FEDERAL STATE AUTONOMOUS EDUCATIONAL INSTITUTION
OF HIGHER EDUCATION
ITMO UNIVERSITY

Report
on the practical task No. 1
Experimental Time Complexity Analysis

Performed by:
Roman Bezaev
J4133c

Accepted by:
Dr Petr Chunaev

St. Petersburg
2021

1 Goal

The goal of this work is to empirically estimate the time complexity of few basic algorithms.

2 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.

Task I. Generate an n-dimensional random vector v with non-negative elements. For v implement the following calculations and algorithms:

1. $f(v) = const$
2. $f(v) = \sum_{k=1}^{n} v_k$
3. $f(v) = \prod_{k=1}^{n} v_k$
4. Suppose that elements of v is a coefficients of a polynomial of degree $n-1$ **P**. Calculate **P(1.5)** by naive method and by Gorner method
5. Bubble Sort of the elements of v_k
6. Quick Sort of the elements of v_k
7. Timsort of the elements of v_k

Task II. Generate random matrices **A** and **B** of size NxN with non-negative elements. Find the usual matrix product for **A** and **B**.

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

3 Brief theoretical part

1. Constant function Function that just return a const value must not depends on the number of elements, hence time complexity of this algorithm should be $O(1)$

2,3. Sum and product functions These functions need to iterate by every element of array at least 1 time, hence they should have time complexity $O(n)$. Since the **numpy** uses fast algorithms to solve these problems, time complexity may be a bit lower than $O(n)$ or constant C will be really low.

4. Calculate the value of a polynomial First way is: $P(x) = \sum_{k=1}^{n} v_k x^{k-1}$, i.e. evaluating terms one by one. Time complexity: $O(n)$, because it's linearly depends on number of terms.
Second way is: $P(x) = v_1 + x(v_2 + x(v_3 + ...))$. Time complexity should be also $O(n)$, but with a smaller constant.

5. BubbleSort Bubblesort is a simple sorting algorithm that repeatedly steps through the list, compares adjacent elements and swaps them if they are in the wrong order. The pass through the list is repeated until the list is sorted.
Time Complexity:
Worst-case performance: $O(n^2)$
Average performance: $O(n^2)$

6. QuickSort
Quicksort is a divide-and-conquer algorithm. It works by selecting a 'pivot' element from the array and partitioning the other elements into two sub-arrays, according to whether they are less than or greater than the pivot. The sub-arrays are then sorted recursively.
Time Complexity:
Worst-case performance: $O(n^2)$
Average performance: $O(nlogn)$

Worst case is then the pivot choosing is bad, i.e. sub-arrays have too different number of elements. For example, one element in one sub-array and the rest in the other.

7. Timsort
Timsort is a hybrid stable sorting algorithm, derived from merge sort and insertion sort, designed to perform well on many kinds of real-world data. Insertion sort is using when data amount is small. When the data is big Timsort use merge sort.
Time Complexity:
Worst-case performance: $O(nlogn)$
Average performance: $O(nlogn)$

$O(n) \sim O(const * n) \sim O(n + const) \sim O(n + n)$. This fact is used when I build plots.

