T.C. YEDITEPE UNIVERSITY
FACULTY OF ENGINEERING

CSE 311 SPRING 2024

EMPIRICAL STUDY
Final Report

Prepared By
Mustafa CABİ
20200702082

Instructor: Assistant Prof. Dr. Onur Demir
Teaching Assistant: Alara Ece Şensoy

Final Report

Title: ***Performance Evaluation of Sorting Algorithms***

Introduction

The objective of this project is to compare the performance of six sorting algorithms: BubbleSort, Improved BubbleSort, SelectionSort, QuickSort, Improved QuickSort, and MergeSort. The evaluation focuses on their time complexity and practical performance for varying input sizes. Improved versions of BubbleSort and QuickSort were implemented to enhance their efficiency. Specifically, Improved BubbleSort terminates early if the input is already sorted, and Improved QuickSort uses SelectionSort for small partitions to avoid excessive recursion.

Algorithms Description

1. **BubbleSort:** A simple comparison-based algorithm with O(n^2) time complexity.
2. **Improved BubbleSort:** Terminates early if no swaps are needed, indicating the list is already sorted.
3. **SelectionSort:** Another comparison-based algorithm with O(n^2) time complexity, selecting the smallest element in each iteration.
4. **QuickSort:** A divide-and-conquer algorithm with average O(nlogn) time complexity, but worst-case O(n^2).
5. **Improved QuickSort:** Uses SelectionSort for partitions smaller than 20 elements to improve performance.
6. **MergeSort:** A divide-and-conquer algorithm with O(nlogn) time complexity, stable and efficient for large datasets.

