FEDERAL STATE AUTONOMOUS EDUCATIONAL INSTITUTION

OF HIGHER EDUCATION

ITMO UNIVERSITY

Report

on the practical task No. 1

"*Experimental time complexity analysis*"

Performed by

Pakulev Aleksandr

J4133c

Accepted by

Dr Petr Chunaev

St. Petersburg

2021

Brief theoretical part

In this lab needed to measure execution time and complexity costs of different functions.

Under this task, please, execute each function(algorithm) with random generated vector(or matrix) sizes n (or nXn), where n in range [1 to 2000].

Each algorithm at each iteration(n) must be executed multiple times (5 times) and each execution must be measured with timestamp(i.e. endTime - beginTime). Obtained results for each n must be averaged.

What is big O notation?

Asymptotics upper bound, i.e. for function g(x) O(g(x)) = {f(x): exists const c, x_0, for which 0 <= f(x) <= c * g(x), for all x >= x_0}, plain English: O(g(x)) such function (multiplied by constant c), which on all interval from x_0 to infinity more than some functions set (set may be uncountable).

For example O(n*n) = {n, sqrt(n), const, etc}

In this task must be implemented 8 algorithms:

1. Const function O(1)
2. Sum of elements O(n)
3. Product of elemets O(n)
4. Plainly calculated polynom , with x=1.5 O(n * log(n)), description in section bellow
5. Polynom calculated using Horner's method. O(n)
6. Bubble Sort O(n ^ 2)
7. Quick sort O(n * log(n)), in worst cases O(n ^ 2)
8. TimSort O(n * log(n))
9. Matrix multiplication O(n^3)

I suppose the first five algorithms can be related to Brute-force technique,

because they are the simplest algorithms that handle all available variants, and there aren't many of them (Only one).

Bubble sort can be related to Brute-force technique, because we iterate over all reasonable variants (i.e. pairs in our cases).

Quick sort and TimSort can be related to Divide and Conquer technique, because on each step we divide the array by more less parts, sort that parts, and merge them back to the initial array. And we do that recursively.

Matrix multiplication may be related to Brute-force because we can just iterate over all available pairs of two matrices.

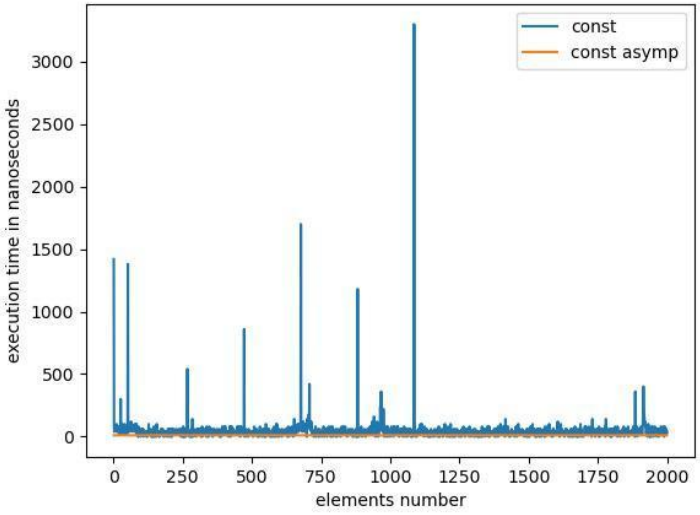
Result

In this section presented received results, and given some description about plots.

