Министерство цифрового развития, связи

и массовых коммуникаций Российской Федерации

Ордена Трудового Красного Знамени

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Московский технический университет связи и информатики»

Лабораторная работа №1

по дисциплине «Структуры и алгоритмы обработки данных»

“Методы сортировки”

Выполнила:

студентка группы БСТ2004

Моисеева С.Д.

Проверил:

А.Д. Чайка

Москва 2022

**Цель работы**: изучение основных алгоритмов сортировки и их реализация на Python

**Задание:** Создать проект .ipynb, включающий реализацию следующих заданий:

1. Вывести на экран строку "Hello World"

2. Написать генератор случайных матриц(многомерных), который принимает опциональные параметры m, n, min\_limit, max\_limit, где m и n указывают размер матрицы, а min\_lim и max\_lim - минимальное и максимальное значение для генерируемого числа. По умолчанию при отсутствии параметров принимать следующие значения: m = 50 n = 50 min\_limit = -250 max\_limit = 1000 + (номер своего варианта (11))

3. Реализовать методы сортировки строк числовой матрицы в соответствии с заданием (сортировки выбором, вставкой, пузырьком, Шелла, пирамидальную, турнирную, быструю). Оценить время работы каждого алгоритма сортировки и сравнить его со Временем стандартной функции сортировки. Испытания проводить на сгенерированных матрицах

**Ход работы.**

1. **Задание 1**

С помощью функционала редактора jupyter notebook для выполнения каждого задания были созданы ячейки с оформлением условия задания, а также ячейки кода. Для выполнения первого задания в первой исполняемой ячейке с помощью метода print() была выведена строка «Hello world». Результат выполнения задания представлен на рисунке 1:

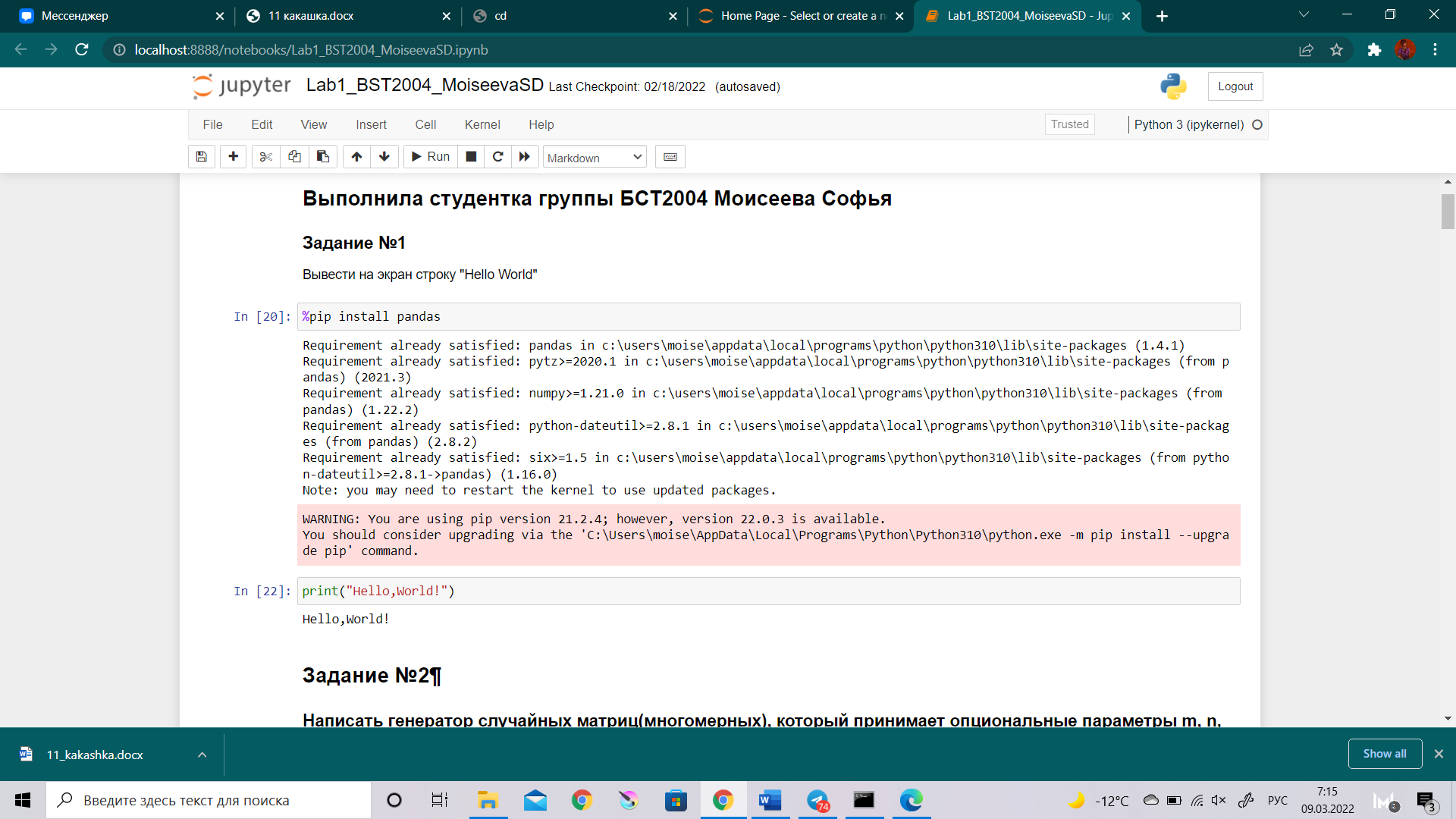


Рисунок 1 – Создание проекта, оформление, вывод и установка pandas

1. **Генератор случайных матриц**

Для реализации следующего задания был написан метод, генерирующий случайные матрицы (двумерные списки), который принимает пользовательские параметры m, n, min\_limit, max\_limit, где m и n указывают размер матрицы, а min\_lim и max\_lim - минимальное и максимальное значение для генерируемого числа. По умолчанию при отсутствии параметров программа берёт значения: m = 50 n = 50 min\_limit = -250 max\_limit = 1011. Код метода представлен на рисунке 2. В данном методе за заполнение и рандомизацию элементов отвечает строка: [[randint(min\_limit, max\_limit+1) for j in range(n)] for i in range(m)]. Результат выполнения представлен на рисунке 3.