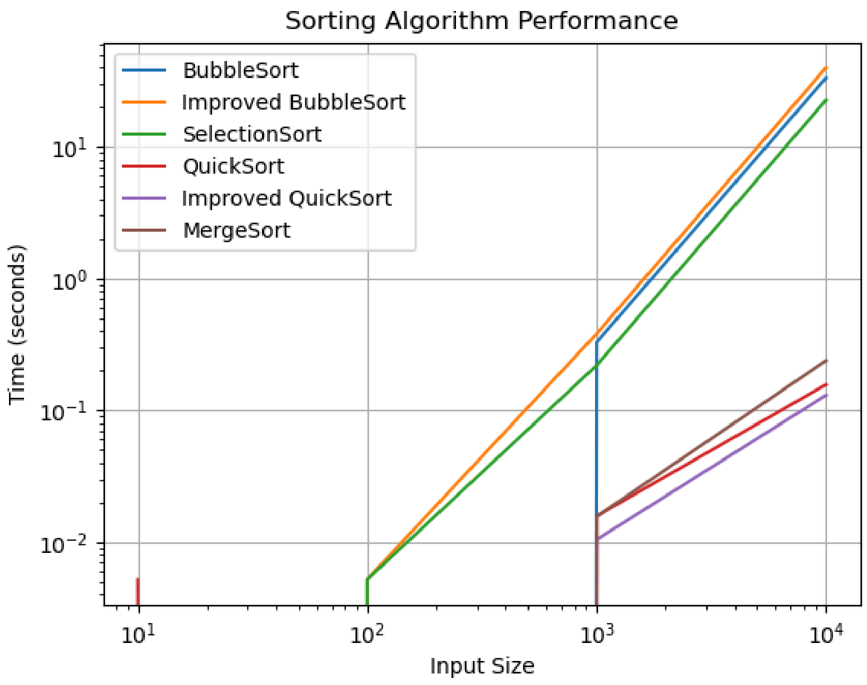
Experimental Setup

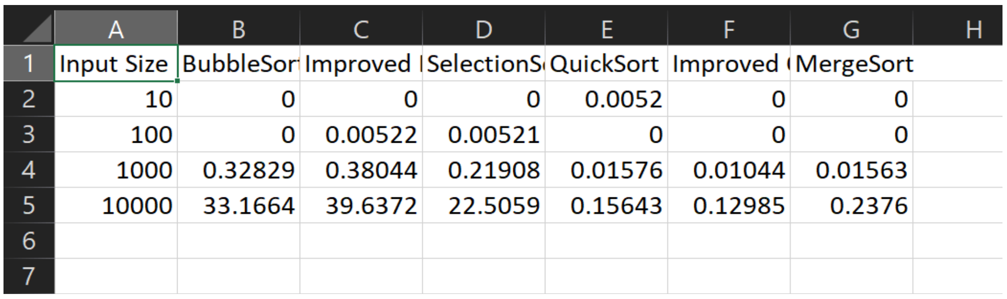
The algorithms were implemented in Python and tested on random input arrays of varying sizes. The input sizes included: 10, 100, 1000, 10000, 10000 elements. The performance of each algorithm was measured using the time module, focusing solely on the execution time of the sorting function. Each test was run multiple times, and the average time was recorded to mitigate variations. The maximum input size each algorithm could handle within 2 seconds was determined.

Note: This study was intended to be done for sizes of 100000 and 1000000, but because it took too long to output, it was tested only on sizes of 10, 100, 1000 and 10000.

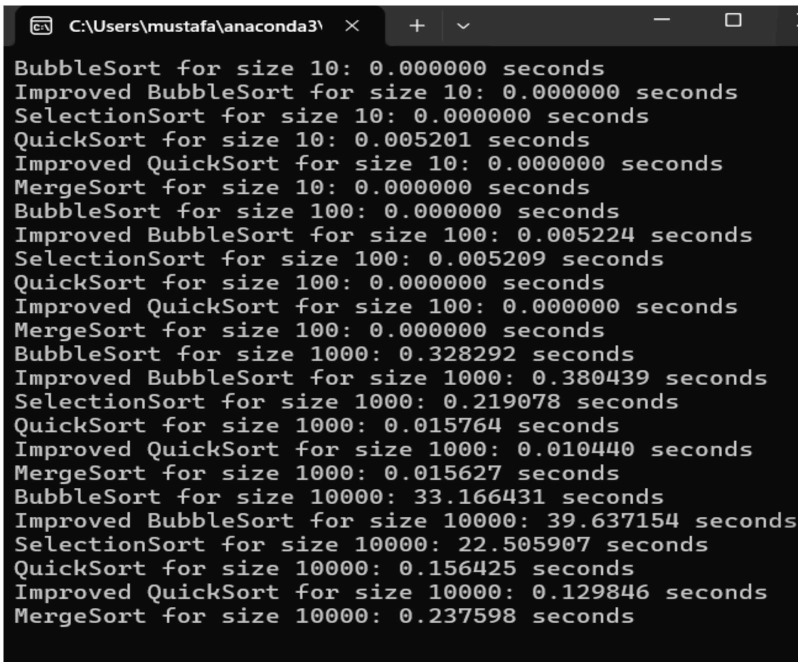
Results and Analysis

Running times of 6 algorithms for sizes 10, 100, 1000, 10000:



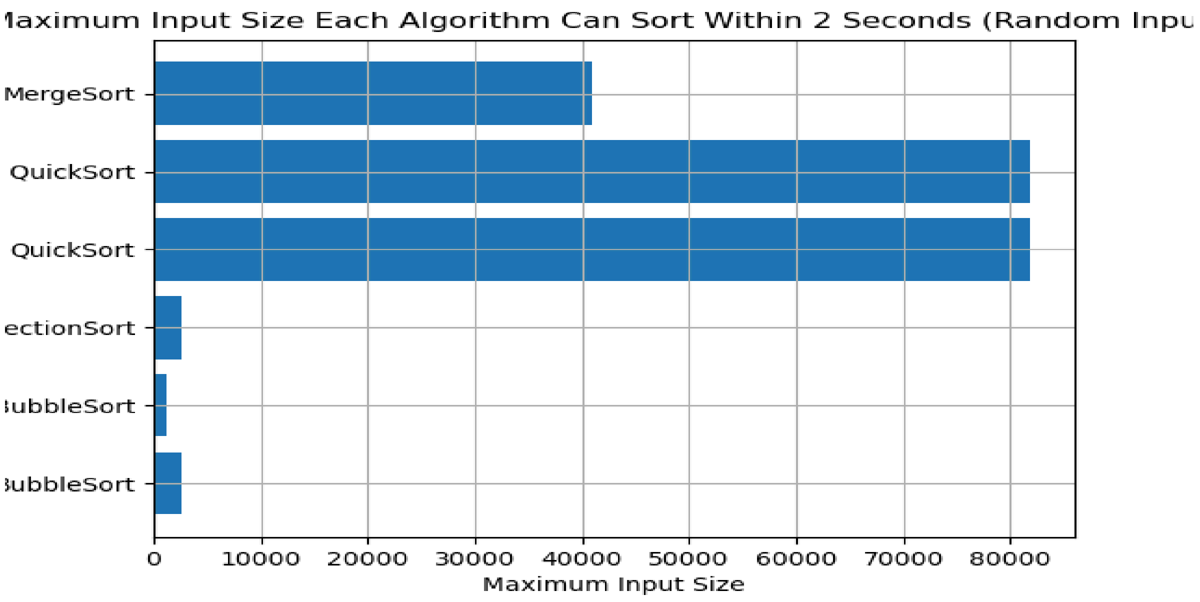


	A	B	C	D	E	F	G	H
1	Input Size	BubbleSor	Improved	SelectionS	QuickSort	Improved	MergeSort	
2	10	0	0	0	0.0052	0	0	
3	100	0	0.00522	0.00521	0	0	0	
4	1000	0.32829	0.38044	0.21908	0.01576	0.01044	0.01563	
5	10000	33.1664	39.6372	22.5059	0.15643	0.12985	0.2376	
6								
7								

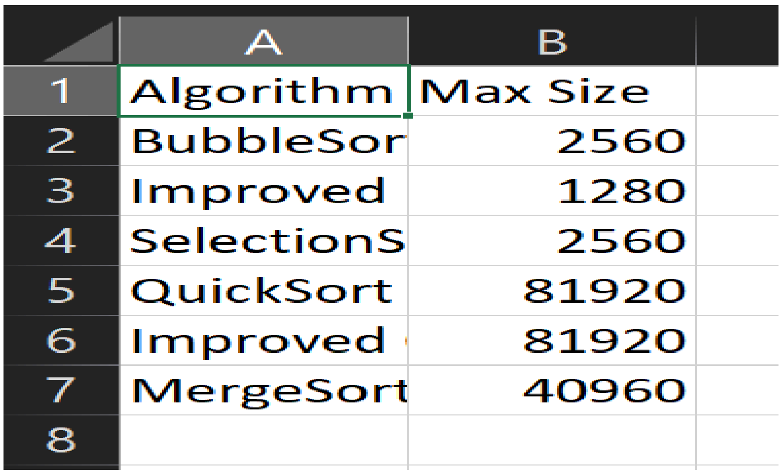


```
BubbleSort for size 10: 0.000000 seconds
Improved BubbleSort for size 10: 0.000000 seconds
SelectionSort for size 10: 0.000000 seconds
QuickSort for size 10: 0.005201 seconds
Improved QuickSort for size 10: 0.000000 seconds
MergeSort for size 10: 0.000000 seconds
BubbleSort for size 100: 0.000000 seconds
Improved BubbleSort for size 100: 0.005224 seconds
SelectionSort for size 100: 0.005209 seconds
QuickSort for size 100: 0.000000 seconds
Improved QuickSort for size 100: 0.000000 seconds
MergeSort for size 100: 0.000000 seconds
BubbleSort for size 1000: 0.328292 seconds
Improved BubbleSort for size 1000: 0.380439 seconds
SelectionSort for size 1000: 0.219078 seconds
QuickSort for size 1000: 0.015764 seconds
Improved QuickSort for size 1000: 0.010440 seconds
MergeSort for size 1000: 0.015627 seconds
BubbleSort for size 10000: 33.166431 seconds
Improved BubbleSort for size 10000: 39.637154 seconds
SelectionSort for size 10000: 22.505907 seconds
QuickSort for size 10000: 0.156425 seconds
Improved QuickSort for size 10000: 0.129846 seconds
MergeSort for size 10000: 0.237598 seconds
```