4 Results

4.1 Const function

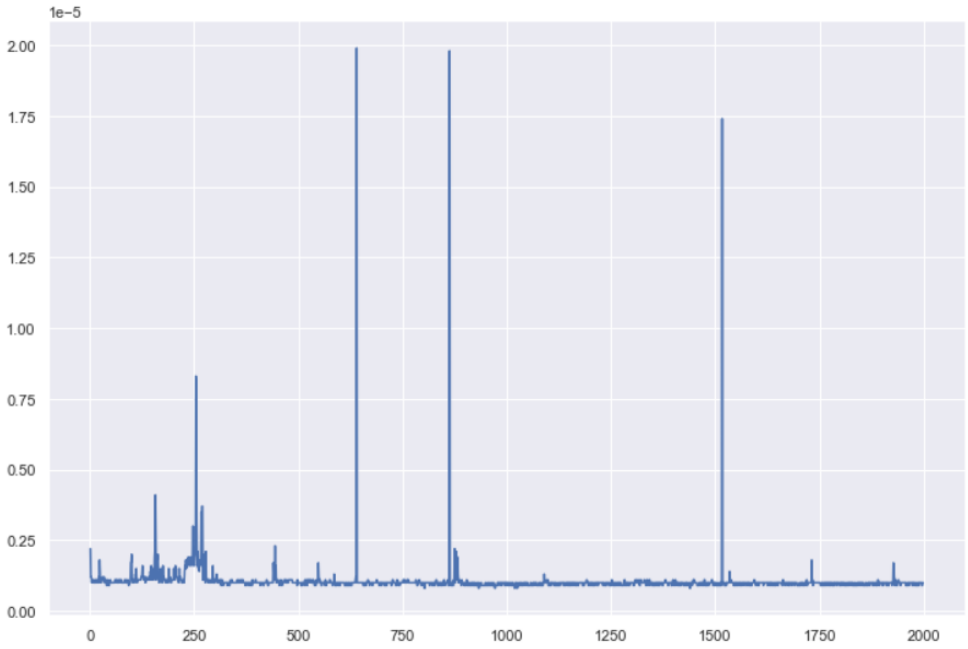

Figure 1: Const function

As we see on the Figure 1, function doesn't depend on vector size and always have approximately the same execution time, which means O(1) complexity.

4.2 Sum and product fuctions

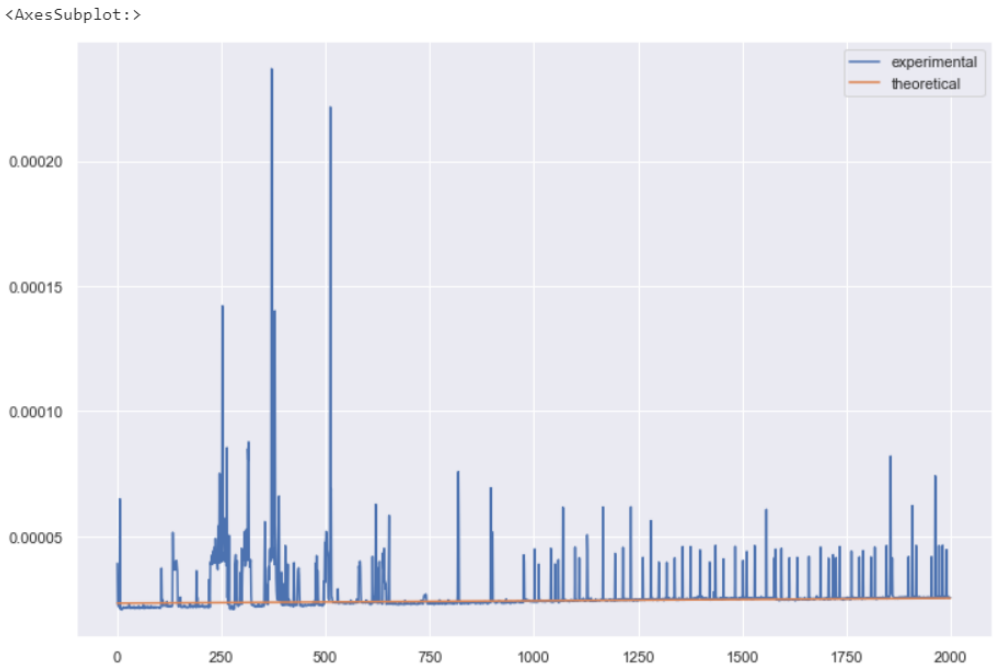

Figure 2: Sum function

Sum function must has O(n) complexity. As we see on the Figure 2 the main trend is $O(c * n) \sim O(n)$.

