Министерство образования Российской Федерации

Пензенский государственный университет

Кафедра «Вычислительной техники»

**Отчет**

По лабораторной работе №2

По курсу «Логика и основы алгоритмизации в инженерных задачах»

На тему «Оценка времени выполнения программ»

Выполнила студентка гр.20ВВ1

Тюгаева К.А.

Проверили:

Акифьев И. В.

Юрова О. В.

Пенза, 2021

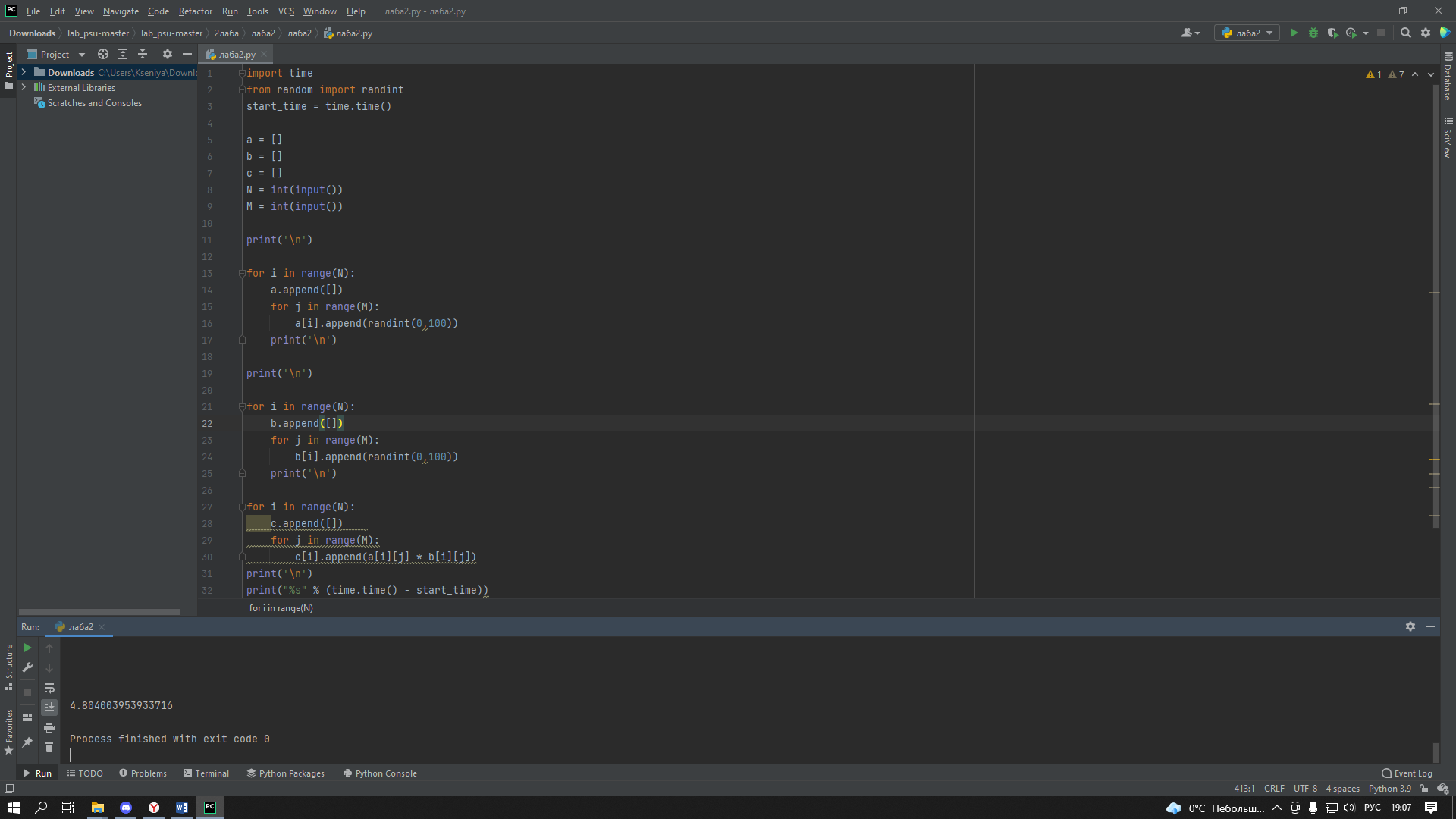
**Цель:** оценить время исполнения работы программы, реализованной разными алгоритмами. Научиться вычислять порядок сложности программы.

