FEDERAL STATE AUTONOMOUS EDUCATIONAL INSTITUTION
OF HIGHER EDUCATION
ITMO UNIVERSITY

Report

on the practical task No. 1

«Experimental time complexity analysis»

Performed by:
Putnikov Semyon
J4132c

Accepted by:
Dr Petr Chunaev

Санкт-Петербург
2020

1. Goal

Experimental study of the time complexity of different 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.

1. Generate an n-dimensional random vector $\vec{v}$ with non-negative elements. For $\vec{v}$ implement the following calculations and algorithms:
   (a) $f(v) = const$ *(constant function)*
   (b) $f(v) = \sum_{k=1}^{n} v_k$ *(the sum of elements)*
   (c) $f(v) = \prod_{k=1}^{n} v_k$ *(the product of elements)*
   (d) supposing that the elements of $\vec{v}$ 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 $P(x) = v_1 + x(v_2 + x(v_3 + ...))$;
   (e) Bubble Sort of the elements of $\vec{v}$;
   (f) Quick Sort of the elements of $\vec{v}$;
   (g) Timsort of the elements of $\vec{v}$;
2. Generate random matrices A and B of size $n \times n$ with non-negative elements. Find the usual matrix product for A and B.
3. Describe the data structures and design techniques used within the algorithms.

3. Brief theoretical part

3.1 Horner's method

Horner's method is a method for approximating the roots of polynomials that was described by William George Horner in 1819.

Given the polynomial $p(x) = \sum_{k=1}^{n} a_i x^i = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + ... + a_n x^n$ where $a_0, ...a_n$ are constant coefficients, the problem is to evaluate the polynomial at a specific value x_0 of x.

For this, a new sequence of constants is defined recursively as follows:

$$
\begin{aligned}
b_n &:= a_n \\
b_{n-1} &:= a_{n-1} + b_n x_0 \\
&\vdots \\
b_1 &:= a_1 + b_2 x_0 \\
b_0 &:= a_0 + b_1 x_0
\end{aligned}
$$

Then b_0 is the value of $p(x_0)$. To see why this works, the polynomial can be written in the form

$$p(x) = a_0 + x\Bigg(a_1 + x\bigg(a_2 + x\Big(a_3 + ... + x\big(a_{n-1} + x a_n\big)...\Big)\bigg)\Bigg)$$

3.2 Bubble sort

3.3 Quick sort

3.4 Timsort

4. Results

4.1 Constant function (subtask a)

At Figure 1 we can see execution time of constant function. This plot tends to direct line as expected, but have some outliers. It is a result of cpu overheating which affects execution process.

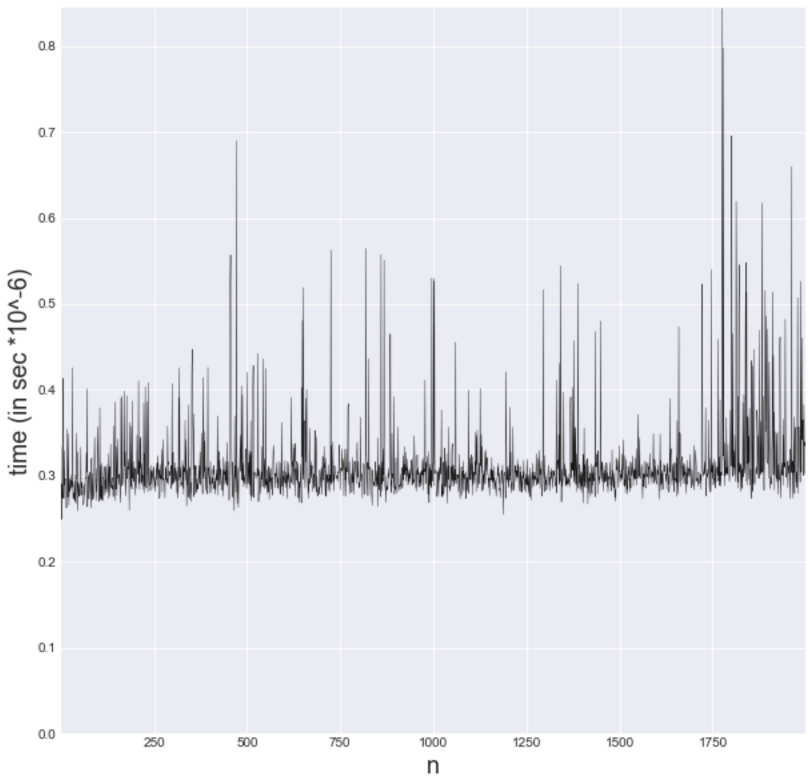

Figure 1: Plot of constant function.

Nevertheless, the graph proofs that the execution time of constant function does not depend on the number of variables.