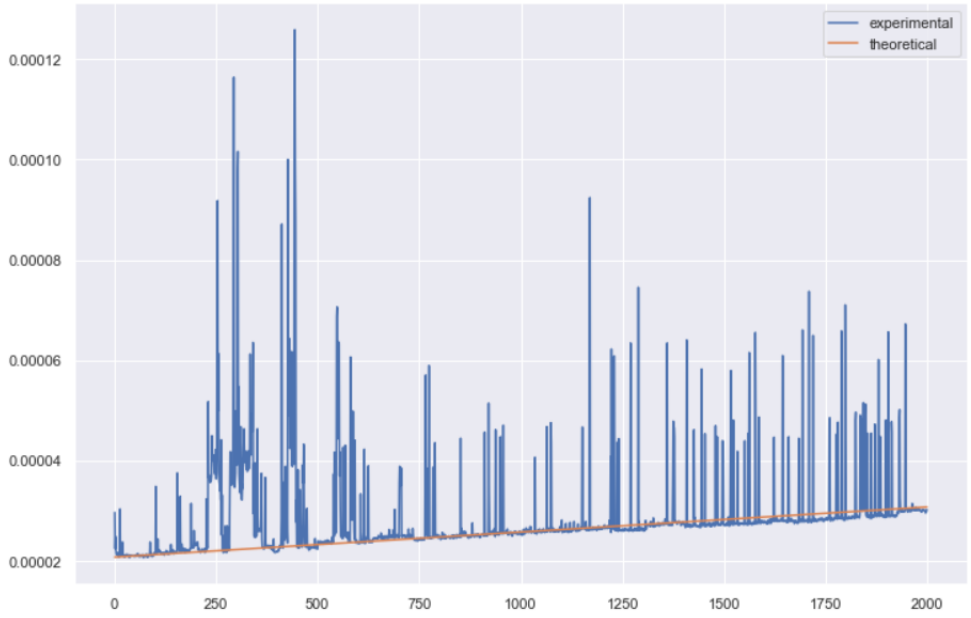

Figure 3: Product function

Conclusion is the same as it was in sum function. $O(n)$ complexity.

4.3 Calculating polynomial

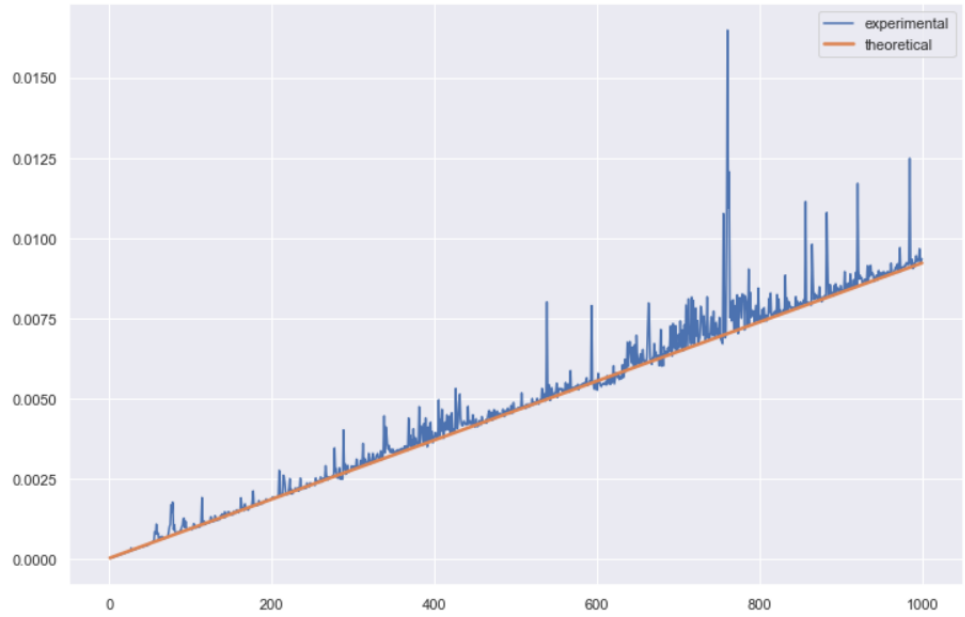

Figure 4: Naive function calculating

Time complexity is also $O(n)$.

