FEDERAL STATE AUTONOMOUS EDUCATIONAL INSTITUTION OF HIGHER EDUCATION
ITMO UNIVERSITY

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

Performed by
Denis Zakharov,
Maxim Shitilov
Academic group J4132c
Accepted by
Dr Petr Chunaev

St. Petersburg
2022

**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.

**Brief theoretical part**

Time complexity is an important part when writing an algorithm. So let's give a definition of Time Complexity — is the amount of time taken by an algorithm to run, as a function of the length of the input. It measures the time taken to execute each statement of code in an algorithm. It is not going to examine the total execution time of an algorithm.

Time complexities used Big O notation listed below:

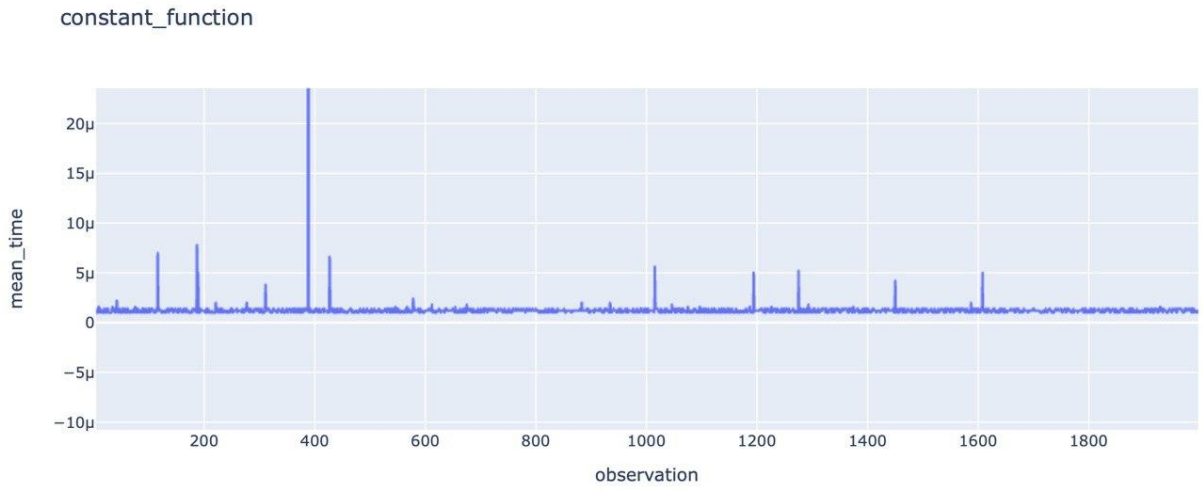
- constant time — O(1)
- logarithmic time — O(log n)
- linear time — O(n)
- linearithmic time— O(n log n)
- quadratic time — $O(n^2)$
- exponential time — $O(2^n)$
- factorial time — $O(n!)$

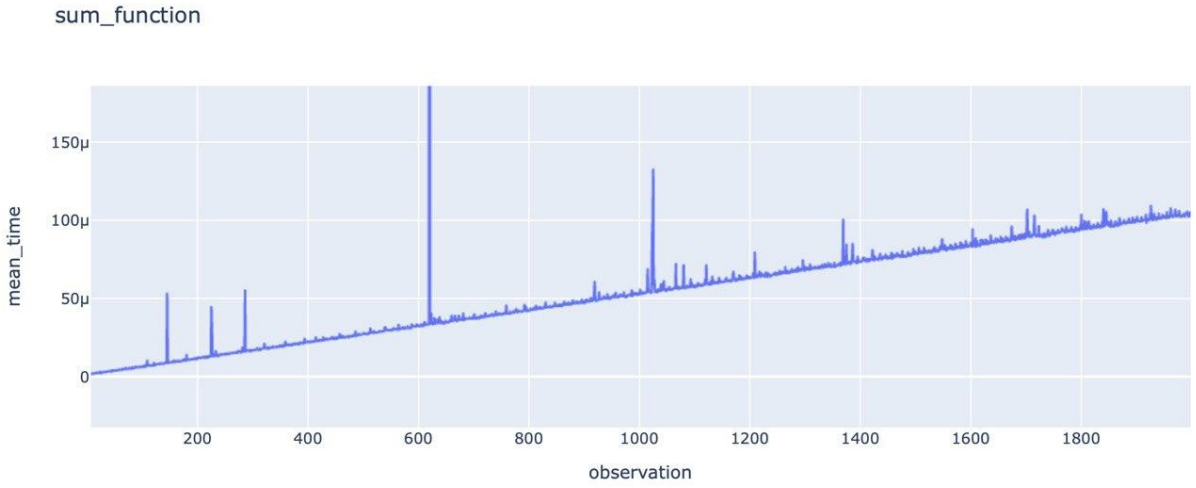
There are several algorithms that we are putting our hands on in this lab, to be more exact as an input they takes a random generated vector of numbers and do the job:

- Constant function;
    - Description — function whose (outputs) value is the same for every iteration.
    - Time Complexity — O(1)
- The sum of elements;
    - Description — function which sums all elements from vector
    - Time Complexity — O(n)
- The product of elements;
    - Description — function which multiplies all elements from vector
    - Time Complexity — O(n)
- Horner's method;
    - Description — this method used for approximating the roots of polynom with the root of 1.5,
    - Time Complexity — $O(n^2)$ which can be improved to O(n log n)
- Bubble Sort of the elements;

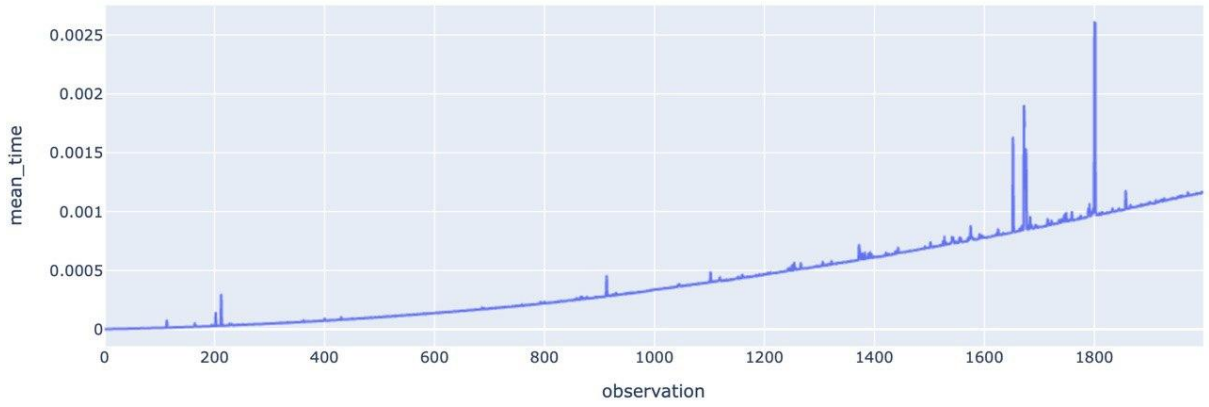
    - Description — is one of the naive sorting algorithms.
    - Time Complexity — $O(n^2)$
- Quick Sort of the element;
    - Description — it is a Divide and Conquer algorithm (*class of algorithms that used approach Divide: This involves dividing the problem into smaller sub-problems. Conquer: Solve sub-problems by calling recursively until solved. Combine: Combine the sub-problems to get the final solution of the whole problem.*). It picks an element as a pivot and partitions the given array around the picked pivot.
    - Time Complexity
        - Best case: O(n log n) we don't use 3-way partition procedure that improves TC to O(n);
        - Worst case: $O(n^2)$
- Timsort of the elements.
    - Description — sorting algorithm based on Insertion Sort and Merge Sort.
    - Time Complexity — is an adaptive sorting algorithm that needs O(n log n)