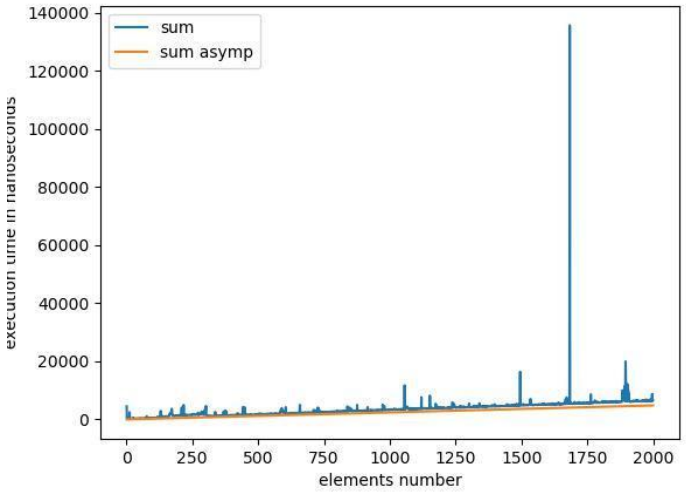
For Implementing this task I used Python3.

Graphs bellos have x-axis which means iteration number (elements in array or matrix) and y-axis which means time in nanoseconds, which was spent for average execution. Additionally each graphs contains

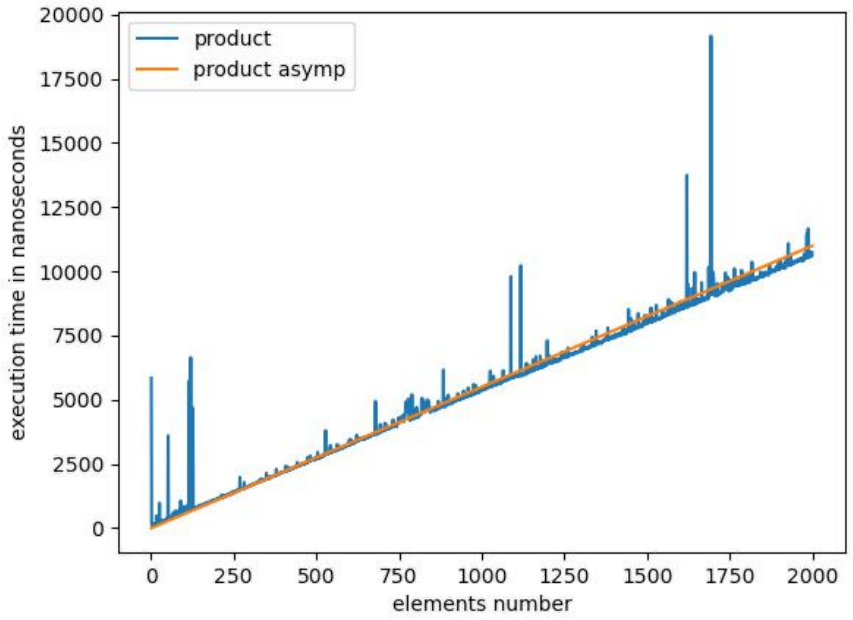
Algorithm executing polynomial function work with time complexity equals quick sort because POW function has O(n * log(n)) time complexy