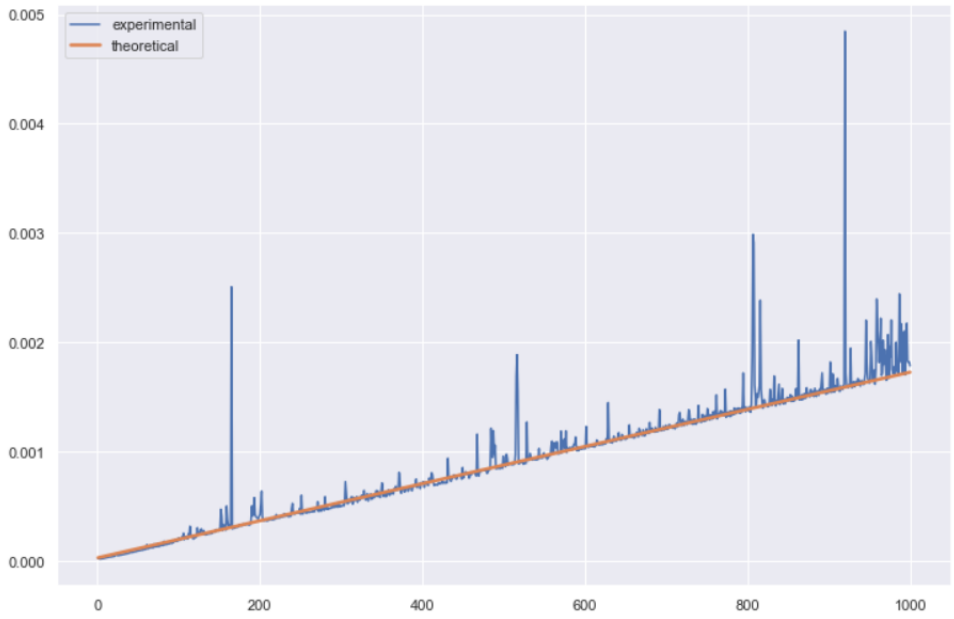

Figure 5: Gorner's Method

$O(n)$ time complexity on Figure 5.

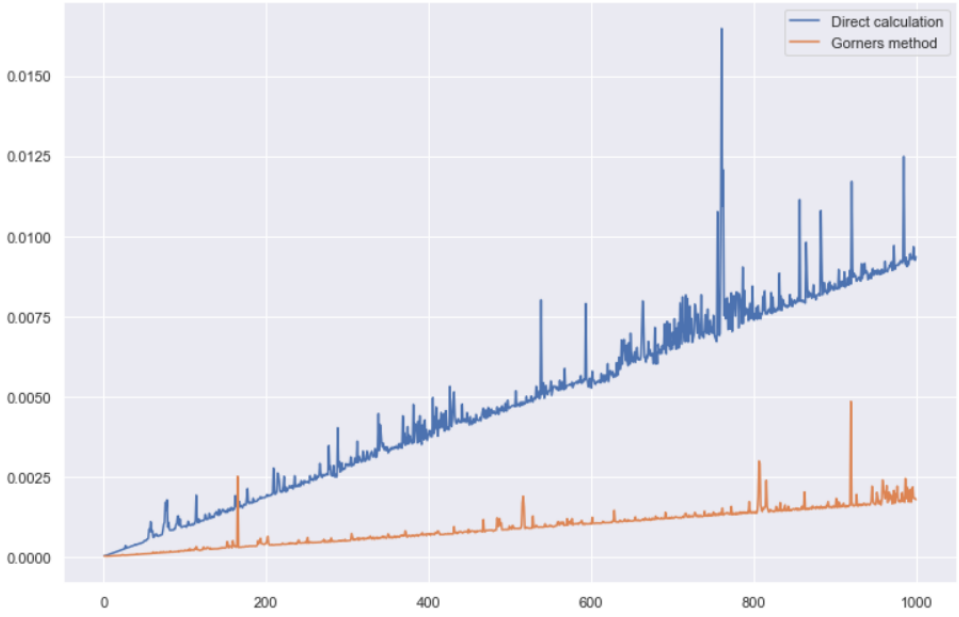

Figure 6: Comparing direct calculations and Gorner's Method

As we can see the Gorner's method calculating polynomial faster than direct calculation.

