FEDERAL STATE AUTONOMOUS EDUCATIONAL INSTITUTION
OF HIGHER EDUCATION
ITMO UNIVERSITY

Report
on the practical task 1
“Experimental time complexity analysis”

Performed by
Ilya Lyalinov
Academic group J4134c
Accepted by
Dr Petr Chunaev

St. Petersburg
2021

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* $v = [v_1, \dots, v_n]$ *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 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*
$P(x) = v_1 + x(v_2 + x(v_3 + \dots))$

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 × 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.*

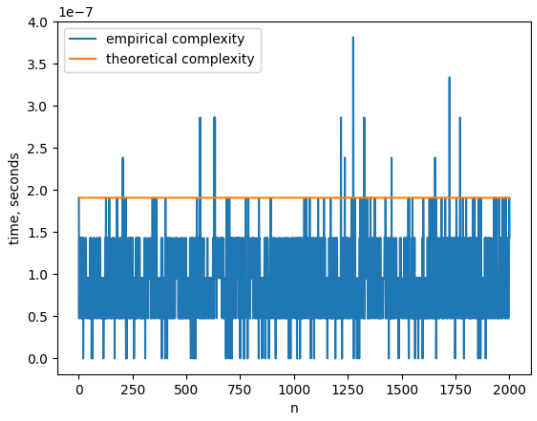
Brief theoretical part

Time complexity estimates for the corresponding algorithms from the "problems formulation" section:

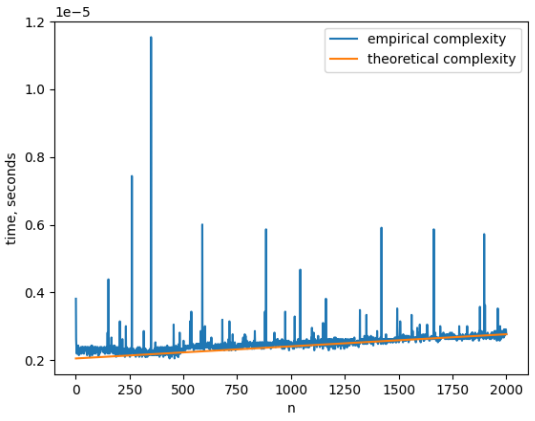
I. 1) O(1)
2) O(n)
3) O(n)
4) a) O($n * log(n)$) because time complexity of calculating x^n is O(log(n))
4) b) O(n)
6) O(n^2)
7) O($n * log(n)$) (average time complexity)
8) O($n * log(n)$) (average time complexity)

II. O(n^3) for naive implementation. Also there are methods with O($n^{2.3728596}$) time complexity. We use theoretical estimation for naive implementation.