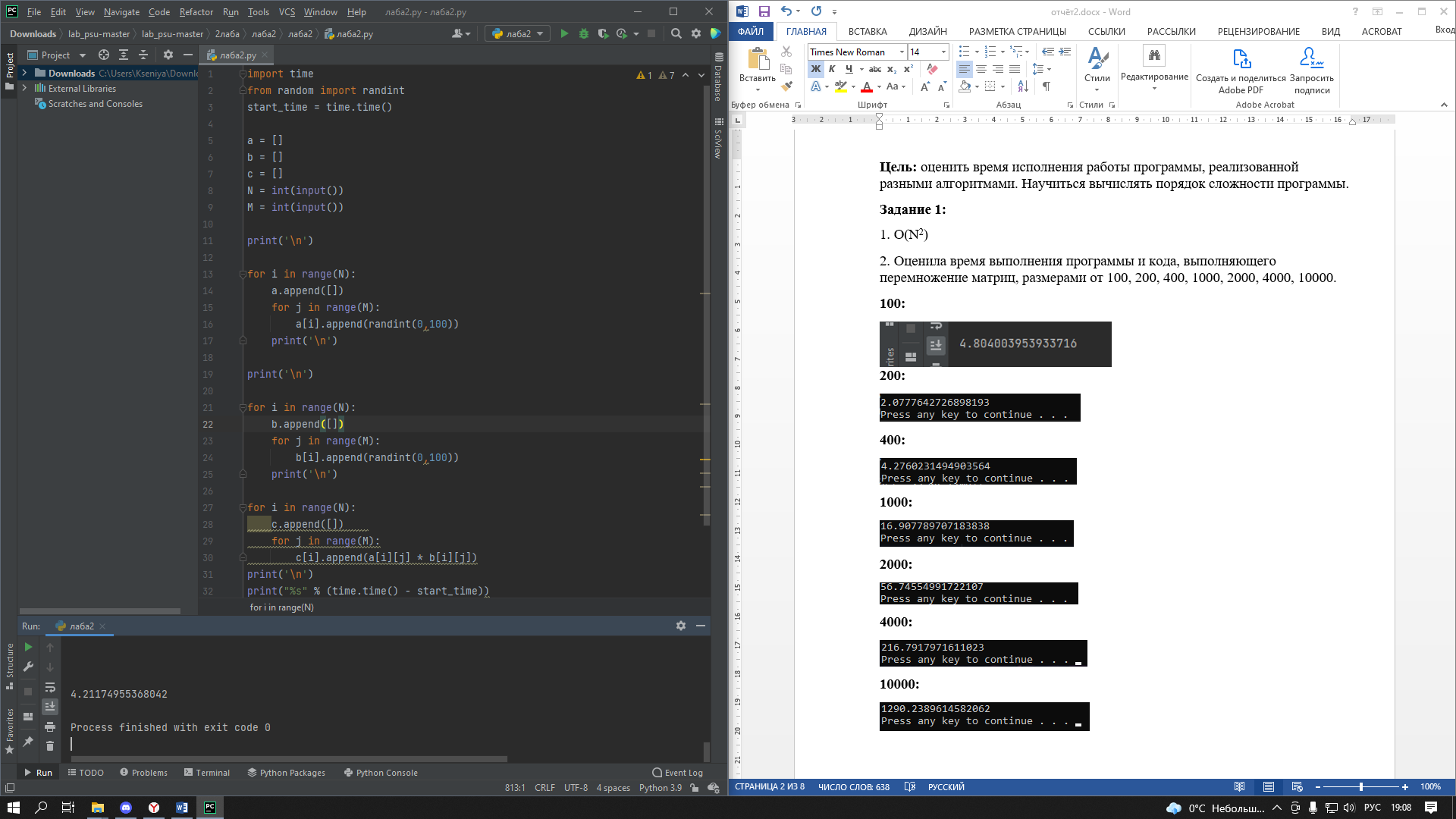
**Задание 1:**

1. O(N2)

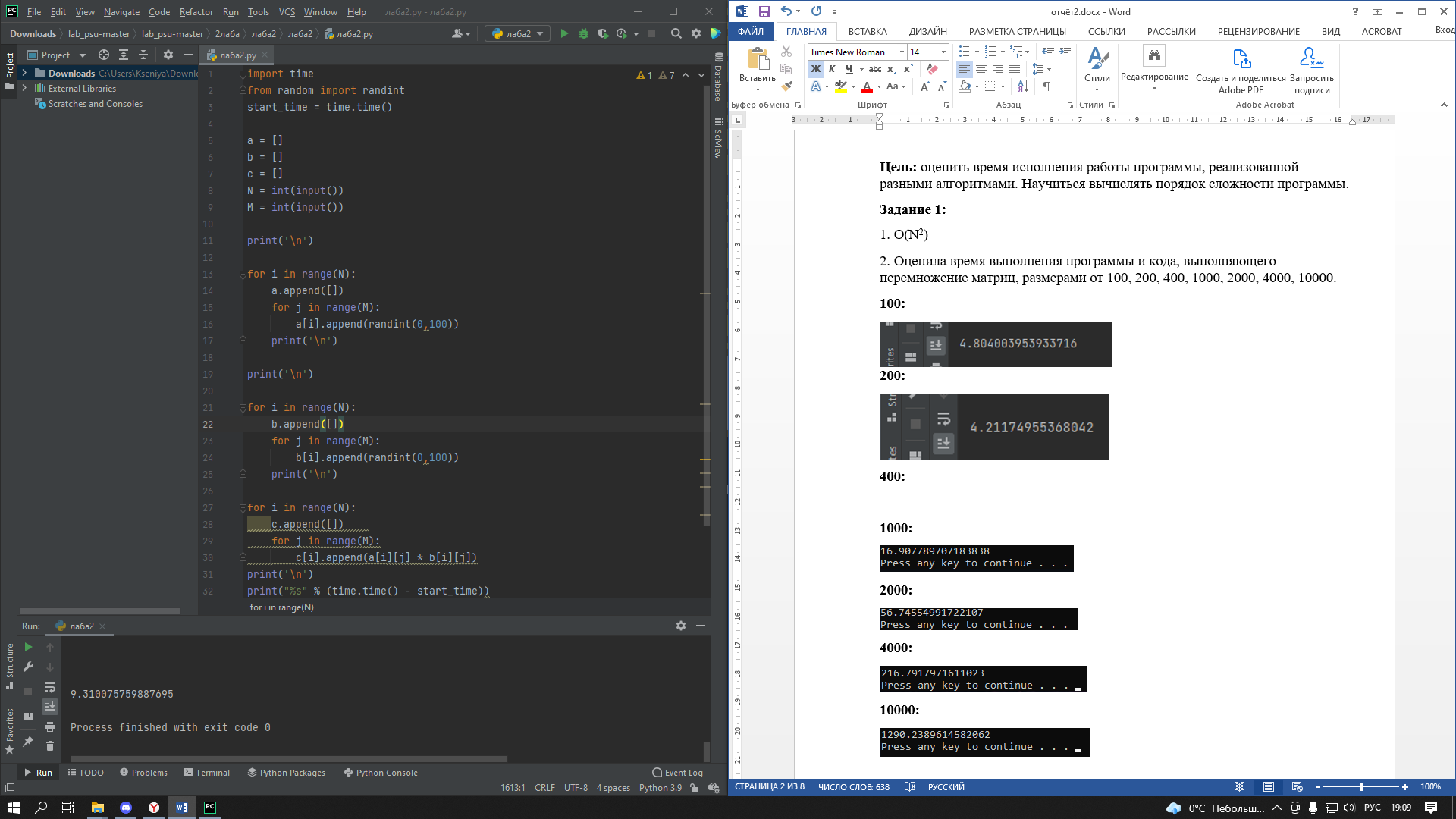
2. Оценила время выполнения программы и кода, выполняющего перемножение матриц, размерами от 100, 200, 400, 1000, 2000, 4000, 10000.