4.4 Bubble sort

Figure 7: Bubble sort

Complexity of bubble sort is $O(n^2)$. Since we see the parabola on the graph, we are right.

4.5 Quicksort

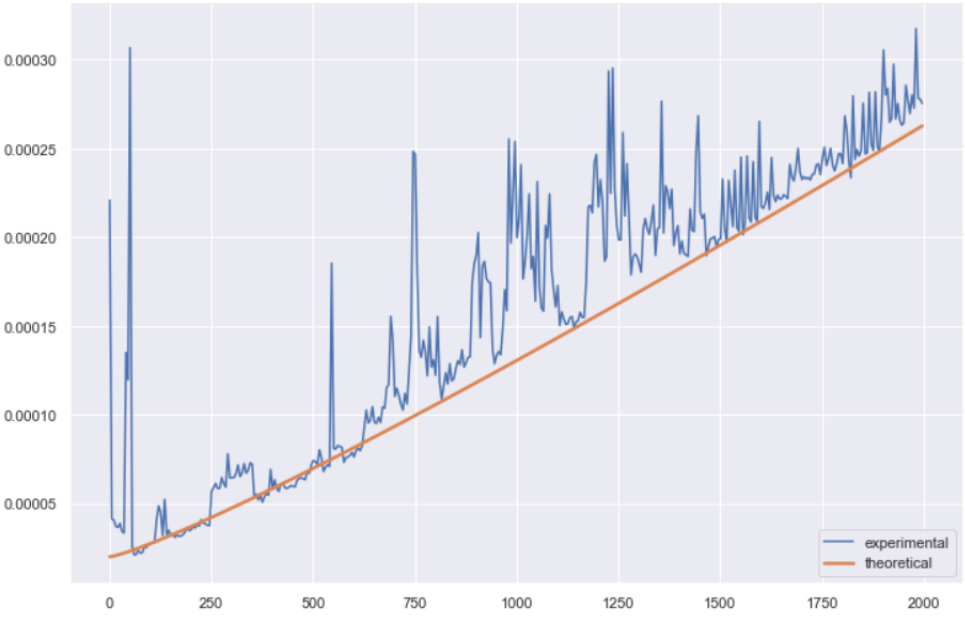

Figure 8: QuickSort

Quicksort time complexity is $O(nlogn)$.

4.6 Timsort

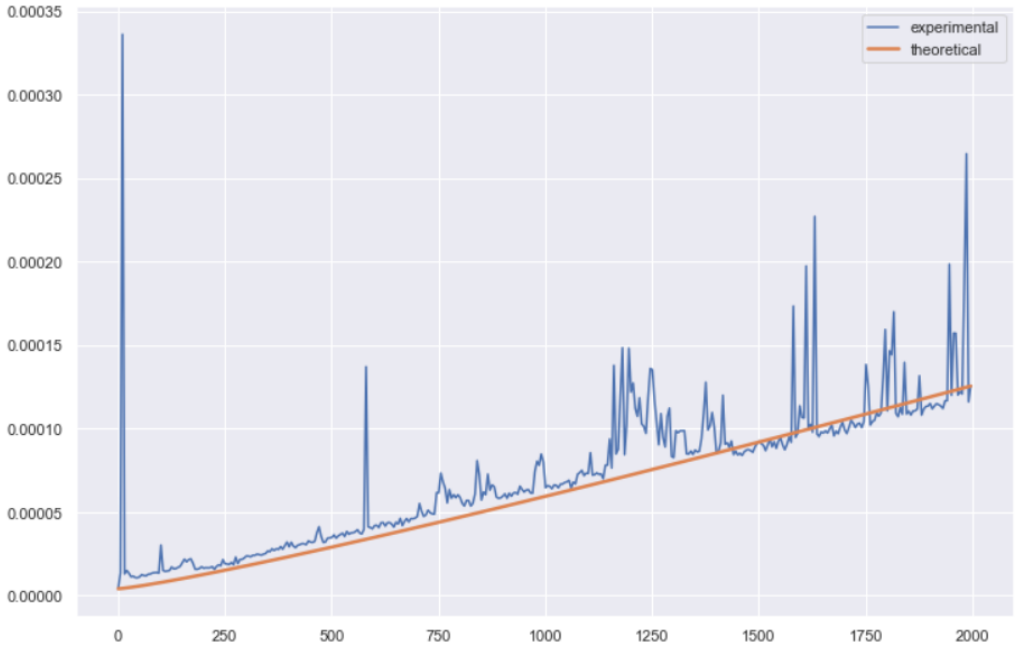

Figure 9: Timsort

Timsort complexity is $O(nlogn)$.

