Algorithm Analysis Report

Harry CHICHEPORTICHE , Theo DE MORAIS

February 9, 2025

1 Introduction

This report analyzes four sorting algorithms: Insertion Sort, Merge Sort, Heap Sort, and Quicksort. The number of computational steps is recorded and plotted against input size to verify asymptotic complexity. Additionally, execution time is compared across different programming languages, and fundamental proofs and divide-and-conquer analysis are provided.

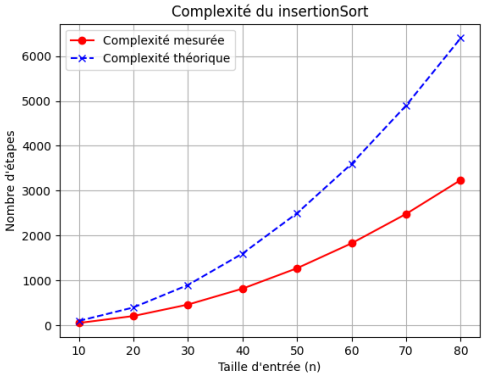
2 Task 1: Counting the Steps

The four algorithms were implemented in Python, and the number of steps was counted for varying input sizes n. The recorded results were plotted to confirm their asymptotic complexities:

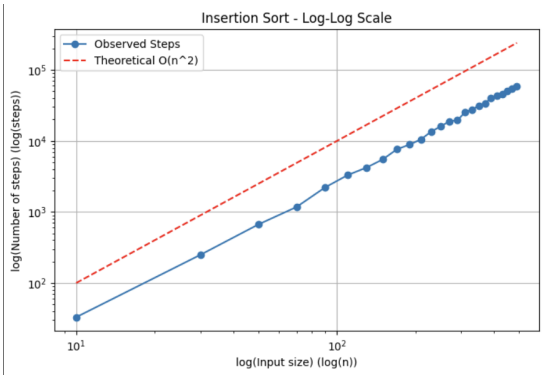
- **Insertion Sort:** $\Theta(n^2)$
- **Merge Sort:** $\Theta(n \log n)$
- **Heap Sort:** $O(n \log n)$
- **Quicksort:** $\Theta(n^2)$ (worst-case)

The plots confirm that the theoretical and experimental complexities align. The implementation can be found in the provided Jupyter Notebook.

For the case of Insertion Sort and Quick Sort, we decided to plot the complexity on a logarithmic scale to better observe the n2n2 trend of the experimental curves. Indeed, we obtain straight lines with the same slope as the theoretical curve, which accurately reflects the expected relationship.



(a) Linear Complexity



(b) Log Complexity

Figure 1: Complexity Insertion Sort Linear and Log scale

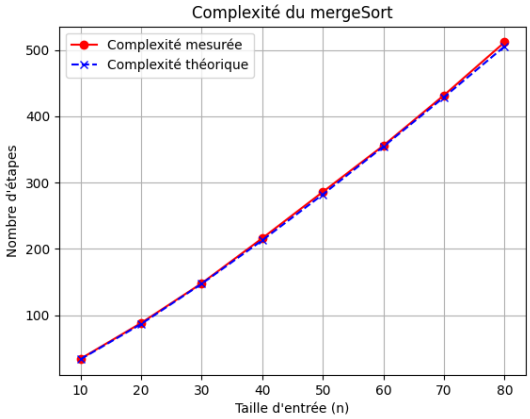


Figure 2: Complexity Merge Sort

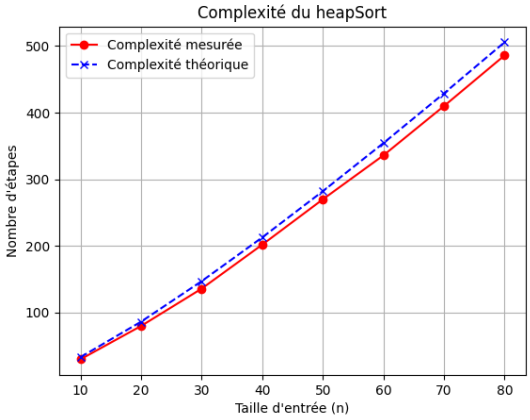
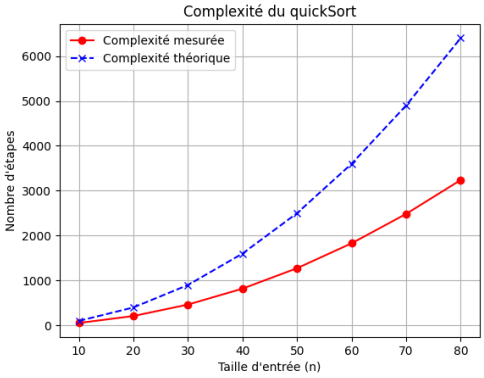
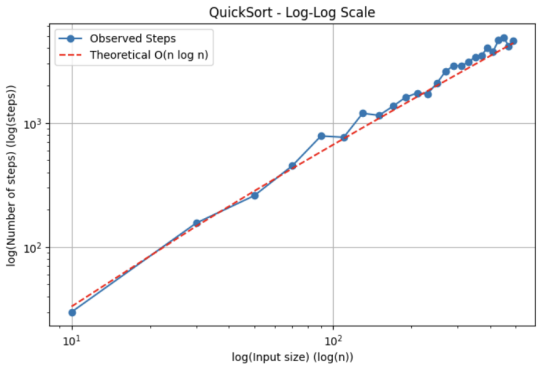