**Results**

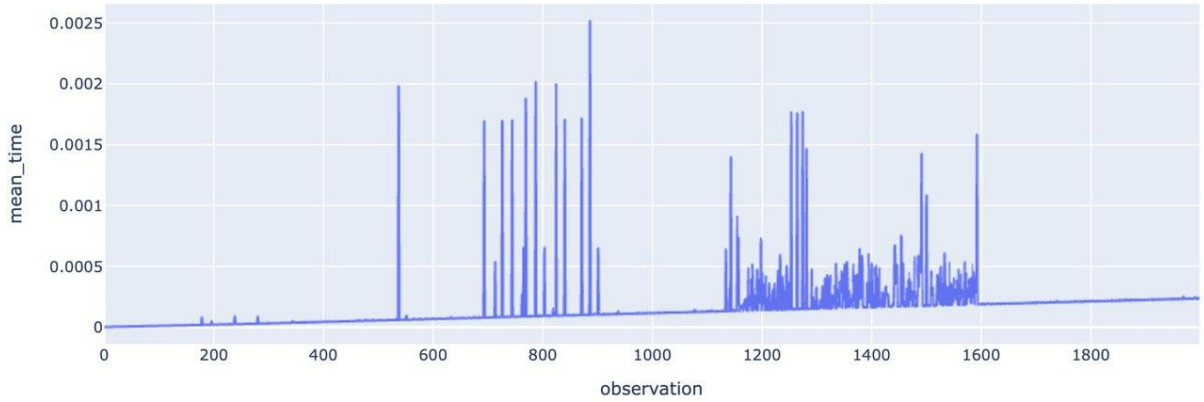


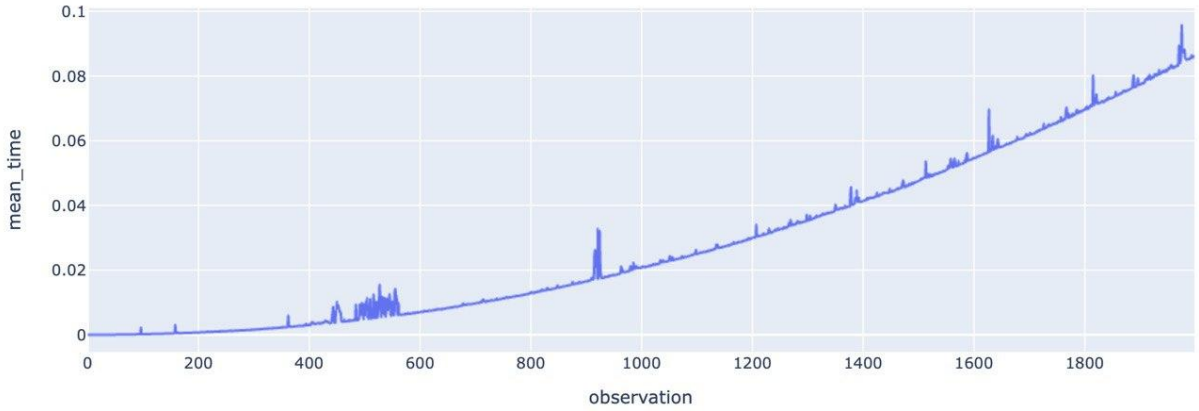
product_function

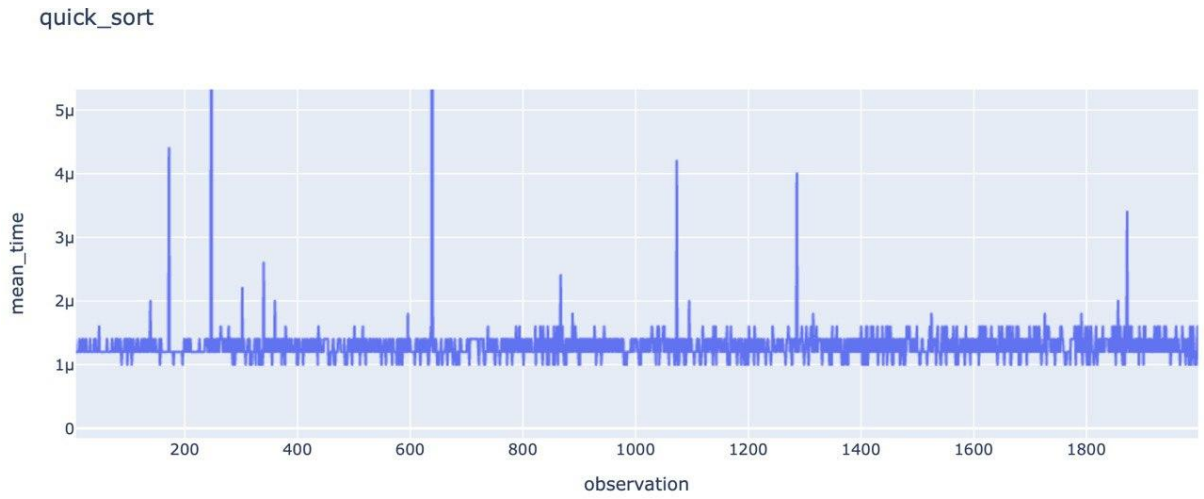


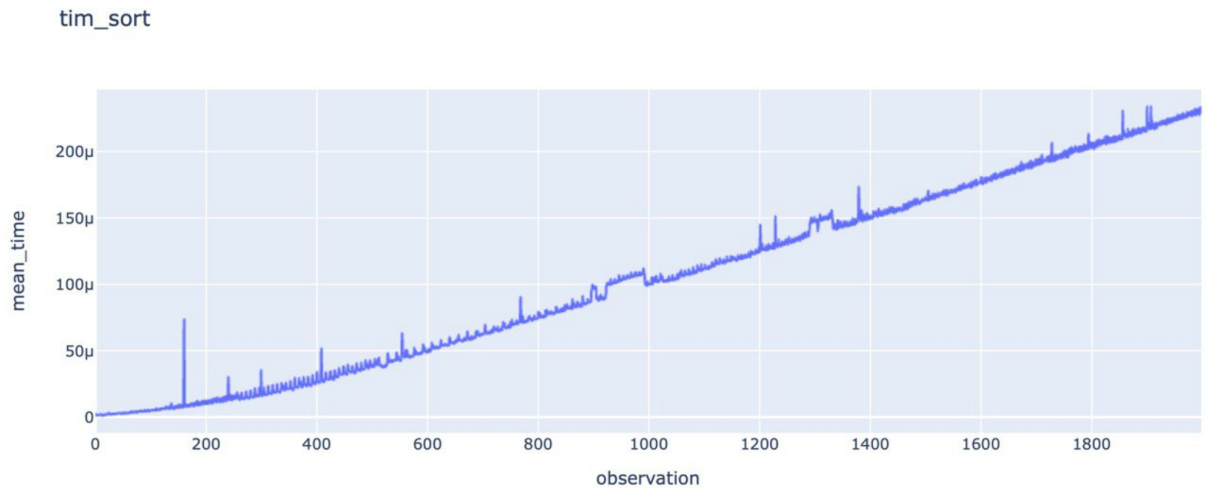

horners_function

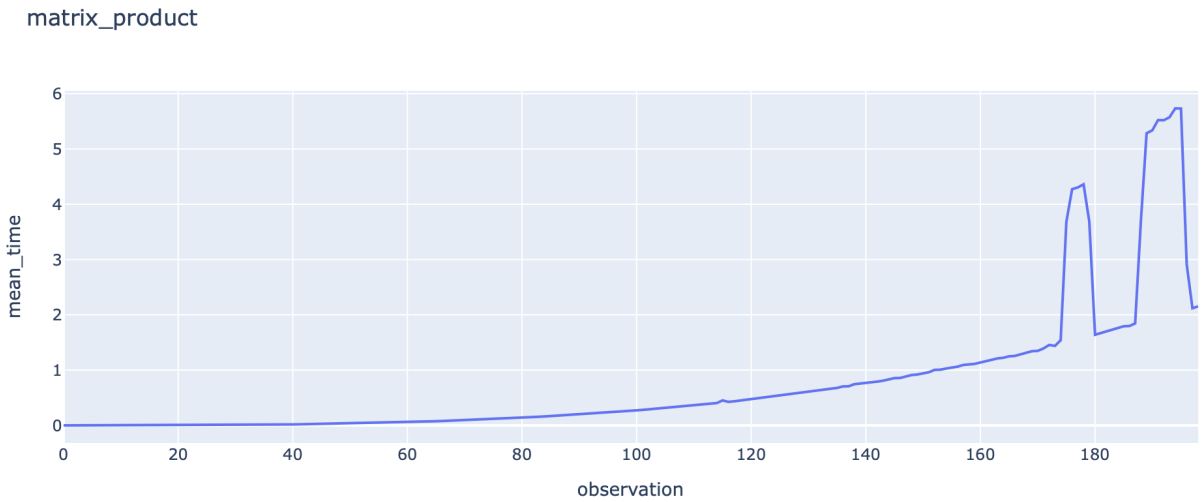



bubble_sort






## Conclusions

As can be seen from results above all method have good result in matter of function minimization, because the value of function and parameters are pretty close as for linear function as for rational. However, the most important difference between these methods is certainly Time Complexity and resource consumption for some point. For example, Product of Two Matrices 2000 x 2000 elements was counting about 7 hours and still without the

result, what's more Exhausting Search method consumes huge amount of time and resources. That is why we do not recommend to use this method of function minimization in case when different methods can be implied.

**Questions for self-monitoring from Russian educational materials**

1. Denote the number of vertices in a graph by n. What does the expression O(n) describing time complexity of some algorithm on graphs mean?

   We suppose the algorithm takes linear time to walk through each vertex in a graph.

2. Why is it necessary to average over several runs of the algorithm's running time measurements in the empirical analysis of the algorithm's temporal complexity?

   We suppose that the main goal for doing this is to exclude any errors/emissions, so we get the average practical runtime of the algorithm, which approximates the theoretical time complexity. As it said in Wikipedia — "*algorithm's running time may vary among different inputs of the same size*".

**Appendix**

DataLore : site. – URL: https://datalore.jetbrains.com/notebook/RemqSkuJwmr1PM4Gc3cBqB/IQU3E9TykxWZrIKQdPEZeN (circulation date: 09.09.2022)