4.7 Comparing sort algorithms

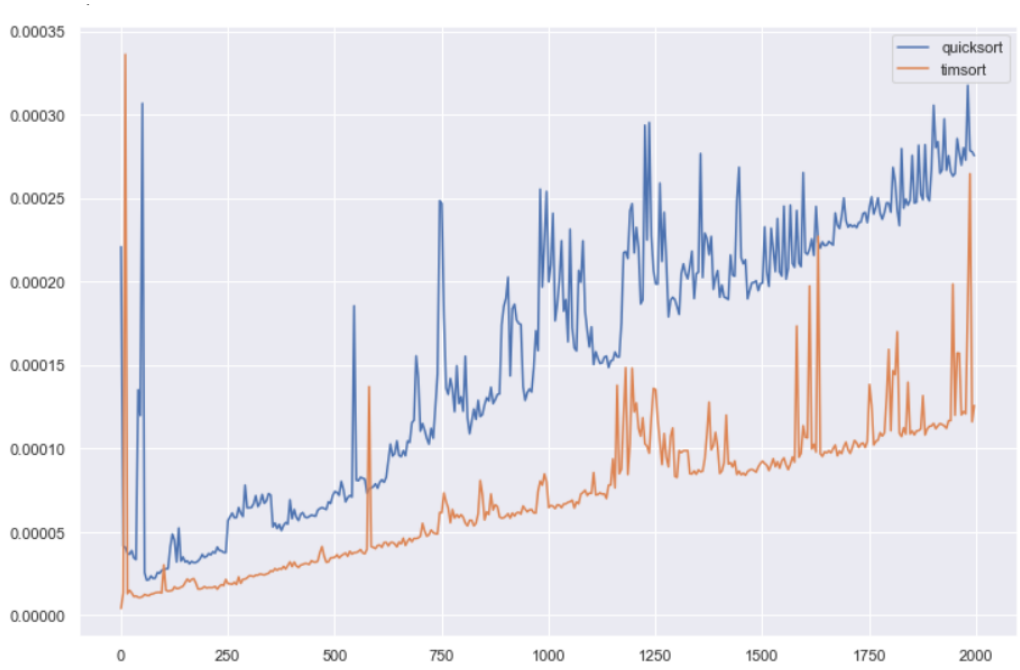

Figure 10: Timsort And quicksort

On the Figure 14 blue line is a quicksort and orange line is a timsort. As we see, timsort is faster.