**100:**

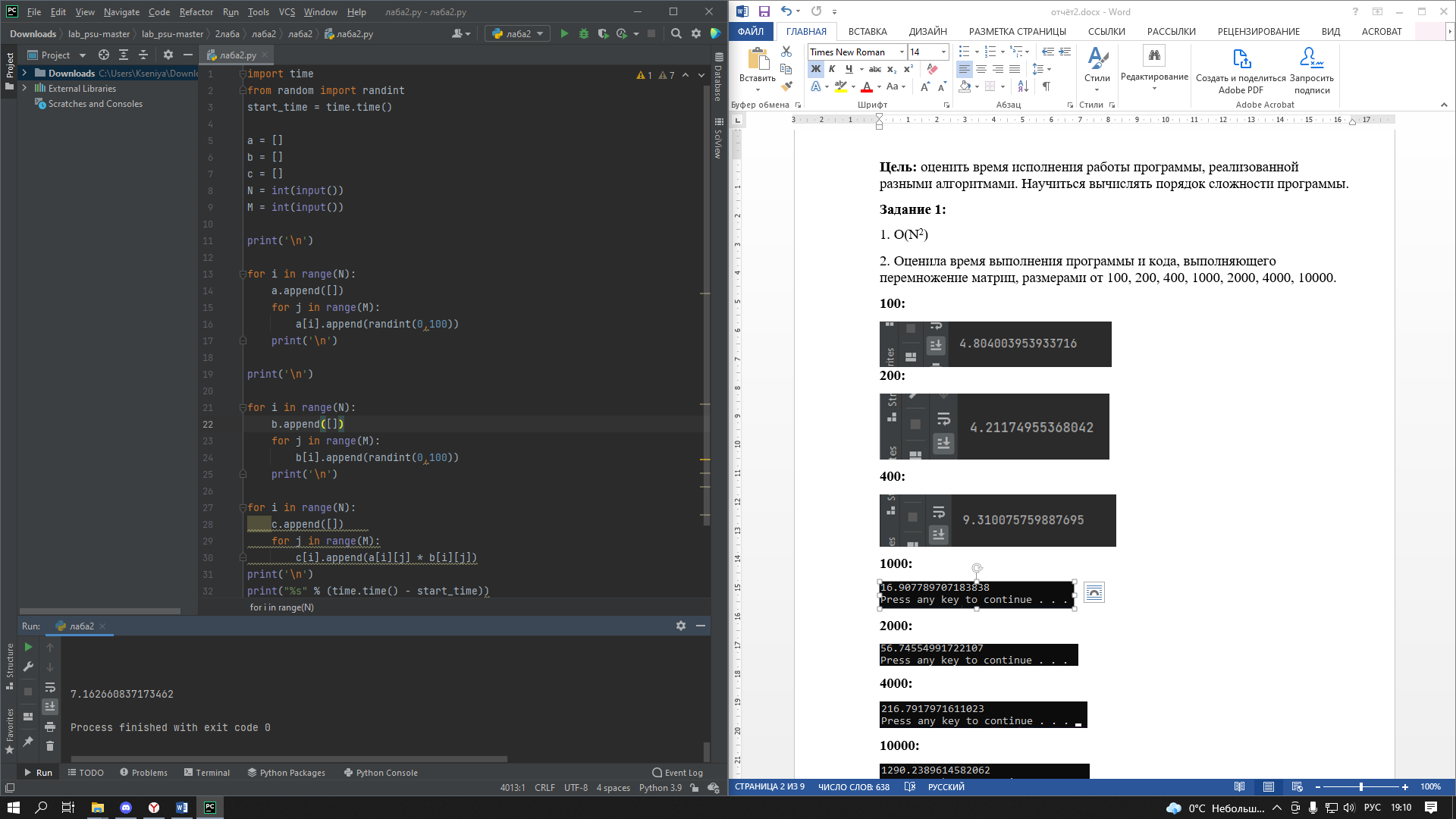
 **200:**



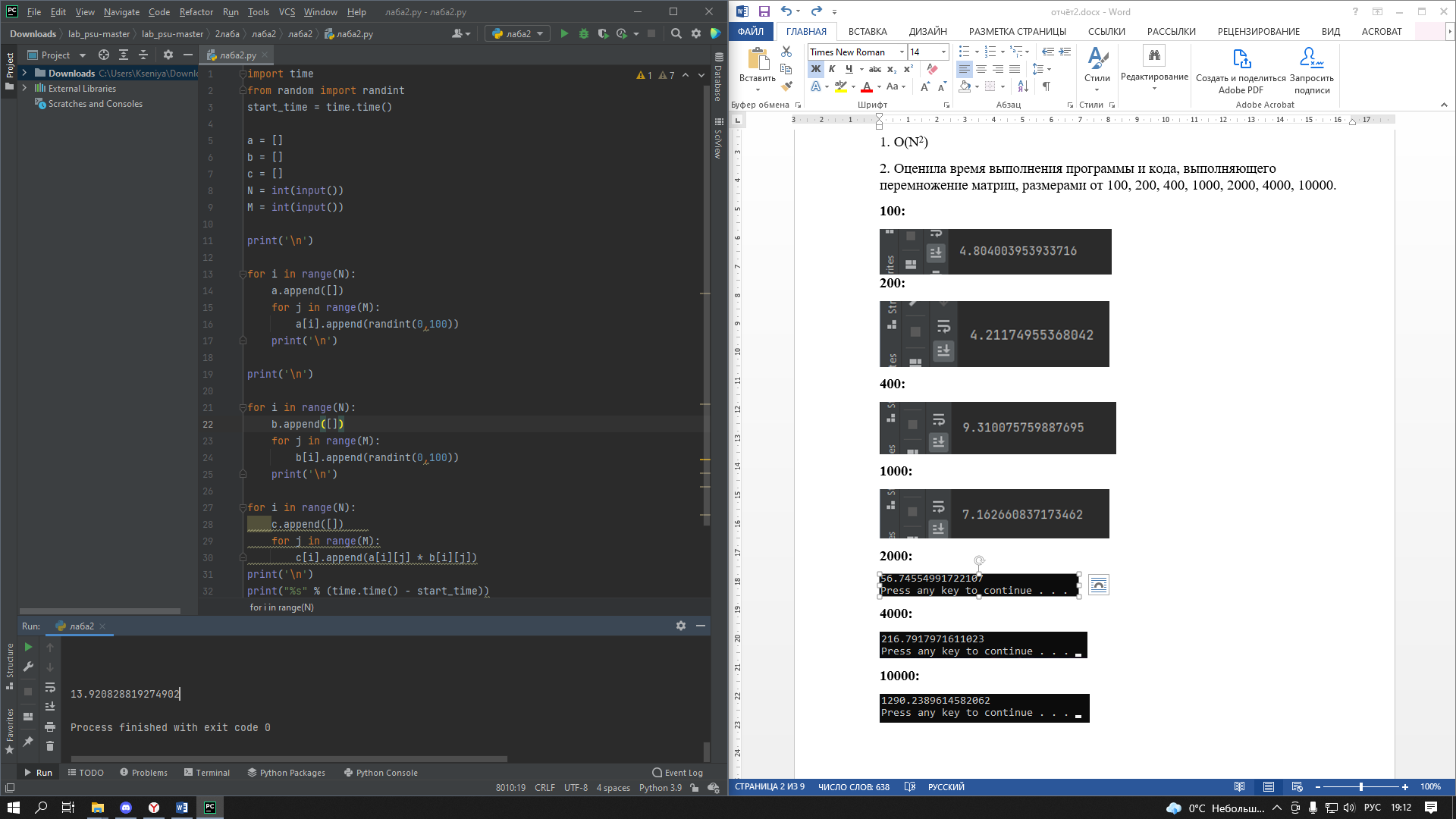
**400:**



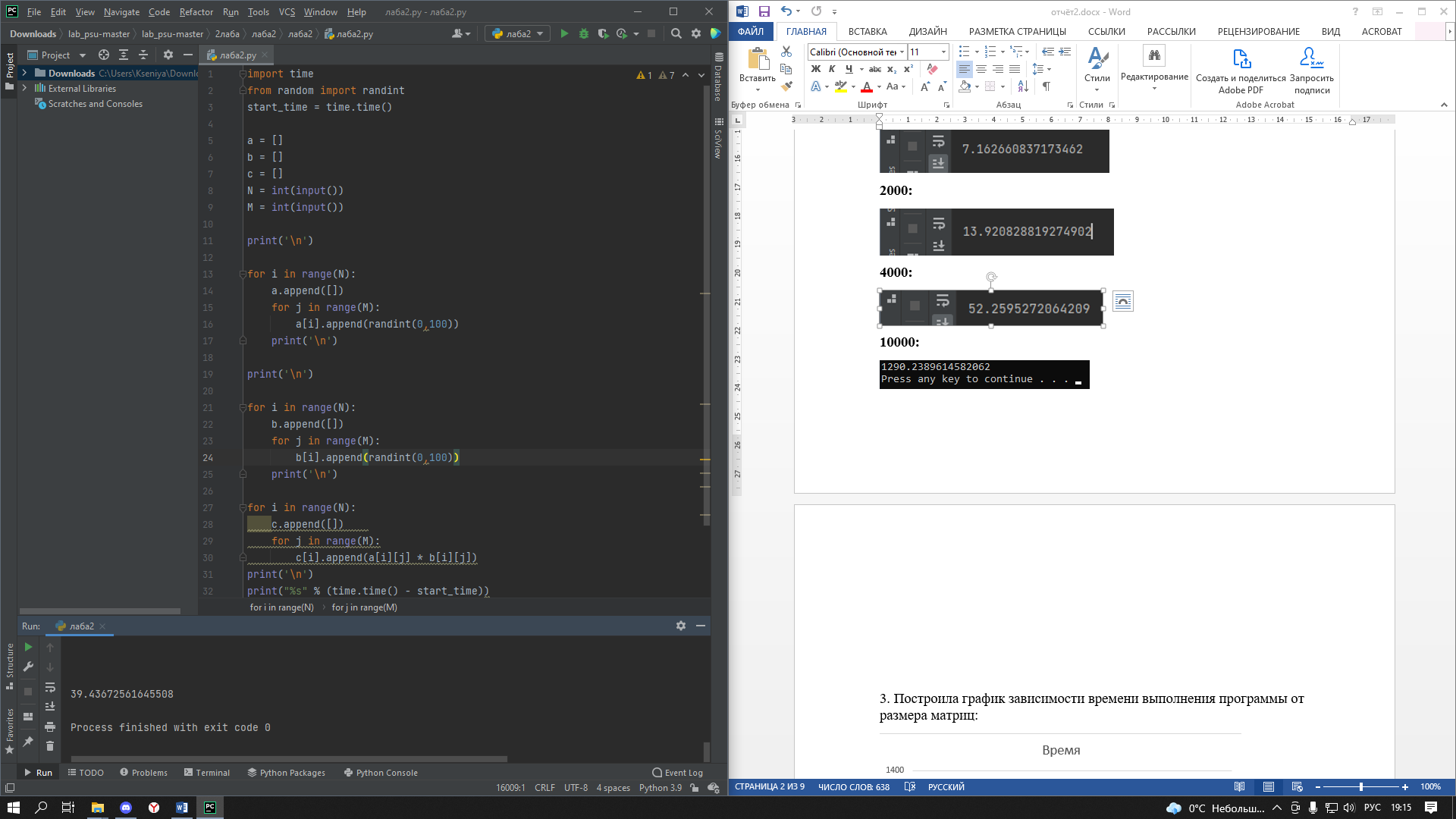
**1000:**



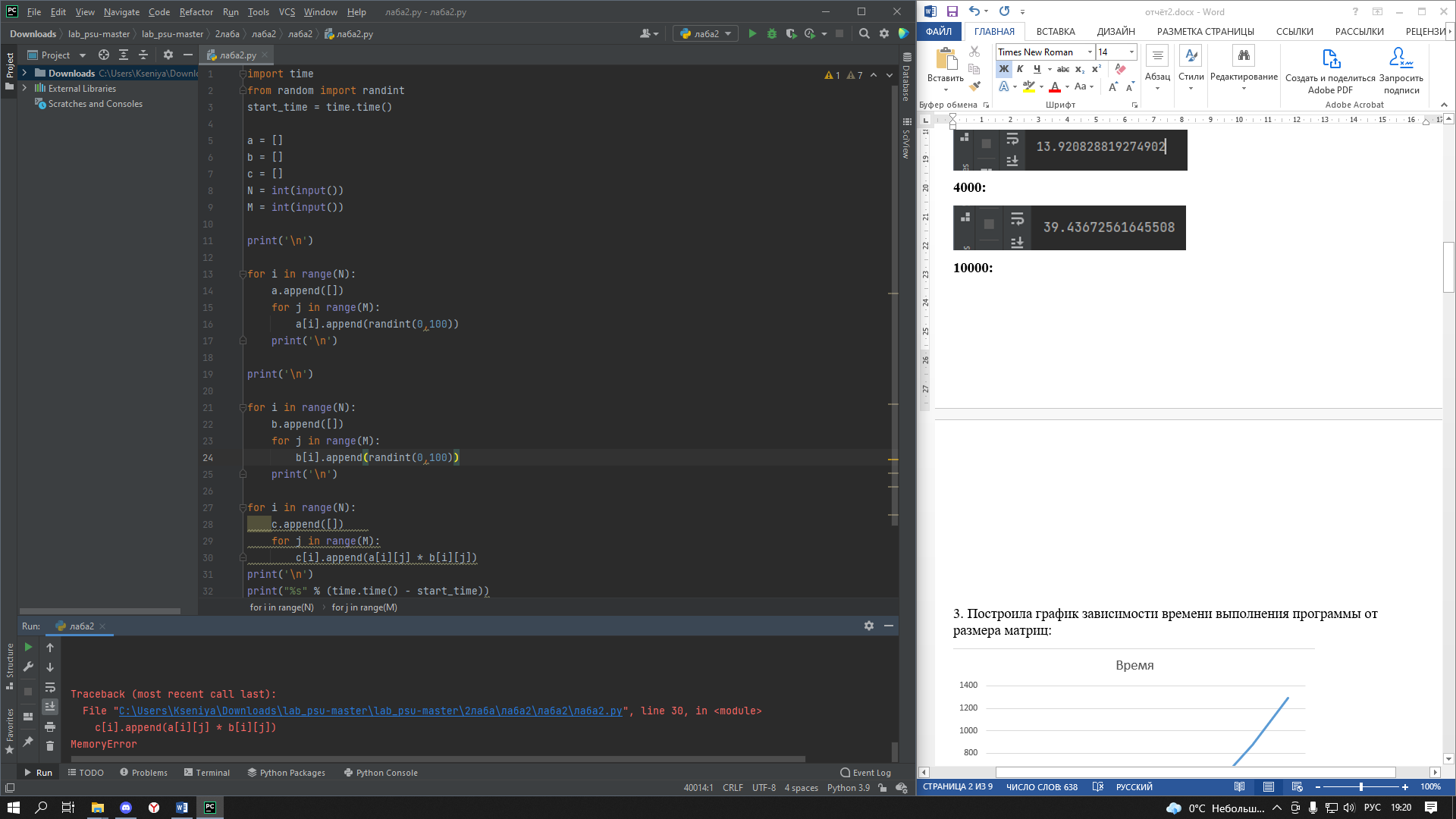
**2000:**



**4000:**



**10000:**



3. Построила график зависимости времени выполнения программы от размера матриц:

График соответствует теоретической оценки сложности программы - O(N2).

**Листинг задания 1:**

import time  
from random import randint  
start\_time = time.time()   
  
a = []  
b = []  
c = []  
N = int(input())  
M = int(input())  
  
print('\n')  
  
for i in range(N):  
 a.append([])  
 for j in range(M):  
 a[i].append(randint(0,100))  
 print('\n')  
  
print('\n')  
  
for i in range(N):  
 b.append([])  
 for j in range(M):  
 b[i].append(randint(0,100))  
 print('\n')  
  
for i in range(N):  
 c.append([])   
 for j in range(M):  
 c[i].append(a[i][j] \* b[i][j])  
print('\n')  
print("%s" % (time.time() - start\_time))