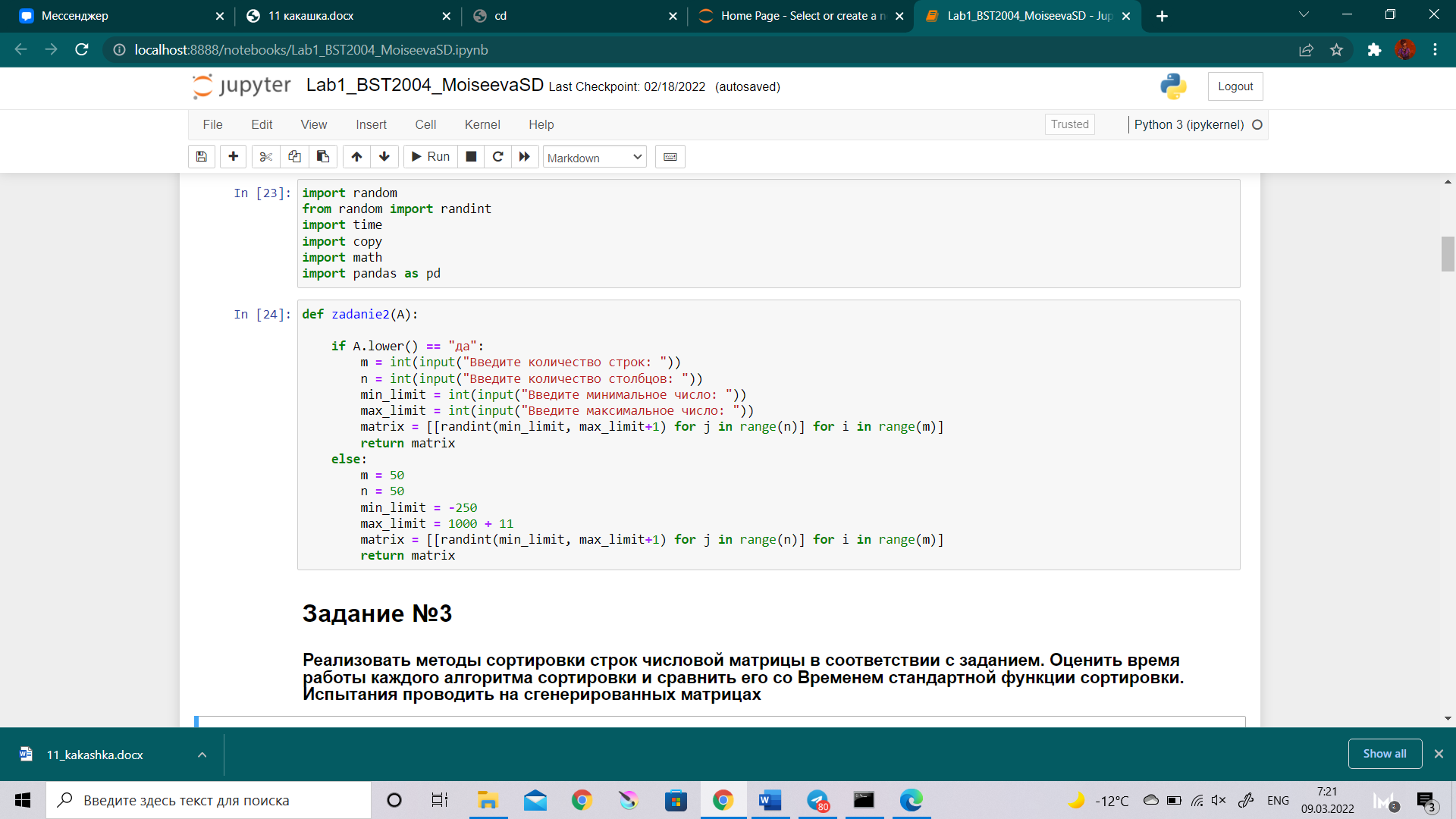


Рисунок 2 – Метод генерации матрицы

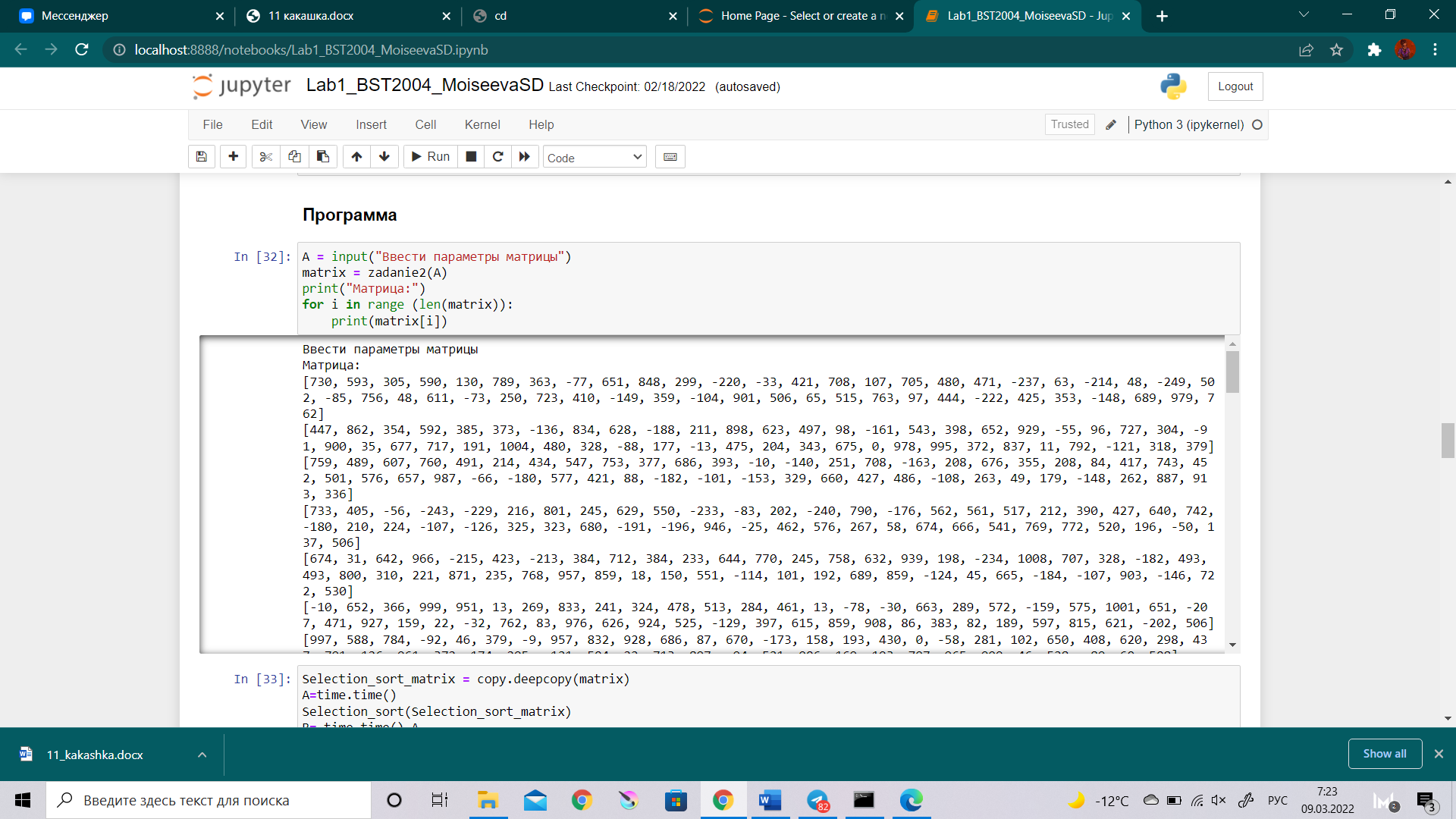


Рисунок 3 – Результат выполнения метода заполнения матрицы

1. **Реализация методов сортировки**

Далее были реализованы методы сортировки строк сгенерированной матрицы: выбором, вставкой, пузырьком, Шелла, пирамидальной, турнирной, быстрой). Испытания были проведены на сгенерированной матрице. По оценке времени самой быстрой оказалась встроенная сортировка (0.00058 секунды), а из реализованных – сортировка Шелла (0.008 секунды). Далее приведён код каждого метода сортировки:

* Сортировка выбором:

def Selection\_sort(matrix):

for row in range(len(matrix)):

for i in range(len(matrix[row]) - 1):

m = i

j = i + 1

while j < len(matrix[row]):

if matrix[row][j] < matrix[row][m]:

m = j

j = j + 1

matrix[row][i], matrix[row][m] = matrix[row][m], matrix[row][i]

return matrix

* Сортировка вставкой:

def insertion\_sort(matrix):

for row in range(len(matrix)):

for col in range (1,len(matrix[0])):

key = matrix[row][col]

j = col-1

while j >=0 and key < matrix[row][j]:

matrix[row][j+1] = matrix[row][j]

j -= 1

matrix[row][j+1] = key

return matrix

* Сортировка обменом:

def bubble\_sort(matrix):

for row in range(len(matrix)):

for col in range(len(matrix[0])-1):

for j in range(len(matrix[0])-col-1):

if matrix[row][j] > matrix[row][j+1]:

matrix[row][j], matrix[row][j+1] = matrix[row][j+1], matrix[row][j]

return matrix

* Сортировка Шелла

def shellSort(matrix):

n = len(matrix[0])

k = int(math.log2(n))

interval = int(2\*\*k -1)

while interval > 0:

for row in range(len(matrix)):

for col in range(interval, n):

temp = matrix[row][col]

j = col

while j >= interval and matrix[row][j - interval] > temp:

matrix[row][j] = matrix[row][j - interval]

j -= interval

matrix[row][j] = temp

k -= 1

interval = int(2\*\*k -1)

