FEDERAL STATE AUTONOMOUS EDUCATIONAL INSTITUTION
OF HIGHER EDUCATION
ITMO UNIVERSITY

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

Performed by
Alexandra Matveeva
J4134c
Accepted by
Dr Petr Chunaev

St. Petersburg
2021

Goal

Experimental study of the time complexity of different algorithms

Problems

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 $v = [v_1, v_2, \dots, v_n]$ with non-negative elements. For v, implement the following calculations and algorithms:

1) $f(v) = const$ (constant function);
2) $f(v) = \sum_{k=1}^{n} v_k$ (the sum of elements);
3) $f(v) = \prod_{k=1}^{n} v_k$ (the product of elements);
4) supposing that the elements of 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 $P(x) = v_1 + x(v_2 + x(v_3 + \cdots)$;
5) Bubble Sort of the elements of v;
6) Quick Sort of the elements of v;
7) Timsort of the elements of 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

An **algorithm** is any well-defined computational procedure that takes some value, or set of values, as input and produces some value, or set of values, as output. An algorithm is thus a sequence of computational steps that transform the input into the output. The main characteristic of an algorithm is **the time complexity**. Time complexity signifies the total time required by the program to run till its completion. It is most commonly expressed using the Big-O notation $O(f(n))$.

Constant function $f(x) = c$, where $c - const$ is a function that has the same output value (no matter what input value is). Algorithm of calculation of constant function has constant time complexity $O(1)$ and is called constant time algorithm. An algorithm is said to be a constant time algorithm if $T(n)$ ($T(n)$ is the function that demonstrate the dependence of the execution time on the size of the input data) is limited to a value that does not depend on the size of the input data.

The **sum of elements** algorithm and the **product of elements** algorithm are sequential algorithms and have linear time complexity $O(n)$. It means that execution time proportional to their input. Algorithms that require iterating through all elements of an array/list in a *for*/*while* loop have such complexity (i.e. Horner's method).

Bubble sort is one of the simplest, but also inefficient algorithms that is used to sort small arrays.

In general, we can define the next steps of bubble sort:

1. Compare the current item with the next one.

2. If the next element is smaller/larger than the current one, swap them.

3. If the array is sorted, finish the algorithm, otherwise go to step 1.

Best Time Complexity: $O(n)$

Average and Worse Time Complexity: $O(n^2)$

Quick sort algorithm is based on the divide-and-conquer approach. The general scheme is as follows:

1. some reference element $a[i]$ is selected from the array

2. an array splitting procedure is started, which moves all keys smaller or equal to $a[i]$ to the left of it, and all keys larger or equal to $a[i]$ to the right

3. now the array consists of two subsets, and the left one is less than or equal to the right one

4. for both subarrays: if there are more than two elements in the subarray, recursively run the same procedure for it

At the end, you will get a completely sorted sequence.

Best Time Complexity: $O(n\ log\ n)$

Average Time Complexity: $O(n\ log\ n)$

Worst Time Complexity: $O(n^2)$

Timsort is a stable sorting algorithm that works in $O(n\ Log\ n)$ time. This algorithm based on dividing the array into blocks known as Run. Each Run is sorted by insertion sort and after that merged by combine function used in merge sort. Depending on the size of array, the run size can range from 32 to 64. Merge function performs well when the size subarrays are powers of 2. The idea is based on the fact that for small arrays, insertion sort works well. In best-case time complexity of Timsort is $O(n)$.

Results

Theoretical time complexity of calculating function $f(v) = const$ (constant function) is $O(1)$. The function was approximated by a constant function of the form $f(v) = a$ $(a > 0)$.