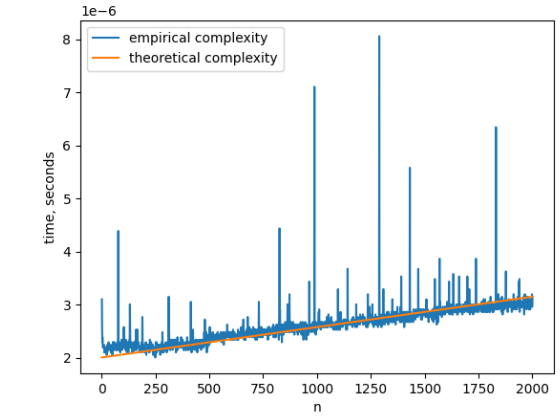
Results



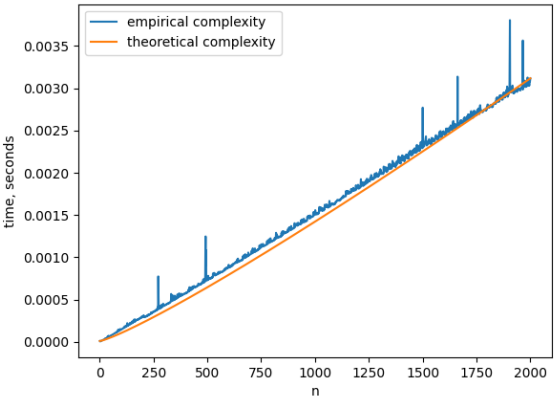
const



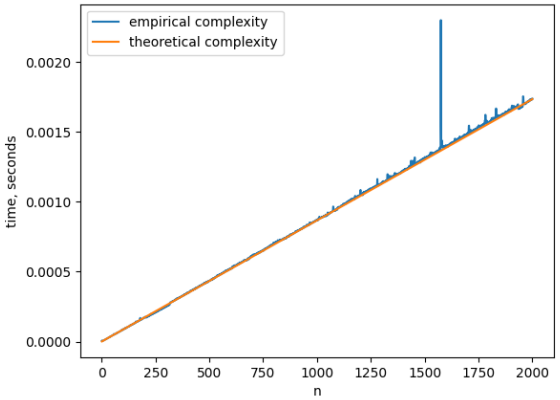
sum



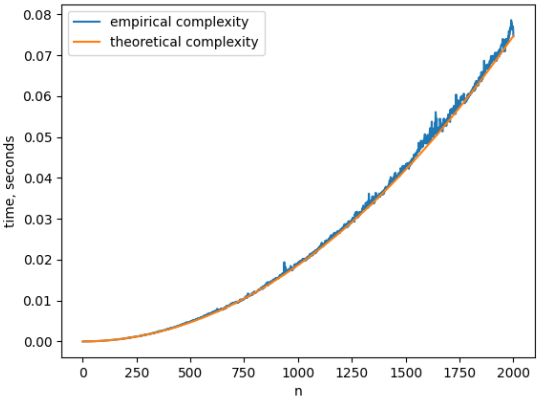
product



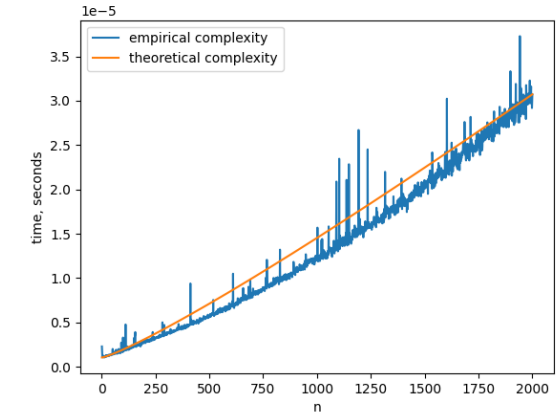
P(1.5)



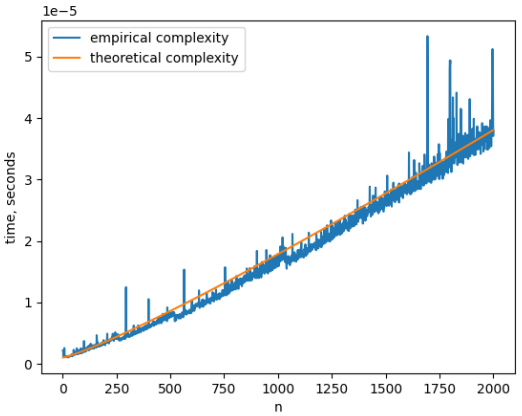
P(1.5) Horner’s method



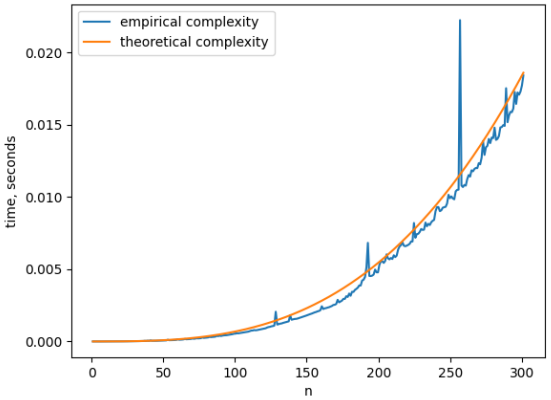
bubble sort



quicksort



timsort



square matrix multiplication

Theoretical curves were constructed using the formula $c_1 * f(n) + c_0$, where O(f) is the corresponding algorithm's complexity. Constants c_i were calculated using the equation of intersection of the empirical and theoretical curves at some points. The value for n = 1 for the theoretical curve was taken to equal to minimal empirical time of work of the corresponding method.

Data structures: one and two-dimensional arrays were used.
Design techniques of algorithms: divide and conquer in quicksort and timsort.

Divergence on sum and product charts is the result of a parallel implementation of the corresponding methods of numpy library.

Conclusion

We measured the average runtime of each algorithm for 5 runs for different n. We also proposed a method to visualize the theoretical complexity curve for these algorithms. The visual divergence between the theoretical and empirical curves was as expected. We also described used data structures and known algorithms design techniques.

Appendix

https://github.com/ilyalinov/Algorithms