Figure 3: Complexity Heap Sort



(a) Linear Complexity



(b) Log Complexity

Figure 4: Complexity Quick Sort Linear and Log scale

We can see that, for all four algorithms, the expected complexity trends are observed.

3 Task 2: Execution Time Comparison

Insertion Sort was implemented in both Python and C. The execution times were recorded for different input sizes. The results indicate that C is significantly faster due to lower-level memory management and reduced interpreter overhead.

```
Input size: 100, Time taken: 0.000122 seconds
Input size: 1000, Time taken: 0.011361 seconds
Input size: 5000, Time taken: 0.347096 seconds
Input size: 10000, Time taken: 1.602137 seconds
Input size: 20000, Time taken: 5.224736 seconds
```

Figure 5: Execution time for different inputs in Python

```
Input size: 100, Time taken: 0.000000 seconds
Input size: 1000, Time taken: 0.000000 seconds
Input size: 5000, Time taken: 0.003000 seconds
Input size: 10000, Time taken: 0.021000 seconds
Input size: 20000, Time taken: 0.104000 seconds
```

Figure 6: Execution time for different inputs in C

4 Task 3: Basic Proofs

4.1 Proof 1: $(n+a)^b = \Theta(n^b)$ for $b > 0$

Using the definition of Θ, we need to show that there exist constants $c_1, c_2 > 0$ and n_0 such that for all $n \geq n_0$:

$$c_1 n^b \leq (n+a)^b \leq c_2 n^b.$$

Factorizing n^b:

$$(n+a)^b = n^b \left(1 + \frac{a}{n}\right)^b.$$

Since $\lim_{n\to\infty}(1+\frac{a}{n})^b = 1$, there exist constants bounding the function, proving $(n+a)^b = \Theta(n^b)$.

4.2 Proof 2: $\frac{n^2}{\log n} = o(n^2)$

By the definition of little-o notation:

$$\frac{n^2}{\log n} = o(n^2) \iff \lim_{n\to\infty} \frac{n^2/\log n}{n^2} = 0.$$

Simplifying:

$$\lim_{n\to\infty} \frac{1}{\log n} = 0.$$

Since $\log n \to \infty$ as $n \to \infty$, we conclude:

$$\frac{n^2}{\log n} = o(n^2).$$

4.3 Proof 3: $n^2 \neq o(n^2)$

Assume $n^2 = o(n^2)$. By definition, this would mean:

$$\lim_{n\to\infty} \frac{n^2}{n^2} = 0.$$

Since $\frac{n^2}{n^2} = 1$, which does not tend to 0, we reach a contradiction. Therefore, we conclude:

$$n^2 \neq o(n^2).$$

5 Task 4: Divide and Conquer Analysis

We analyze the recurrence relation:

$$T(n) = 3T(n/2) + \Theta(n)$$

Using the Master Theorem, which states that for recurrences of the form:

$$T(n) = aT(n/b) + f(n),$$

we identify:

- $a = 3$, $b = 2$, $f(n) = \Theta(n)$
- We compare $f(n)$ with $n^{\log_b a} = n^{\log_2 3} \approx n^{1.58}$

Since $f(n) = \Theta(n)$ is asymptotically smaller than $n^{\log_2 3}$, we apply Case 1 of the Master Theorem, yielding:

$$T(n) = \Theta(n^{1.58}).$$

Alternatively, using the recursion tree method, we expand the recurrence:

$$\begin{aligned} T(n) &= 3T(n/2) + n \\ &= 3(3T(n/4) + n/2) + n \\ &= 9T(n/8) + 3(n/4) + n/2 + n \end{aligned}$$

Expanding until the base case, summing the contributions at each level, and using the geometric sum formula, we confirm that:

$$T(n) = \Theta(n^{1.58}).$$

6 Conclusion

This report verifies theoretical complexity with empirical results and demonstrates the impact of programming language choice on execution speed. Further exploration can include optimized Quicksort variations and additional languages for comparison.

For the full implementation and additional resources, visit the project repository on GitHub: https://github.com/Theodemo/DAT600.git