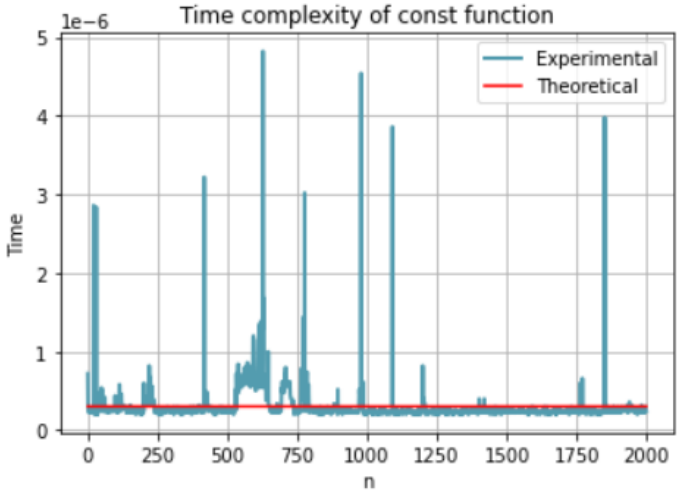

Fig. 1 – Time complexity of the constant function

Theoretical time complexity of calculating functions $f(v) = \sum_{k=1}^{n} v_k$ is $O(n)$. The function of time was approximated by a function $f(v) = a\,v\ (a > 0)$. Graphs of empirical and theoretical time complexities are shown in Figure 2.

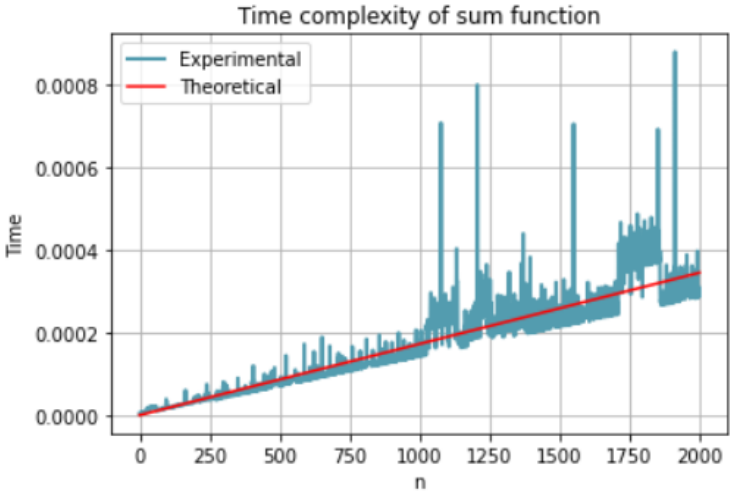

Fig. 2 - Time complexity of the sum elements function

Theoretical time complexity of calculating functions $f(v) = \prod_{k=1}^{n} v_k$ is $O(n)$. The function of time was approximated by a function $f(v) = a\,v\ (a > 0)$. Graphs of empirical and theoretical time complexities are shown in Figure 3.

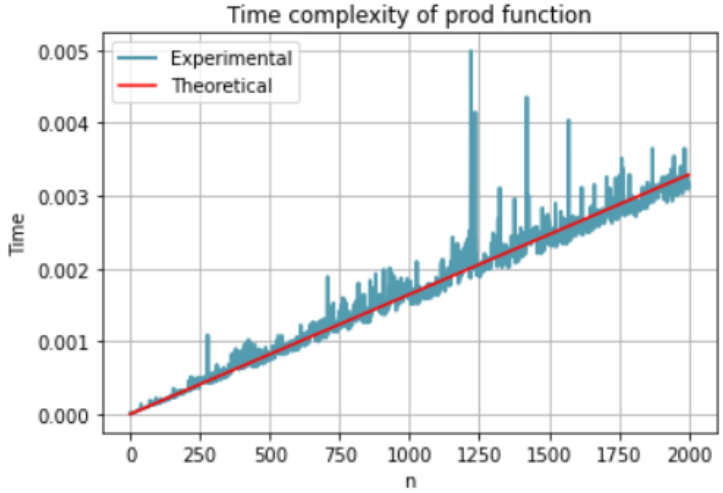

Fig. 3 – Time complexity of the product of the elements

Theoretical time complexity of direct calculation of polynomial function is $O(n \log n)$. The function of time was approximated by a function $f(v) = a\,v \log v\ (a > 0)$. Graphs of empirical and theoretical time complexities of direct calculation of polynomial function are shown in Figure 4. If we use Horner's method then time complexity of calculation of polynomial function become $O(n)$ (Figure 5).

