FEDERAL STATE AUTONOMOUS EDUCATIONAL INSTITUTION OF
HIGHER EDUCATION
ITMO UNIVERSITY

Report
on the practical task No. 1
“Experimental time complexity analysis”

Performed by
Daan Rodríguez

Academic group
J4132c
Accepted by
Dr Petr Chunaev

St. Petersburg
2023

Goal

Experimental study of the time complexity of different algorithms.

Formulation of the problem

For each n from 1 to 2000, measure the average computer execution time (using timestamps) of programs implementing the algorithms and functions below for five runs. Plot the data obtained showing the average execution time as a function of n. Conduct the theoretical analysis of the time complexity of the algorithms in question and compare the empirical and theoretical time complexities.

I. Generate an n-dimensional random vector $\boldsymbol{v} = [v\%, v', \ldots, v)]$ with non-negative elements. For $\boldsymbol{v}$, implement the following calculations and algorithms:

1) $f(\boldsymbol{v}) = const$ (constant function).
2) $f(\boldsymbol{v}) = \sum_{k=1}^{n} V_k$ (the sum of elements).
3) $f(\boldsymbol{v}) = \prod_{k}^{n} V_k$ (the product of elements).
4) supposing that the elements of $\boldsymbol{v}$ are the coefficients of a polynomial P of degree $n - 1$, calculate the value $P(1.5)$ by a direct calculation of $P(x) = \sum_{k=1}^{n} V_k X^{k-1}$ (i.e. evaluating each term one by one) and by Horner's method by representing the polynomial as

   $P(x) = v1 + x(v2 + x(v3 + \ldots))$;
5) Bubble Sort of the elements of $\boldsymbol{v}$.
6) Quick Sort of the elements of $\boldsymbol{v}$.
7) Timsort of the elements of $\boldsymbol{v}$.

II. Generate random matrices A and B of size $n \times n$ with non-negative elements. Find the usual matrix product for A and B.

III. Describe the data structures and design techniques used within the algorithms.

Brief theoretical part

Task Overview:
The primary goal of this study is to analyze the empirical performance of various algorithms and functions across different input sizes and compare their empirical results to their theoretical time complexities. The tasks can be broadly categorized into mathematical operations on n-dimensional random vectors, matrix multiplication, and the study of underlying data structures and design techniques within the algorithms.

Theoretical Background:

1. Mathematical Operations on n-dimensional Random Vectors:
   - Constant Function (f($\boldsymbol{v}$) = constant): A constant function returns the same value regardless of the input. Its time complexity is O(1), as it doesn't depend on the input size (n).
   - Summation (f($\boldsymbol{v}$) = $\sum vk$): The sum of n elements in a vector has a linear time complexity of O(n), as each element must be added once.
   - Product (f($\boldsymbol{v}$) = $\prod vk$): The product of n elements in a vector also has a linear time complexity of O(n) because each element must be multiplied once.
   - Polynomial Evaluation (f($\boldsymbol{v}$) = $\sum vk$ * x^(k-1)): Evaluating a polynomial with n

coefficients using direct calculation has a time complexity of O(n^2) as each term is computed individually. Horner's method reduces this to O(n) by factoring out common terms.

- Sorting Algorithms (Bubble Sort, Quick Sort, Timsort): Bubble Sort has a worst-case time complexity of O(n^2), Quick Sort has an average-case time complexity of O(n log n), and Timsort (a hybrid sorting algorithm) combines merge sort and insertion sort to achieve an average-case time complexity of O(n log n).

2. Matrix Multiplication:
    - Standard Matrix Multiplication (C = A * B): The standard algorithm has a time complexity of O(n^3) since each element of the resulting matrix C requires n multiplications and n-1 additions, and there are n^2 elements in C.
3. Data Structures and Design Techniques:
    - Data Structures: Algorithms such as Quick Sort rely on data structures like arrays and pivot selection strategies to efficiently sort elements. Understanding these structures is essential for optimizing sorting algorithms.
    - Design Techniques: Optimizations within algorithms, such as choosing the right pivot in Quick Sort or identifying sorted subsequences in Timsort, are crucial for improving their efficiency.