4.2 The sum of elements (subtask b)

2

4.3 The product of elements (subtask c)

3

4.4 Polynom calculation (subtask d)

4 5

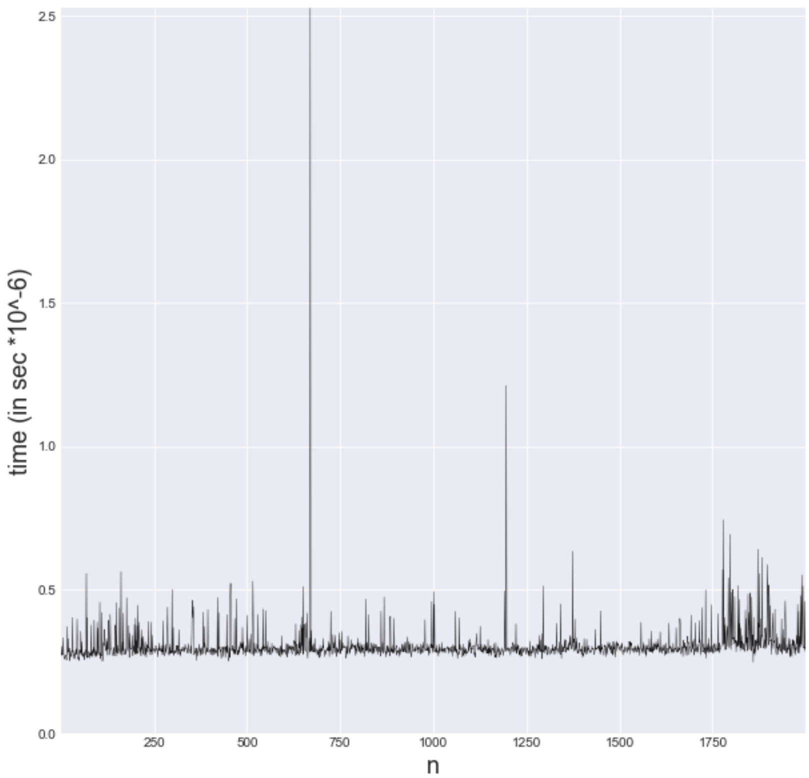

Figure 2: Plot for sum of elements.

5. Conclusions

6. Appendix

```
def constant(vector,value):
  return 0
```

Listing 1: Constant function

```
def sumOfVector(vector, value):
  return sum(vector)
```

Listing 2: Sum of elements

```
def productOfVector(vector, value):
  return numpy.prod(vector)
```

Listing 3: Product of elements

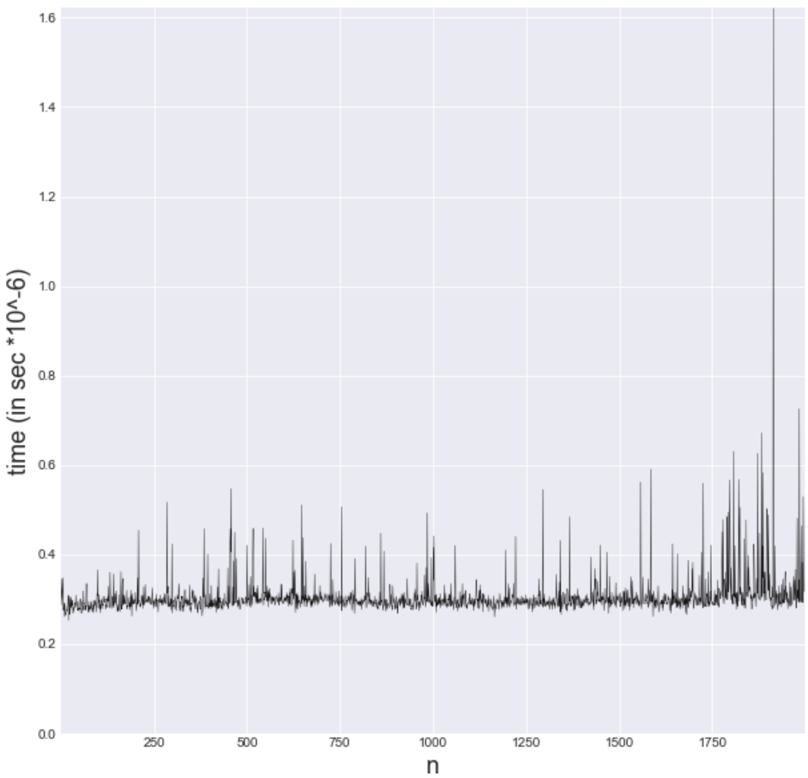

Figure 3: Plot for product of elements.

```python
def calculatePolinom(vector, value):
  result = 0
  power = 0
  for item in vector:
    result += item*(value**power)
    power += 1
  return result
```

Listing 4: Direct polinom calculation

```python
def horners(vector,value):
  v = list(vector)
  v.reverse()
  def hRec(v, value):
    if (len(v)==1):
      return v.pop()
    else:
      return v.pop() + value*hRec(v,value)
```

