or descending. Bubble Sort is an algorithm that shows simple comparison by repeatedly swapping adjacent elements if they are in the wrong order. Then, the largest elements will "bubble up" to the end of the list.

Although Bubble Sort is not efficient for large datasets due to its $O(n^2)$ time complexity in both average and worst-case scenarios, it performs slightly better (O(n)) when the data is already sorted. It also uses only O(1) space, as it sorts the data in-place without needing extra memory. In contrast, Merge Sort consistently offers O(n log n) time complexity across best, average, and worst cases due to its divide-and-conquer method. However, this efficiency comes at the cost of O(n) space complexity, since it requires additional memory to store temporary subarrays during the merging process.

Bubble Sort advantages are simplicity, minimal memory usage due to in-place sorting, and easy implementation, making it ideal for beginners and educational purposes. Bubble Sort is not efficient for large datasets due to its $O(n^2)$ time complexity, it is best used to teach basic algorithmic thinking, control structures, and time complexity analysis. On the other hand, Merge Sort is an efficient, stable sorting algorithm that uses a divide-and-conquer method. It recursively divides the dataset into smaller parts, sorts them, and merges them back together in order. Merge Sort is more complex to implement but

provides consistent O(n log n) performance, making it highly suitable for real-world applications such as file systems, databases, and large-scale data processing. It performs well on both arrays and linked lists and is particularly beneficial in situations where sorting stability and predictable performance are prioritized. However, Merge Sort requires additional memory for merging which is often acceptable in modern systems.

Demonstration

Input: 5 , 7 , 2 , 0

Bubble Sort

Step 1: Compare 5 & 7 → No swap → [5, 7, 2, 0]

Step 2: Compare 7 & 2 → Swap → [5, 2, 7, 0]

Step 3: Compare 7 & 0 → Swap → [5, 2, 0, 7]

Step 4: Compare 5 & 2 → Swap → [2, 5, 0, 7]

Step 5: Compare 5 & 0 → Swap → [2, 0, 5, 7]

Step 6: Compare 5 & 7 → No swap → [2, 0, 5, 7]

Step 7: Compare 2 & 0 → Swap → [0, 2, 5, 7]

Step 8: Compare 2 & 5 → No swap → [0, 2, 5, 7]

Step 9: Compare 5 & 7 → No swap → [0, 2, 5, 7]

Final output: 0 , 2 , 5 , 7 (Sorted)

Total Steps: 9

Total Swaps: 5

Time Complexity: $O(n^2)$

Input: 5 , 7 , 2 , 0

Merge Sort

Step 1: Divide (5,7) & (2, 0)

Step 2: Divide (5) , (7) , (2) , (0)

Step 3: Merge (5) and (7) → (5, 7)

Step 4: Merge (2) and (0) → (0, 2)

Step 5: Merge (5, 7) and (0, 2)

Final Output: 0 , 2 , 5 , 7 (Sorted)

Time Complexity: O(n log n)

Space Complexity: O(n)

Side-by-Side Comparison Table

Criteria	Bubble Sort	Merge Sort
Working Principle	Repeatedly stepping through the list, comparing adjacent elements, and swapping them if they are in the wrong order. This process is repeated until the list is sorted. The largest elements "bubble up" to the end of the list after each pass.	A divide and conquer algorithm. Divides the array into half until each sub-array has one element, and then merges them back together in sorted order. The merging step ensures the resulting array is always sorted.
Time Complexity	Worst: $O(n^2)$ Average: $O(n^2)$ For worst and average case, every element is compared with every other element. Performance degrades quickly with size. Best: O(n) For best case, sorted list and only one pass with no swaps.	Worst: O(n log n) Average: O(n log n) Best: O(n log n) For worst, average, and best case, performance is consistent regardless of initial order. Dividing the array takes log n steps and merging takes O(n), hence O(n log n). Efficient for large datasets.
Space Complexity	O(1) — In-place sorting No additional memory required other than a few variables for swapping. Good for memory-limited environments.	O(n) Requires additional space for temporary arrays during merging. Not in-place, so memory usage increases with input size.
Number of Steps	High for large datasets because each element is compared multiple times across multiple passes.	Fewer steps due to divide-and-conquer, even for large datasets.
Best Use Cases	- Small datasets. - Demonstrate basics of sorting. - Simplicity and minimal memory.	- Large datasets. - Stable sorting. - External sorting (When data is unable to fit in RAM)