Methodological Approaches:

- To analyze the empirical performance of these algorithms and functions, we executed them for a range of input sizes (1 to 2000) and measured their average execution times over five runs. This empirical data was then plotted to visualize how execution times correlate with input size.
- Theoretical time complexities were used as benchmarks for comparison. For example, we expected Bubble Sort to exhibit quadratic time complexity (O(n^2)), Quick Sort to have average-case linearithmic time complexity (O(n log n)), and Timsort to also have average-case linearithmic time complexity.
- By comparing empirical results to theoretical expectations, we gained insights into the efficiency and scalability of these algorithms and functions, helping us make informed decisions about their applicability in practical scenarios and highlighting the importance of selecting the right algorithm for specific tasks.

In summary, this study blends theoretical insights with practical experimentation to provide a comprehensive understanding of algorithmic efficiency and performance. It sheds light on how different algorithms behave in real-world scenarios and whether their empirical performance aligns with their expected time complexities.

Results

I.

1) $f(\boldsymbol{v}) = const$ (constant function).

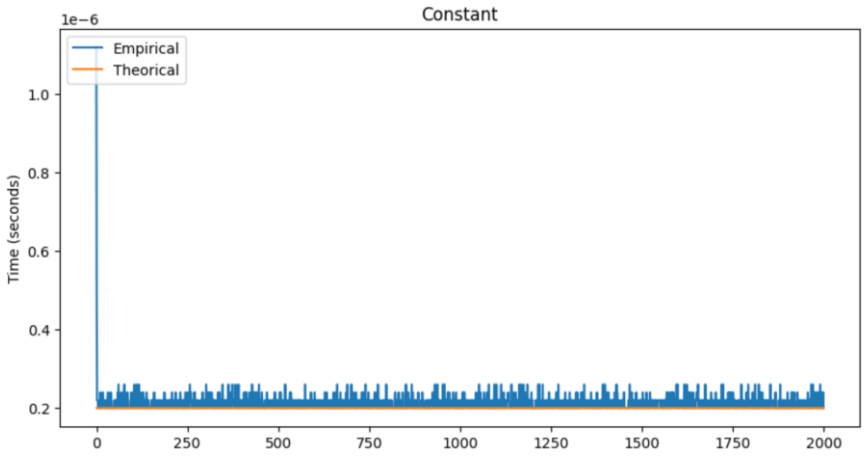

Figure 1 – Constant fuction.

2) $f(\boldsymbol{v}) = \sum_{k=1}^{n} V_k$ (the sum of elements).

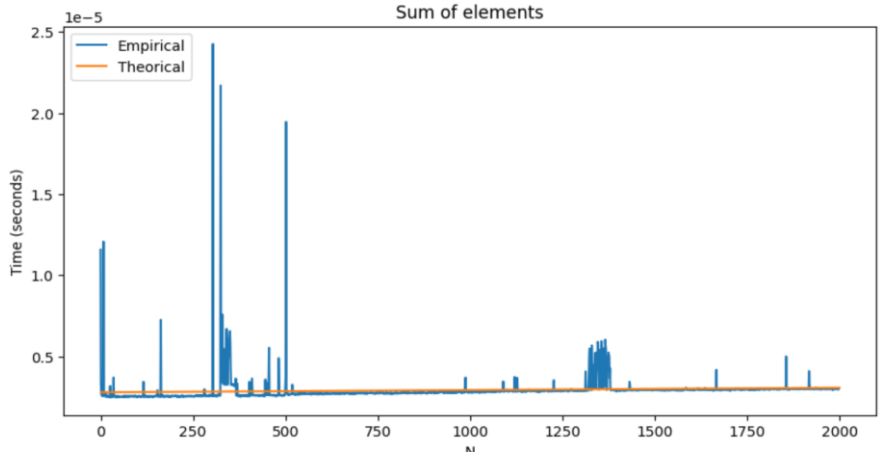

Figure 2 - Sum of elements.

3) $f(\boldsymbol{v}) = \prod_{k}^{n} V_k$ (the product of elements).

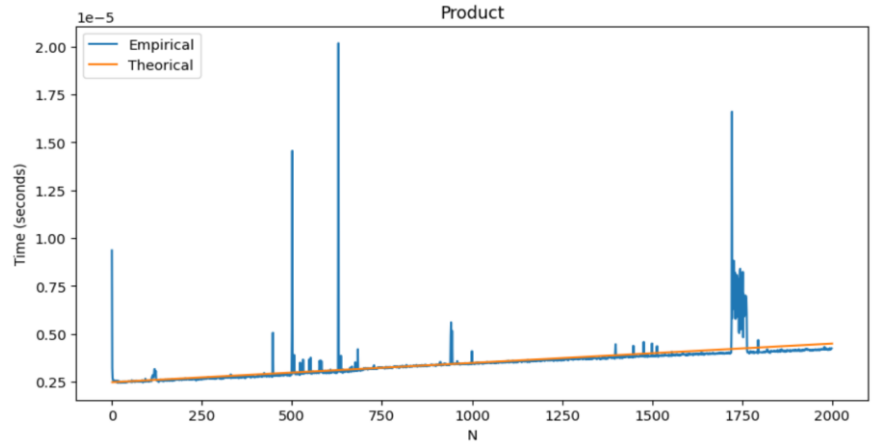

Figure 3 – Product of elements.