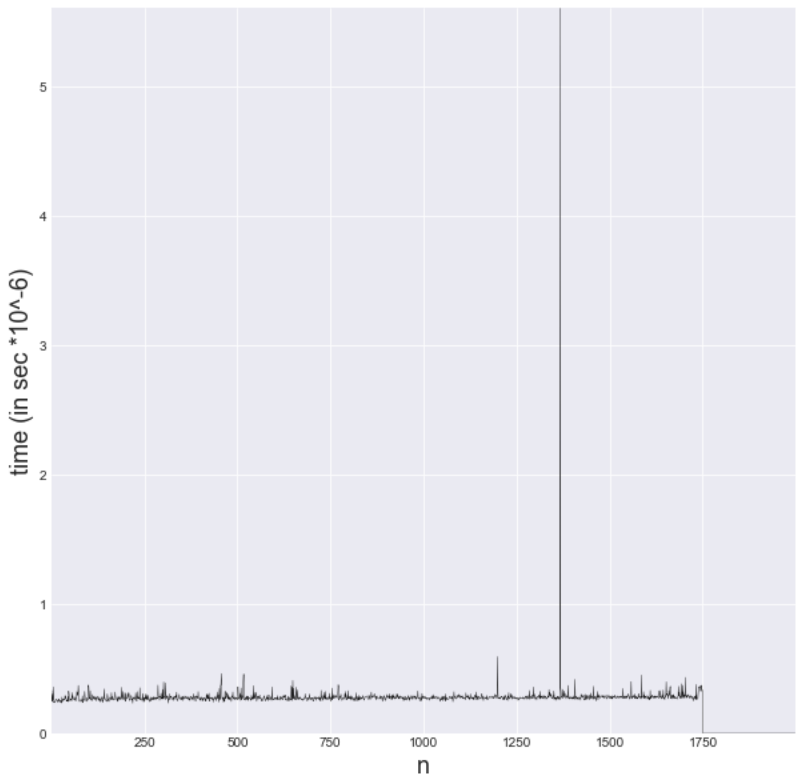

Figure 4: Plot for direct polinom calculation.

```
    return hRec(v,value)
```

Listing 5: Polinom calculation by Horner's method

```
def bubbleSort(vector,value):
  array = list(vector)
  size = len(array)
  for i in range(size-1):
    for j in range(size-i-1):
      if array[j] > array[j+1]:
        array[j], array[j+1] = array[j+1], array[j]
  return array
```

Listing 6: Bubble sort

```
def quickSort(vector, value):
  array = list(vector)
  if len(array) <= 1:
    return array
  else:
```

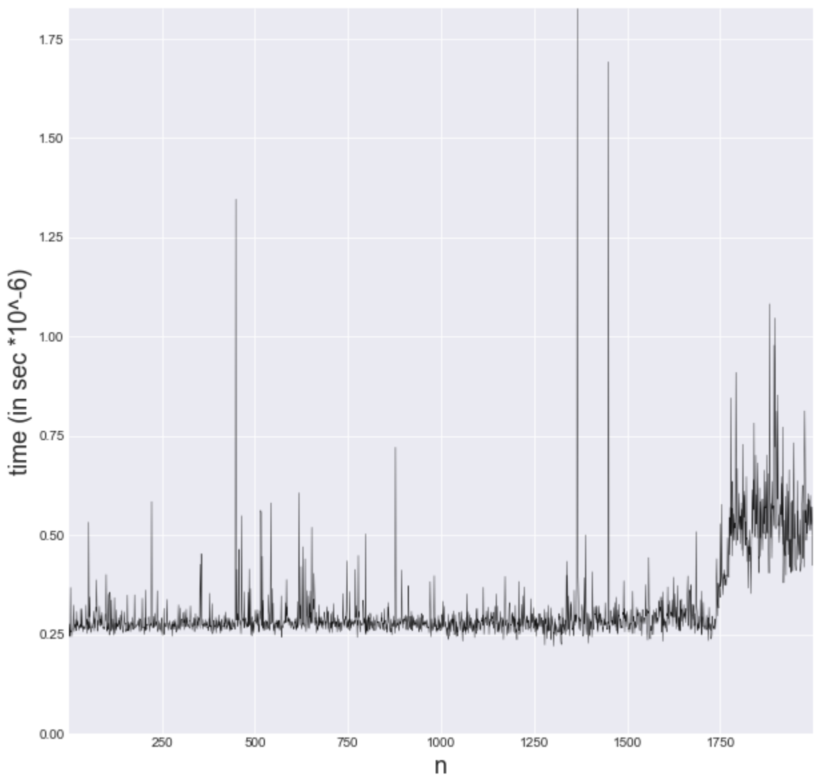

Figure 5: Plot for polinom calculation by Horner's method.

```
    q = random.choice(array)
    s_nums = []
    m_nums = []
    e_nums = []
    for n in array:
      if n < q:
        s_nums.append(n)
      elif n > q:
        m_nums.append(n)
      else:
        e_nums.append(n)
    return quickSort(s_nums, value) + e_nums + quickSort(m_nums, value)
```

Listing 7: Bubble sort