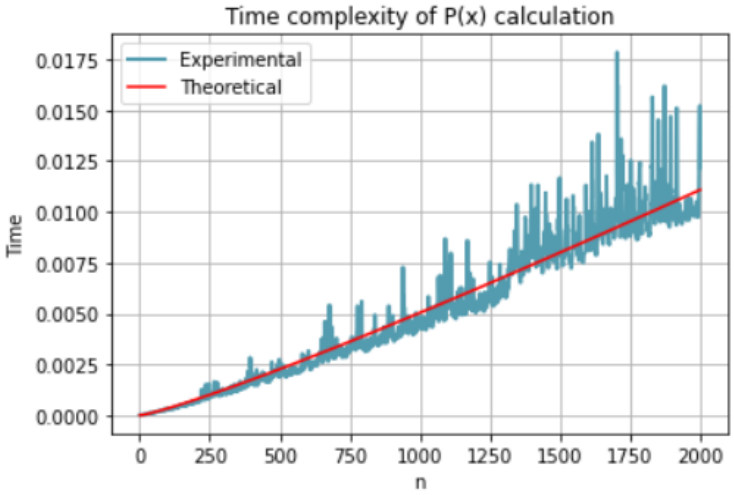

Fig. 4 – Time complexity of the direct calculation of polynomial function

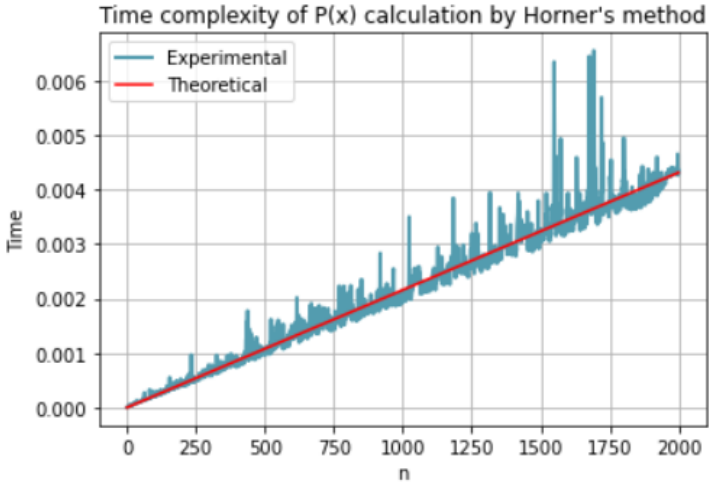

Fig. 5 – Time complexity of Horner's method

Theoretical time complexity of Bubble Sort algorithm is $O(n^2)$ in average and worst cases. The function of time was approximated by a function $f(v) = av^2$ $(a > 0)$. Graphs of empirical and theoretical time complexities are shown in Figure 6.

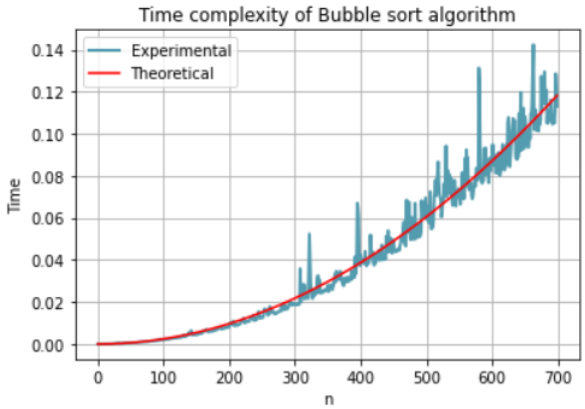

Fig. 6 – Time complexity of Bubble Sort algorithm

Theoretical time complexity of Quick Sort algorithm is $O(n\ log\ n)$ in average and best cases. The function of time was approximated by a function $f(v) = a\ v\ log\ v\ (a < 0)$. Graphs of empirical and theoretical time complexities are shown in Figure 7.