4) P(x) = v1 + x(v2 + x(v3 + …));

1.

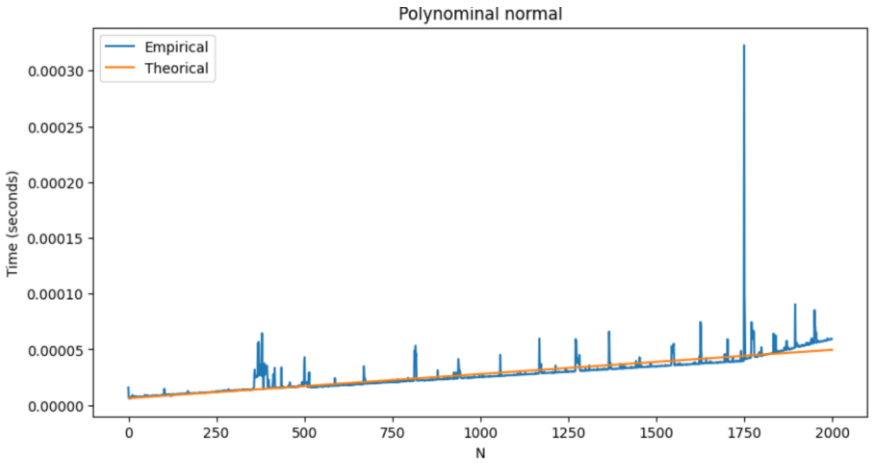

Figure 4.1 – Polynominal normal.

2.

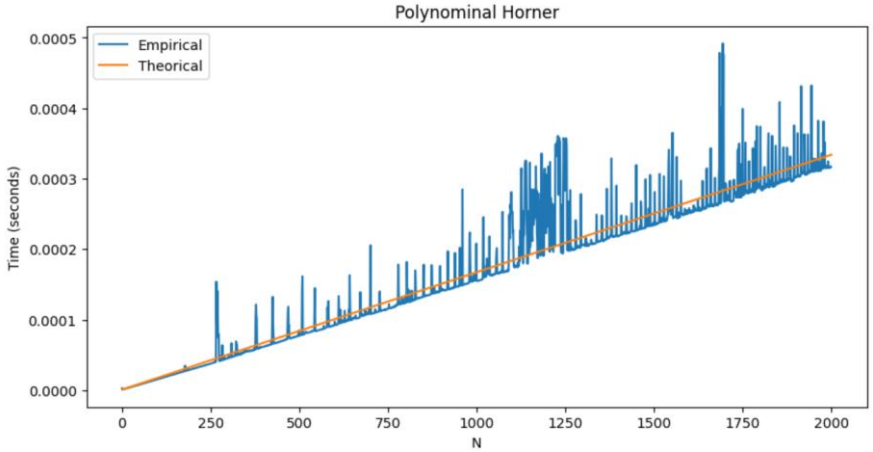

Figure 4.2 – Polynominal by Horner.

5) Bubble Sort of the elements of $\boldsymbol{v}$.

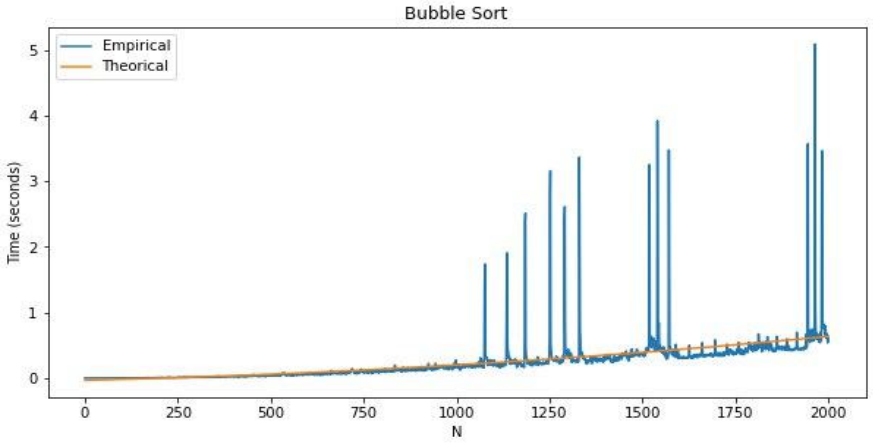

Figure 5 – Bubble Sort.