**Задание 2:**

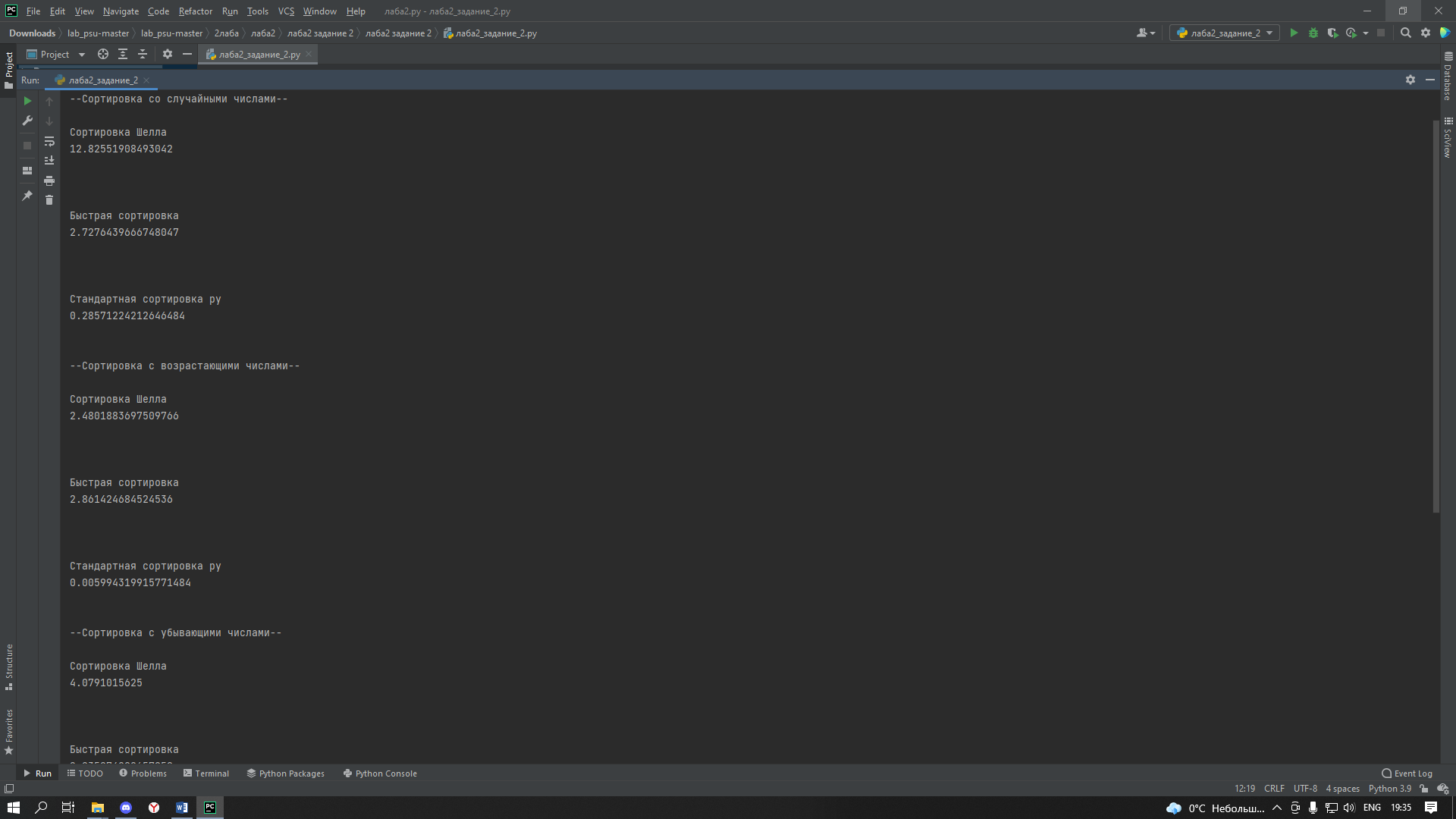
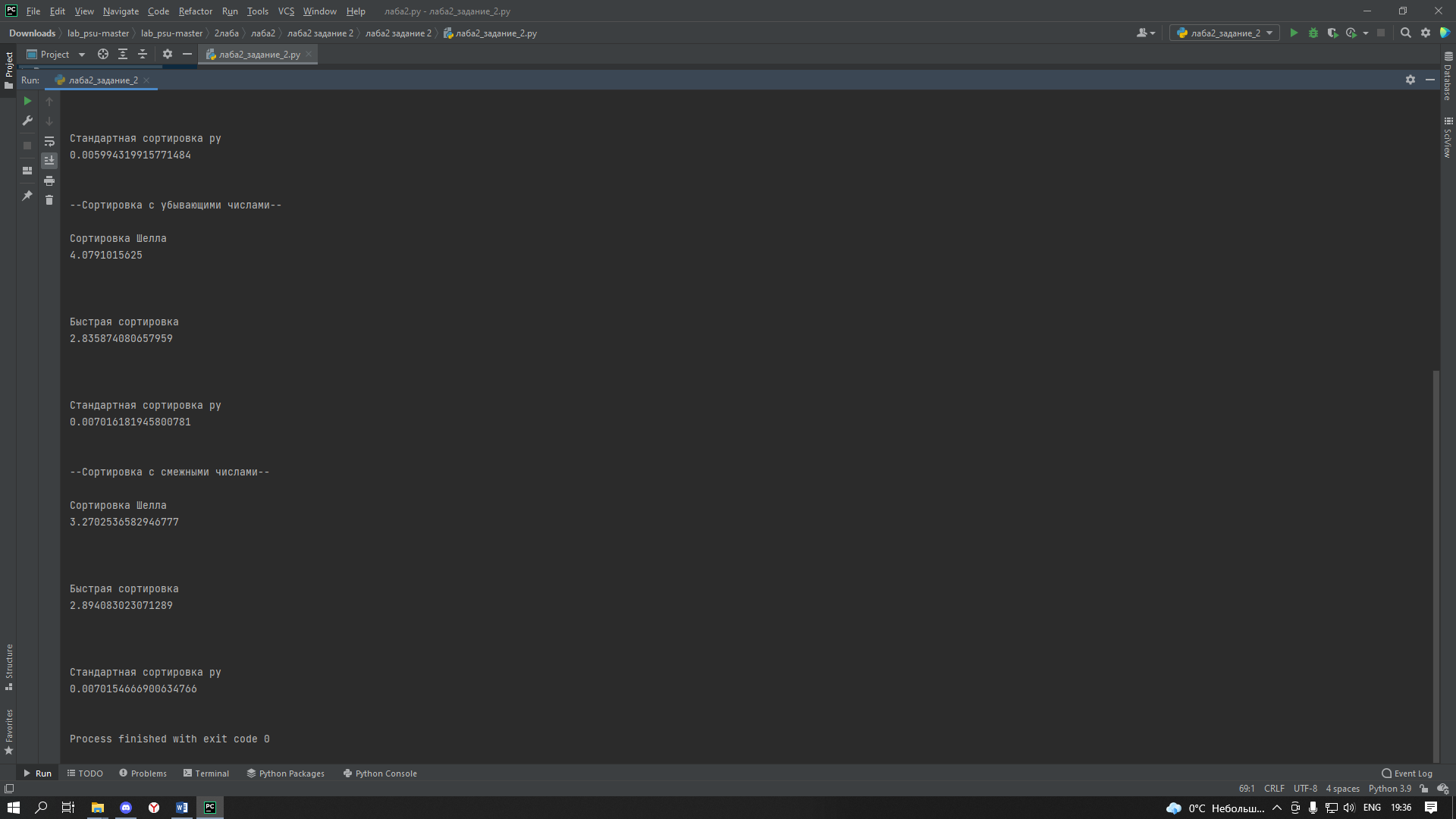
1. Оценить время работы каждого из реализованных алгоритмов на случайном наборе значений массива.