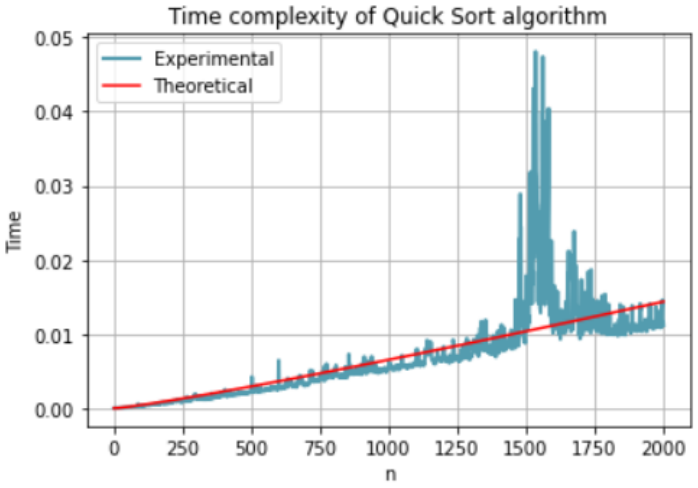

Fig. 7 – Time complexity of Quick Sort algorithm

Timsort has theoretical time complexity $O(n\ log\ n)$ in average and worst cases. The function of time was approximated by a function $f(v) = a\ v\ log\ v\ (a > 0)$. Graphs of empirical and theoretical time complexities are shown in Figure 8.

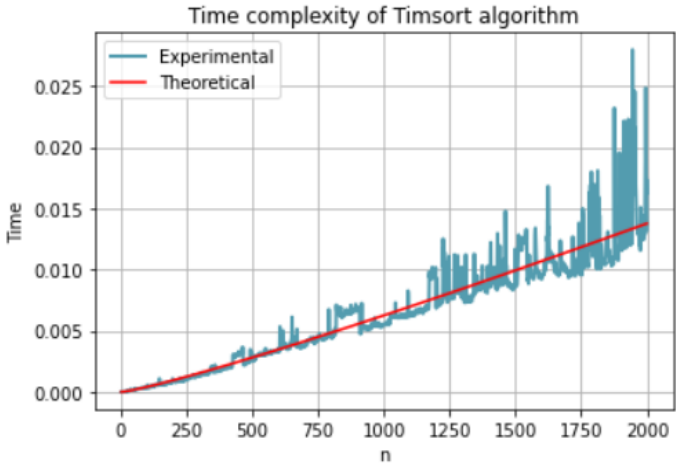

Fig. 8 – Time complexity of Timsort algorithm

Theoretical time complexity of multiplication matrixes algorithm is $O(n^3)$. The function of time was approximated by a function $f(v) = a\,v^3\ (a < 0)$. Graphs of empirical and theoretical time complexities are shown in Figure 9.

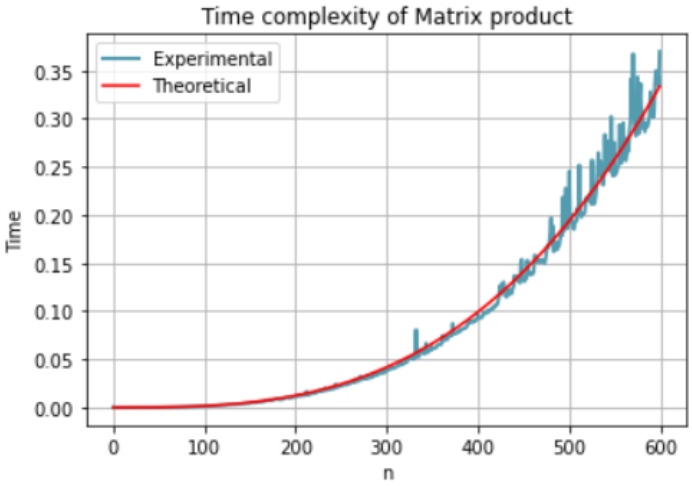

Fig. 9 – Time complexity of product of matrices algorithm

Data structures and design techniques

A list is a data structure that contains an ordered set of elements, i.e. stores a sequence of elements.

To perform matrix product numpy.matmul() was used.

Conclusion

During laboratory work performance, a theoretical analysis of the time complexity of nine algorithms was fulfilled. The obtained empirical data deviate from the theoretical, but their similarity is obvious and demonstrated in the graphs. As a result we demonstrate the dependence of the execution time of various algorithms on the amount of input data.

Appendix

GitHub Link: https://github.com/alex-mat-s/Algorithms/blob/main/Lab1.ipynb