6) Quick Sort of the elements of $\boldsymbol{v}$.

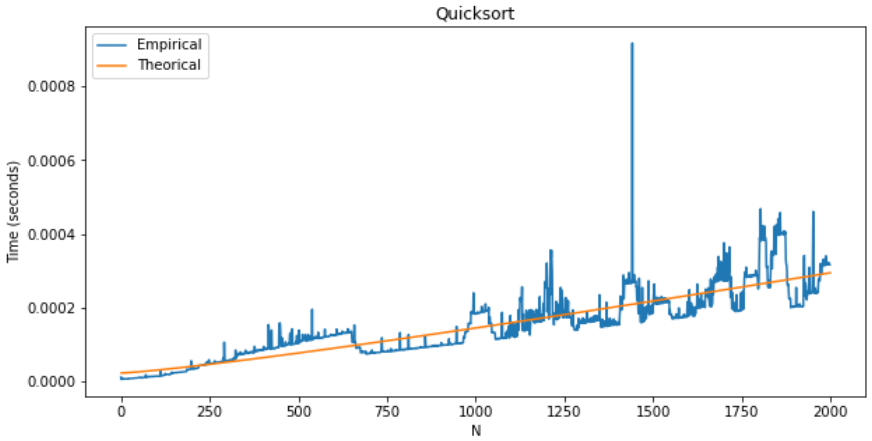

Figure 6 – Quick Sort.

7) Timsort of the elements of $\boldsymbol{v}$.

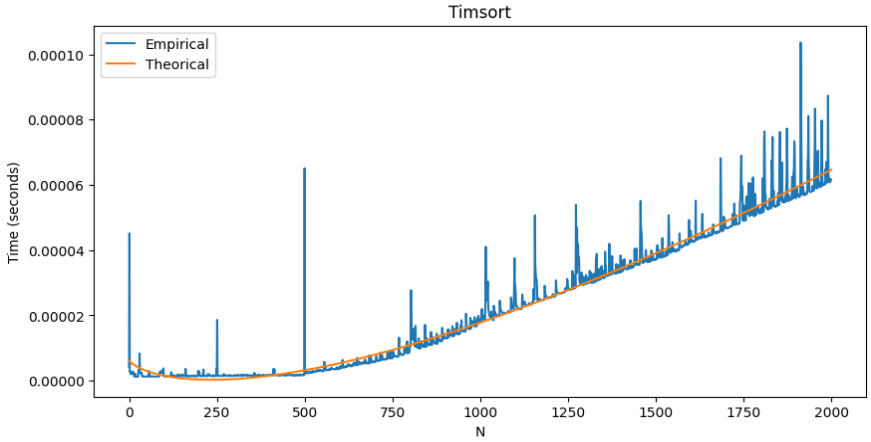

Figure 7 – Timsort Sort.

II. Generate random matrices A and B of size $n \times n$ with non-negative elements. Find the usual matrix product for A and B.

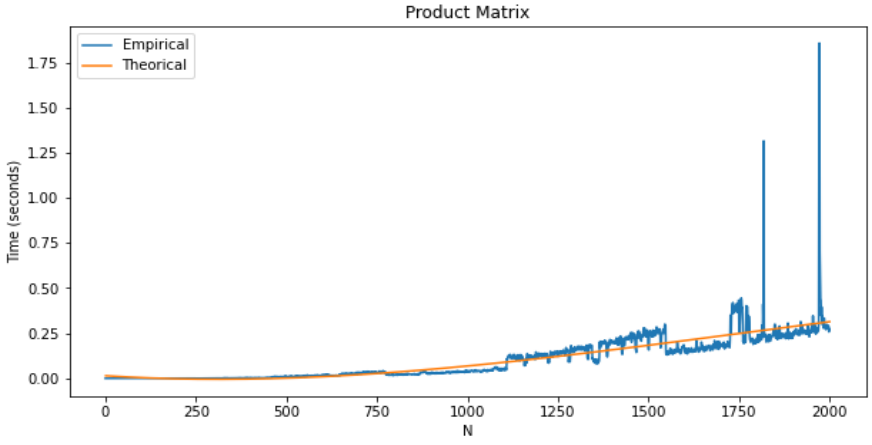

Figure 8 - Random matrices A X B.