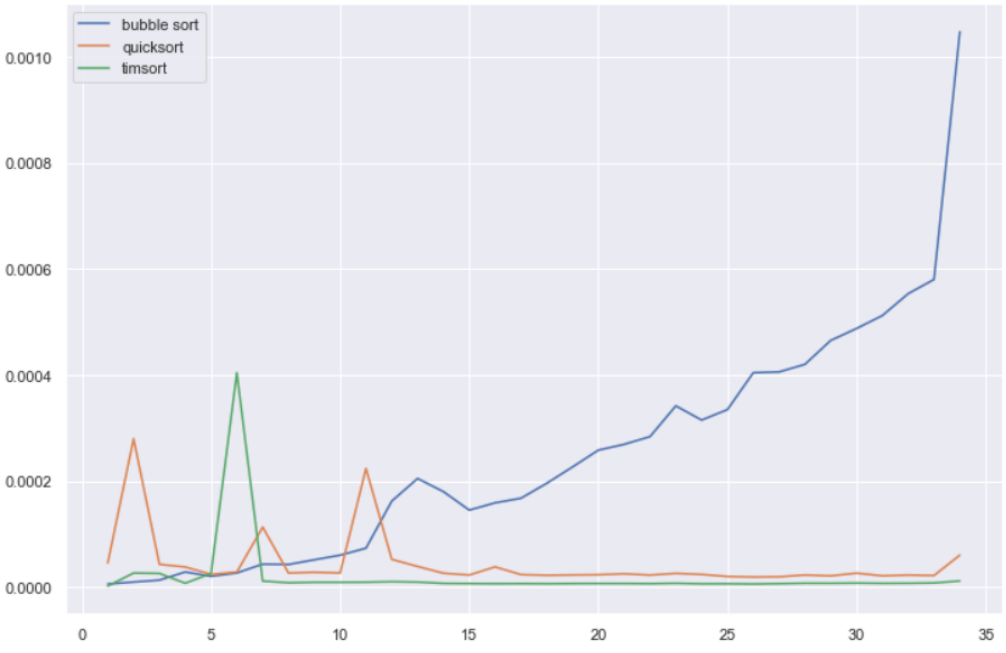

Figure 11: Bubble sort, timsort And quicksort

On the Figure 15 we compare only 1-35-sized vectors because of too much different in time values when vector size > 35.

4.8 Matrix product

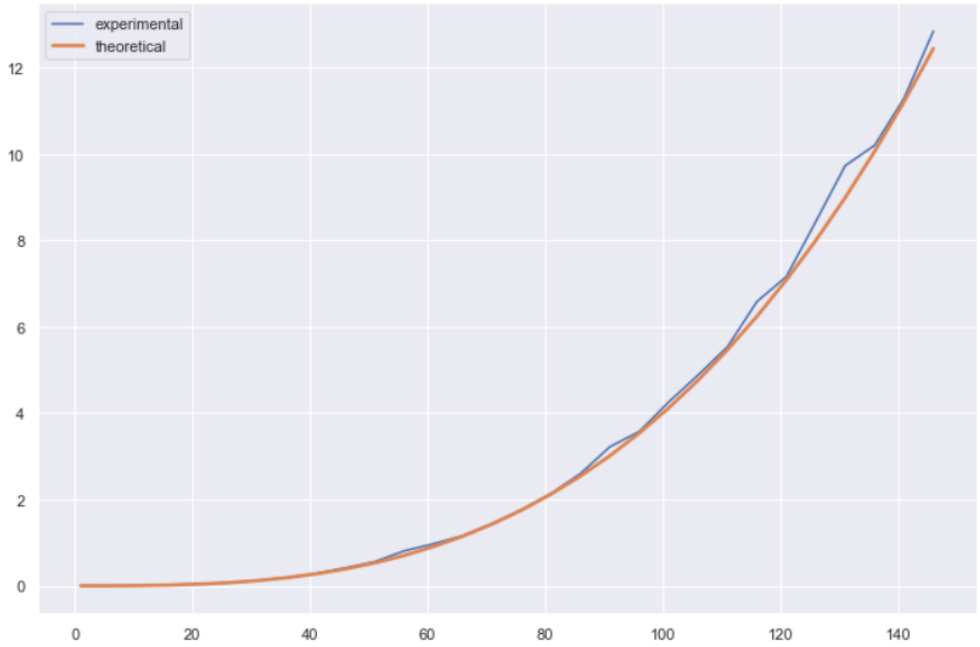

Figure 12: Matrix product

Already when the matrices size is 100x100 it's requires 4 seconds to multiply. And the complexity grows as $O(n^3)$, because of 3 loops inside the product function.

4.9 Data structures and design techniques

Every algorithm is using vector or numpy array as a data structure. Quicksort is a divide-and-conquer algorithm. Also timsort is using this paradigm when it is using merge sort. Merge sort requires $O(n)$ additional memory.

5 Conclusions

In this work I experimentally show the time complexity of all presented algorithms. To calculate the value of function you better use Gorner's method instead of direct calculation. For faster sorting you should use Timsort. If needed to calculate matrix product you better find some other ways to calculate instead of direct calculations, because it is too slow.

6 Appendix

https://github.com/trixdade/ITMO_algorithms/blob/master/Lab1/Algorithms_Lab1.ipynb