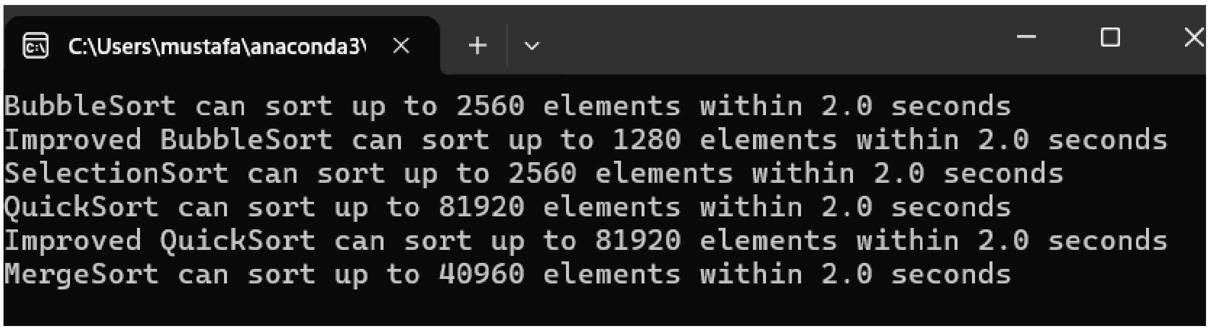
The biggest problem sizes 6 algorithms can run in two seconds



Maximum Input Sizes within 2 Seconds:



	A	B
1	Algorithm	Max Size
2	BubbleSor	2560
3	Improved	1280
4	SelectionS	2560
5	QuickSort	81920
6	Improved	81920
7	MergeSort	40960
8		



```
BubbleSort can sort up to 2560 elements within 2.0 seconds
Improved BubbleSort can sort up to 1280 elements within 2.0 seconds
SelectionSort can sort up to 2560 elements within 2.0 seconds
QuickSort can sort up to 81920 elements within 2.0 seconds
Improved QuickSort can sort up to 81920 elements within 2.0 seconds
MergeSort can sort up to 40960 elements within 2.0 seconds
```

Practical Usage and Verdict

- **BubbleSort and Improved BubbleSort:** Suitable for very small datasets due to their O(n^2) time complexity. Improved BubbleSort is slightly better due to early termination in sorted scenarios.
- **SelectionSort:** Similar to BubbleSort in performance and thus recommended only for small datasets.
- **QuickSort:** Highly efficient for large datasets with an average-case complexity of O(nlogn). However, the worst-case scenario can be mitigated by improvements.
- **Improved QuickSort:** Further enhances QuickSort by using SelectionSort for small partitions, making it highly efficient for large datasets.
- **MergeSort:** Performs consistently well with O(nlogn) time complexity, making it suitable for large datasets and stable sorting needs.

Conclusion: For practical usage, MergeSort and Normal/Improved QuickSort are the best choices for large datasets due to their efficiency and stability. BubbleSort, Improved BubbleSort, and SelectionSort are more suited for small datasets or educational purposes.

YouTube Video Demonstration

Video showing the maximum size each algorithm can run in 2 seconds:

https://youtu.be/zrUAr4tvgP8