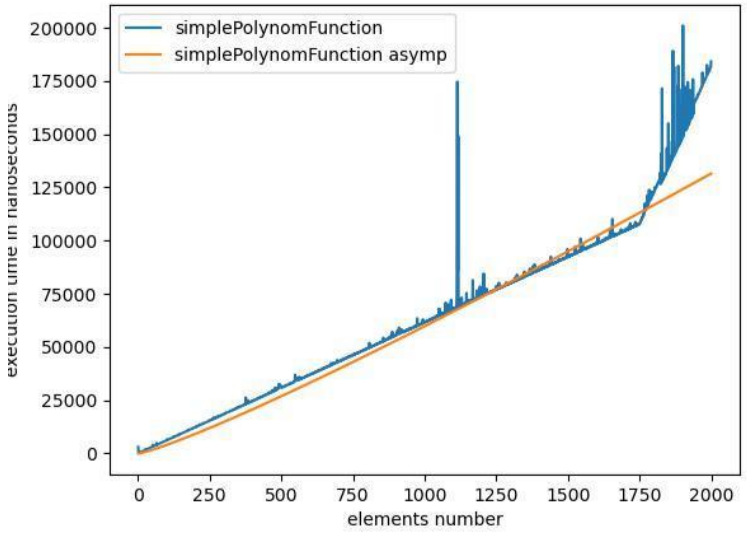
1. const function



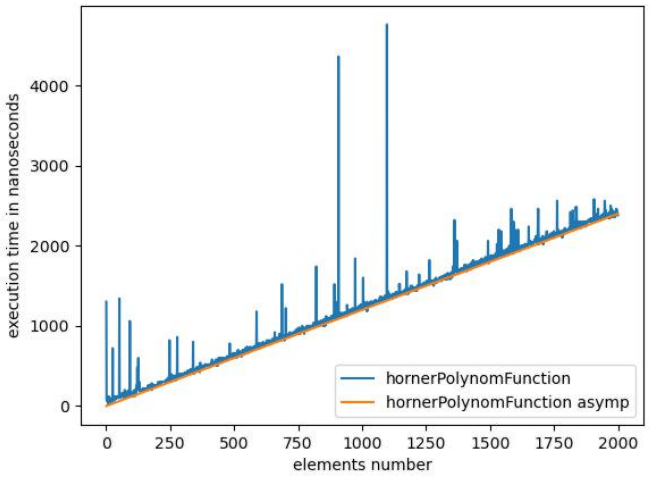
2. sum function



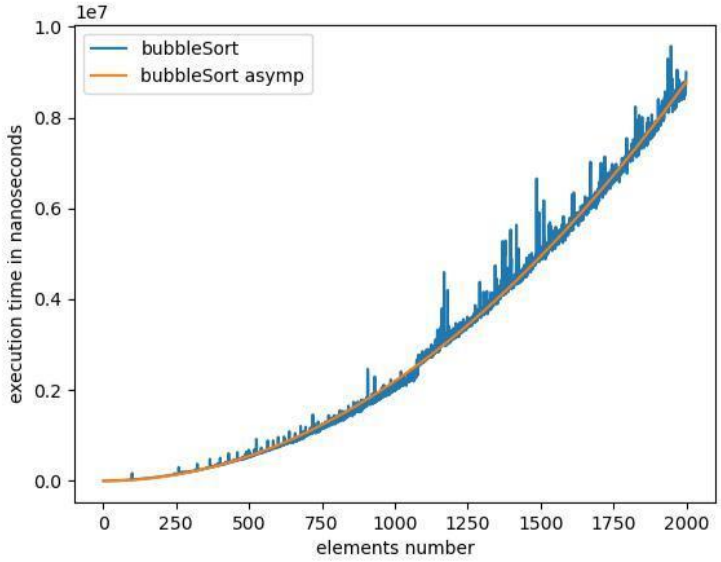
3. product function



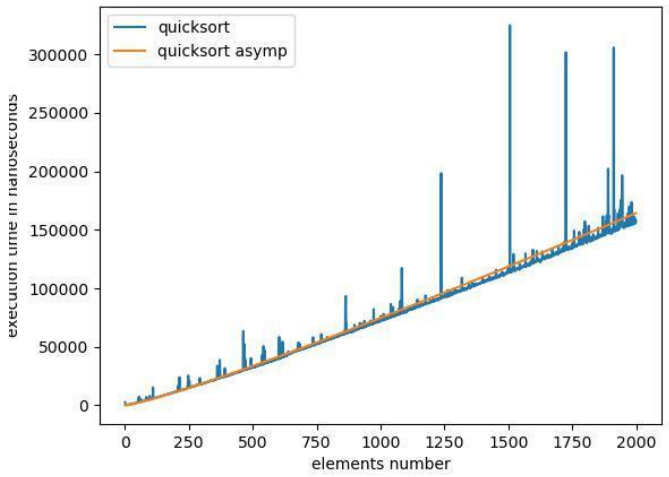
4. polynom function



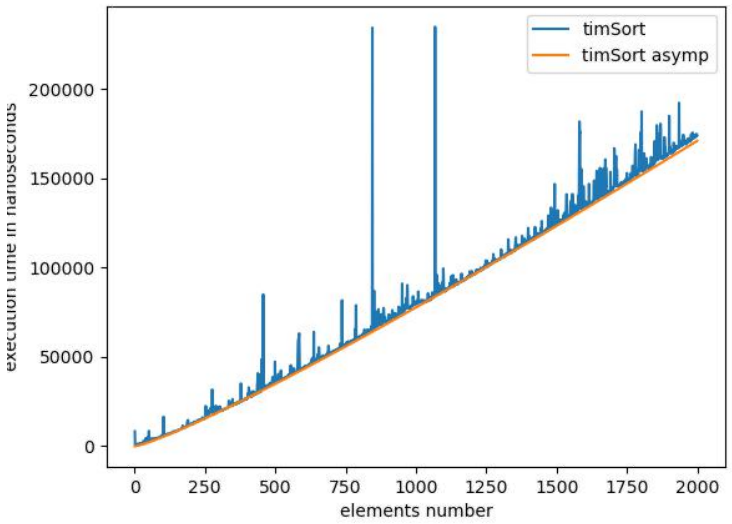
5. horner method



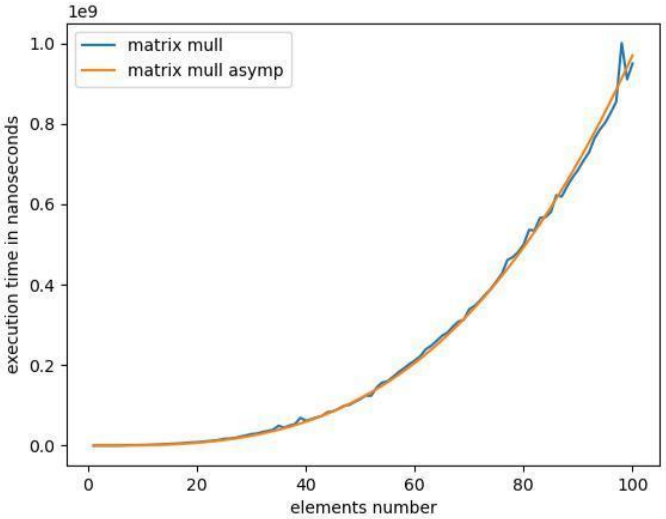
6. bubble sort



7. quick sort



8. tim sort



9. matrix multiplication

In the graphs above the blue plot corresponded evaluated algorithm time complexity, and the other line (orange) corresponded asymptotics complexity.

Conclusions

In this task I measured time execution of a lot of different algorithms.
Here will be some overview.

1. Const function, as told above, has complexity O(1), and on first plot times in nanoseconds don't change since increasing N.

2. Sum elements have complexity O(n), and time in nanoseconds spent on evaluating increased linearing.

3. Product elements have complexity O(n), and time in nanoseconds spent on evaluating increased linearing.

4. polynomial which using horner schema elements have complexity O(n), and time in nanoseconds spent on evaluating increased linearing.

4. plan polynomial have complexity O(n * log(n)), and time in nanoseconds spent on evaluating increased faster then linear, but slower then quadratic speed.

5. Tim sort have complexity O(n * log(n)), and time in nanoseconds spent on evaluating increased faster then linear, but slower then quadratic speed.

6. quick sort have complexity O(n * log(n)), and time in nanoseconds spent on evaluating increased faster then linear, but slower then quadratic speed.

7. bubble sort has complexity O(n ^ 2), and time in nanoseconds spent on evaluating increases with quadratic speed.

8. matrix multiplication has complexity O(n ^ 3), and time in nanoseconds spent on evaluating increases with cubial speed.

Conclusion is next: time complexity equals algorithm complexity multiplied by constant value.