Use Case Comparison

Scenario 1: Sorting Web Server Logs

Best Algorithm: Merge Sort

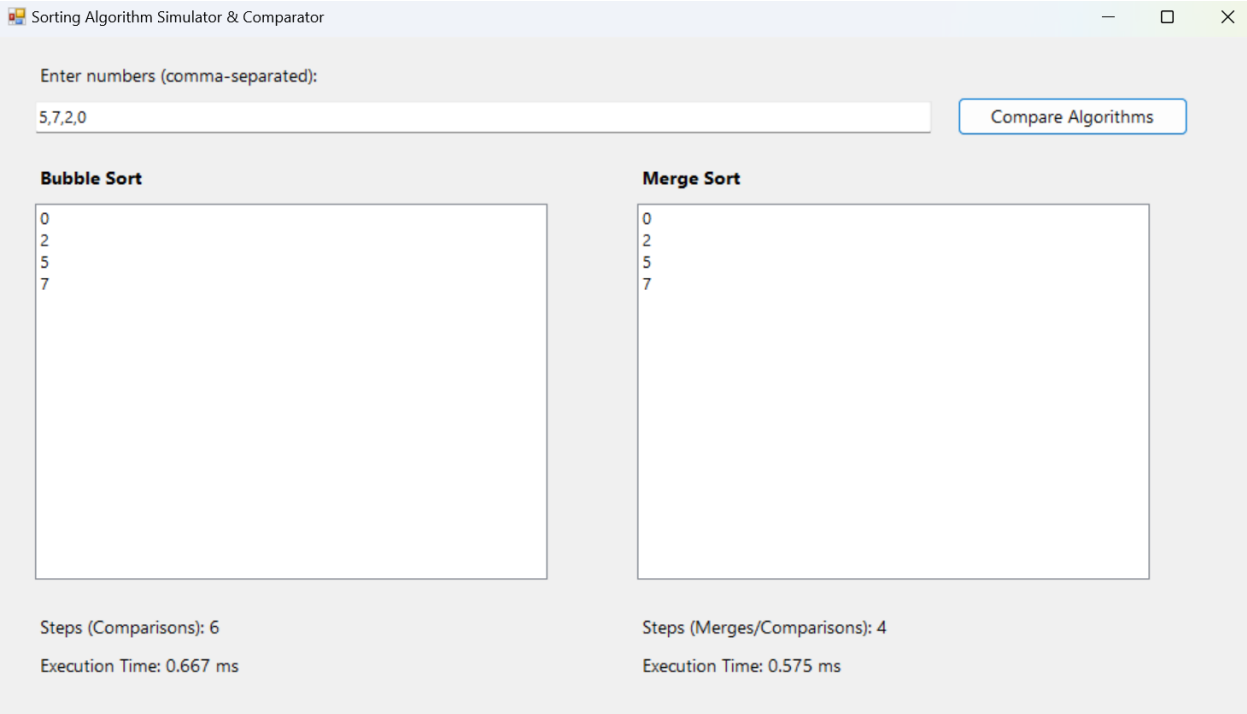
When dealing with very large datasets like web server logs or database entries, Merge Sort is far more suitable. Its consistent O(n log n) time complexity and ability to handle external sorting by using disk rather than RAM, make it ideal for sorting data that doesn't fit entirely in memory. Additionally, its stability ensures that records with equal keys maintain their original order, which is crucial in log analysis or time-sensitive data processing. Thus, Bubble Sort would be inconvenient in this case due to its $O(n^2)$ time complexity and high number of unnecessary comparisons and swaps.

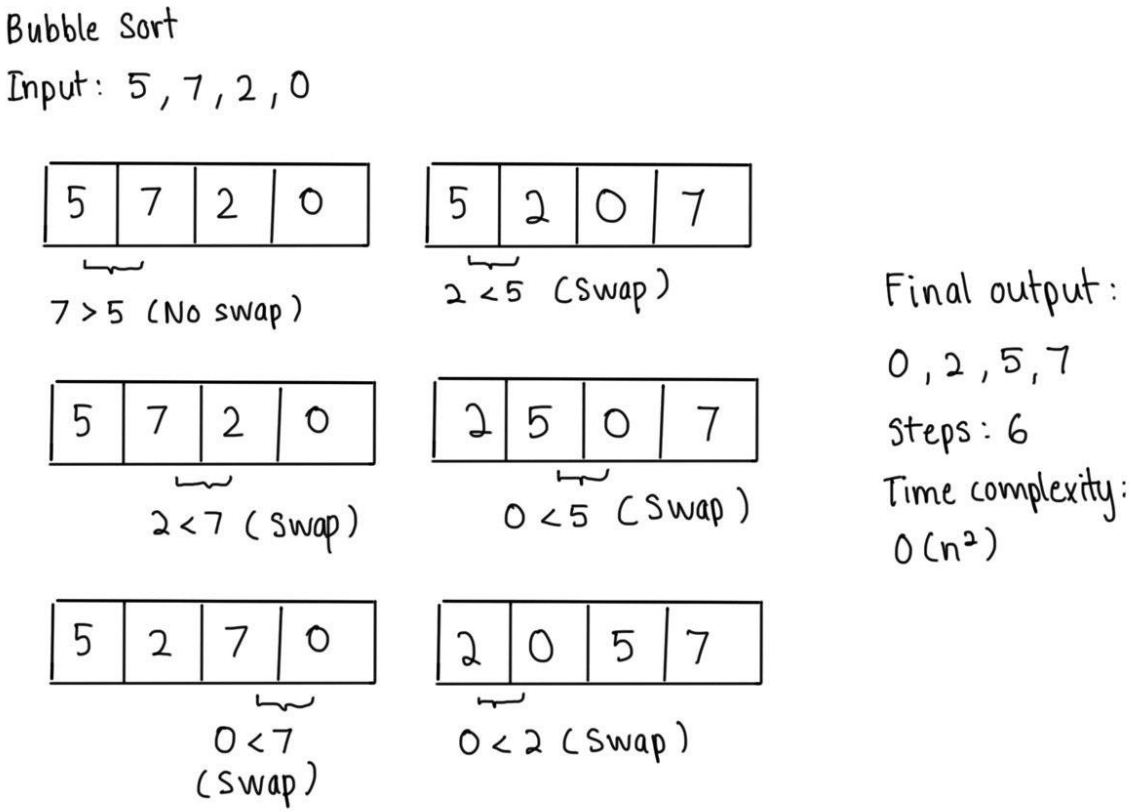
Scenario 2: Teaching Sorting Logic for Beginners' Level