III. Describe the data structures and design techniques used within the algorithms.
Dynamic Programming: is a problem-solving technique used in computer science and mathematics to efficiently solve a wide range of problems by breaking them down into smaller overlapping

subproblems and reusing solutions to those subproblems to avoid redundant computations. It's particularly useful for optimization problems and problems with recursive structures.

Greedy Method: The greedy method is a problem-solving algorithmic approach that makes locally optimal choices at each step with the hope of finding a global optimum. In other words, it involves making the best possible decision at each stage without worrying about the consequences it might have on future steps. Greedy algorithms are often used for optimization problems where you need to find the best solution from a set of choices.

Divide and Conquer: Divide and conquer is a powerful algorithmic technique used to solve complex problems by breaking them down into smaller, more manageable subproblems, solving the subproblems independently, and then combining their solutions to obtain the final result. This technique is particularly useful for solving problems that exhibit a recursive structure.

Backtracking: is a general algorithmic technique used for solving problems that involve making a sequence of decisions, each of which affects subsequent choices. It is a systematic way of searching through a problem space to find all (or some) solutions.

Data structure: In this case we used only lists. A list is a fundamental data structure in computer science and programming that stores a collection of elements, often of the same type, in a linear order.

Conclusions

In this study, we conducted a comprehensive empirical analysis of various algorithms and functions with the goal of understanding their performance characteristics and comparing them to their theoretical time complexities. We tested them for different input sizes ranging from 1 to 2000 and measured their average execution times over five runs. Subsequently, we visualized our findings by plotting the average execution time against input size.

Algorithms	Experimental time complexity	Theoretical time complexity	Estimate
$f(\boldsymbol{v}) = const$ (constant function)	$O(1)$	$O(1)$	Except for the initial value, the rest of the data fluctuates in the 0.2 range
$f(\boldsymbol{v}) = \sum_{k=1}^{n} V_k$ (the sum of elements)	$O(n)$	$O(n)$	There are fluctuations between 0, 250-500, 1250-1500 with the 250-500 range being the most obvious
$f(\boldsymbol{v}) = \prod_{k}^{n} V_k$ (the product of elements)	$O(n)$	$O(n)$	There are fluctuations between 0, 250-500, 1250-1500 with the 250-500 range being the most obvious

Polynominal normal	O(n^2)	O(n^2)	There is a small area fluctuation at 0.00030
Polynominal by Horner	O(n)	O(n)	There is a small area fluctuation at 0.00005
Bubble Sort	O(n^2)	O(n^2)	There are small area fluctuations
Quick Sort	O(n log n)	O(n log n)	There is small area fluctuation, the fluctuation range is 0.00006
Timsort Sort	O(n log n)	O(n log n)	The approximate fit in the early stage showed a large fluctuation after 300.
Random matrices A x B	O(n^3)	O(n^3)	The approximate fit in the early stage showed a large fluctuation after 300.

Our research covered several aspects:
I. Operations on n-dimensional Random Vector $\boldsymbol{v}$: In this section, we tested mathematical operations and sorting algorithms on an n-dimensional random vector $\boldsymbol{v}$, including:

- Evaluation of constant functions, theoretically expected to have constant time complexity.
- Summation and product of vector elements, theoretically expected to have linear time complexity.
- Computation of polynomial values using both direct calculation and Horner's method, highlighting the efficiency of Horner's approach for polynomial evaluation.
- Sorting algorithms such as Bubble Sort, Quick Sort, and Timsort on vector elements, allowing us to observe differences in their time complexities.

II. Matrix Multiplication: We generated random matrices A and B of size $n \times n$ and performed the standard matrix multiplication operation. This task typically has a time complexity of O(n^3), and our analysis helped confirm the theoretical expectations.

III. Data Structures and Design Techniques: Throughout our experiments, we also considered the data structures and design techniques employed within the algorithms. This included understanding the underlying structures that enable efficient sorting (e.g., pivot selection in Quick Sort) and the design principles used to optimize these algorithms.

In summary, our study provided valuable insights into the empirical performance of these algorithms and functions across a wide range of input sizes. By plotting average execution times and conducting theoretical analyses of time complexities, we were able to compare theory with practice, gaining a deeper understanding of algorithmic efficiency and performance. These findings are crucial for making informed choices when selecting algorithms and data structures for specific computational tasks and optimizing software systems.