2. Оценить время работы каждого из реализованных алгоритмов на массиве, представляющем собой возрастающую последовательность чисел.

3. Оценить время работы каждого из реализованных алгоритмов на массиве, представляющем собой убывающую последовательность чисел.

4. Оценить время работы каждого из реализованных алгоритмов на массиве, одна половина которого представляет собой возрастающую последовательность чисел, а вторая, – убывающую.

5. Оценить время работы с помощью стандартной сортировки python, реализующей алгоритм быстрой сортировки на выше указанных наборах данных.



**Листинг задания 2:**  
def shellSort(data, length):  
 gap = length//2  
 while gap > 0:   
 for i in range(gap, length):  
 temp = data[i]  
 j = i  
 while(j >= gap and data[j - gap] > temp):  
 data[j] = data[j - gap]  
 j -= gap   
 data[j] = temp  
 gap //= 2  
   
  
def quickSort(nums):  
 if len(nums) <= 1:  
 return nums  
 else:  
 q = random.choice(nums)  
 s\_nums = []  
 m\_nums = []  
 e\_nums = []  
 for n in nums:  
 if n < q:  
 s\_nums.append(n)  
 elif n > q:  
 m\_nums.append(n)  
 else:  
 e\_nums.append(n)  
 return quickSort(s\_nums) + e\_nums + quickSort(m\_nums)  
  
#случайные  
data1 = []  
for i in range(1000000):  
 data1.append(random.randint(0,100000))  
  
data2 = data1.copy()  
data3 = data1.copy()  
  
starttime = time.time()  
shellSort(data1, len(data1))  
endtime = time.time()  
print("\n\n--Сортировка cо cлучайными числами--")  
print("\nСортировка Шелла")  
print(endtime - starttime)  
print("\n")  
  
starttime = time.time()  
quickSort(data2)  
endtime = time.time()  
print("\nБыстрая сортировка")  
print(endtime - starttime)  
print("\n")  
  
starttime = time.time()  
data3.sort()  
endtime = time.time()  
print("\nСтандартная сортировка py")  
print(endtime - starttime)  
  
#возрастающие  
data1 = []  
  
for i in range(1000000):  
 data1.append(i)   
data2 = data1.copy()  
data3 = data1.copy()  
  
starttime = time.time()  
shellSort(data1, len(data1))  
endtime = time.time()  
print("\n\n--Сортировка с возрастающими числами--")  
print("\nСортировка Шелла")  
print(endtime - starttime)  
print("\n")  
  
starttime = time.time()  
quickSort(data2)  
endtime = time.time()  
print("\nБыстрая сортировка")  
print(endtime - starttime)  
print("\n")  
  
starttime = time.time()  
data3.sort()  
endtime = time.time()  
print("\nСтандартная сортировка py")  
print(endtime - starttime)  
  
#убывающие  
data1 = []  
  
for i in range(1000000):  
 data1.append(1000000-i)   
data2 = data1.copy()  
data3 = data1.copy()  
  
starttime = time.time()  
shellSort(data1, len(data1))  
endtime = time.time()  
print("\n\n--Сортировка с убывающими числами--")  
print("\nСортировка Шелла")  
print(endtime - starttime)  
print("\n")  
  
starttime = time.time()  
quickSort(data2)  
endtime = time.time()  
print("\nБыстрая сортировка")  
print(endtime - starttime)  
print("\n")  
  
starttime = time.time()  
data3.sort()  
endtime = time.time()  
print("\nСтандартная сортировка py")  
print(endtime - starttime)  
  
#смешанная  
data1 = []  
  
for i in range(int(1000000/2)):  
 data1.append(i)  
for i in range(int(1000000/2)):  
 data1.append(1000000-i)  
data2 = data1.copy()  
data3 = data1.copy()  
  
starttime = time.time()  
shellSort(data1, len(data1))  
endtime = time.time()  
print("\n\n--Сортировка с смежными числами--")  
print("\nСортировка Шелла")  
print(endtime - starttime)  
print("\n")  
  
starttime = time.time()  
quickSort(data2)  
endtime = time.time()  
print("\nБыстрая сортировка")  
print(endtime - starttime)  
print("\n")  
  
starttime = time.time()  
data3.sort()  
endtime = time.time()  
print("\nСтандартная сортировка py")  
print(endtime - starttime)  
input()

Построила график зависимости времени выполнения программы от сортировки:

**Вывод:** Исходя из результатов выполнения алгоритмов и подсчета их времени, наглядно на графике видно, что quicksort работает намного стабильнее и эффективнее, чем все остальные.