Best Algorithm: Bubble Sort

Bubble Sort is commonly used for educational purposes, particularly in introductory programming courses. Bubble Sort is often chosen as the first sorting algorithm students learn because it is easy to understand due to its straightforward logic. It provides a clear demonstration of basic programming concepts such as nested loops, value comparisons, and swapping operations. Its step-by-step process makes it ideal for manual tracing, visual aids, and helping learners grasp the concepts of sorting before moving on to more efficient algorithms. Bubble Sort's simplicity and clarity make it a valuable tool for teaching algorithmic thinking and problem-solving. Hence, Merge Sort would be difficult for beginners to grasp because it relies heavily on recursion especially when it involves multiple levels of function calls and splitting arrays.

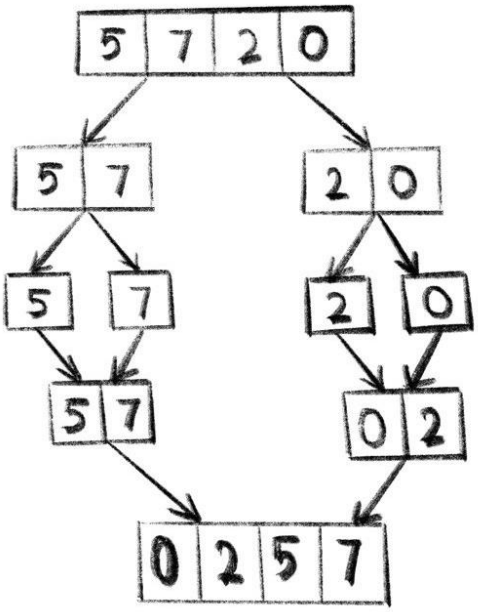
Visual Aid


Sorting Algorithm Simulator & Comparator
Enter numbers (comma-separated):
5,7,2,0
Compare Algorithms
Bubble Sort
0
2
5
7
Merge Sort
0
2
5
7
Steps (Comparisons): 6
Execution Time: 0.667 ms
Steps (Merges/Comparisons): 4
Execution Time: 0.575 ms


Bubble Sort
Input: 5, 7, 2, 0
5 7 2 0
7 > 5 (No swap)
5 2 0 7
2 < 5 (Swap)
5 7 2 0
2 < 7 (Swap)
2 5 0 7
0 < 5 (Swap)
5 2 7 0
0 < 7 (Swap)
2 0 5 7
0 < 2 (Swap)
Final output:
0, 2, 5, 7
steps: 6
Time complexity:
O(n²)

Merge Sort

Input : 5, 7, 2, 0



Final output : 0, 2, 5, 7

Steps : 4

Time complexity :

$O(n \log n)$

Conclusion

To summarize, Merge Sort has proven to be more efficient than Bubble Sort in terms of real-world applications. Merge Sort consistently performs well and faster with a time complexity of O(n log n), making it ideal for large datasets. It is also a stable sorting algorithm that maintains the relative order of equal elements which is an important feature in certain applications. However, Merge Sort is more complex and requires additional memory space due to merging. On the other hand, Bubble Sort is a simple algorithm that is easy to implement and understand. It operates in-place and has minimal memory requirements, but its performance is poor, especially on large or unsorted datasets, due to its $O(n^2)$ time complexity. Therefore, Bubble Sort is best suited for educational purposes or for sorting very small, sorted arrays. Overall, Merge Sort is best used for large datasets, applications that requires stable sorting, and external or file-based sorting. Bubble Sort could be used for small datasets and teaching basic sorting concepts. Based on personal opinions, Bubble Sort might be simpler but in some situations, it is not always better. It is definitely easy to understand and implement but its inefficiency makes it unusable for certain applications. In contrast, Merge Sort that is known to be more complex, demonstrates how well-designed algorithms handle scaling and performance under pressure. Merge Sort shows how divide-and-conquer works clearly. Despite using extra memory, it handles large volumes of data gracefully and predictably. Bubble Sort is equivalent to trial and error method while Merge Sort represents divide and conquer which leads to Merge Sort being a strategic and structural approach with better outcomes. In general, Merge Sort is utilised in applications because it handles large datasets efficiently, and is stable for performance and reliability while Bubble Sort is a great teaching tool for comparison and swapping work in sorting.