return matrix

* Турнирная сортировка:

def tournamentSort(matrix):

tree = [None] \* 2 \* (len(matrix) + len(matrix) % 2)

index = len(tree) - len(matrix) - len(matrix) % 2

for i, v in enumerate(matrix):

tree[index + i] = (i, v)

for j in range(len(matrix)):

n = len(matrix)

index = len(tree) - len(matrix) - len(matrix) % 2

while index > -1:

n = (n + 1) // 2

for i in range(n):

i = max(index + i \* 2, 1)

if tree[i] != None and tree[i + 1] != None:

if tree[i][1] < tree[i + 1][1]:

tree[i // 2] = tree[i]

else:

tree[i // 2] = tree[i + 1]

else:

tree[i // 2] = tree[i] if tree[i] != None else tree[i + 1]

index -= n

index, x = tree[0]

matrix[j] = x

tree[len(tree) - len(matrix) - len(matrix) % 2 + index] = None

def TournamentSort(matrix):

for i in range(len(matrix)):

tournamentSort(matrix[i])

return matrix

* Пирамидальная сортировка:

def heapify(matrix, n, i, row):

largest = i

l = 2 \* i + 1

r = 2 \* i + 2

if l < n and matrix[row][i] < matrix[row][l]:

largest = l

if r < n and matrix[row][largest] < matrix[row][r]:

largest = r

if largest != i:

matrix[row][i], matrix[row][largest] = matrix[row][largest], matrix[row][i]

heapify(matrix, n, largest, row)

def heapSort(matrix):

for row in range(len(matrix)):

n = len(matrix[0])

for i in range(n//2, -1, -1):

heapify(matrix, n, i, row)

for i in range(n-1, 0, -1):

matrix[row][i], matrix[row][0] = matrix[row][0], matrix[row][i]

heapify(matrix, i, 0, row)

return matrix

* Быстрая сортировка:

def quick\_sort(A):

if len(A) <= 1:

return A

else:

q = random.choice(A)

L = [elem for elem in A if elem < q]

M = [q] \* A.count(q)

R = [elem for elem in A if elem > q]

return quick\_sort(L) + M + quick\_sort(R)

def Quick\_sort\_slice(matrix):

for i in range(len(matrix)):

matrix[i] = quick\_sort(matrix[i])

return matrix

Результаты сортировок представлены на рисунках ниже (рисунки 4-11):

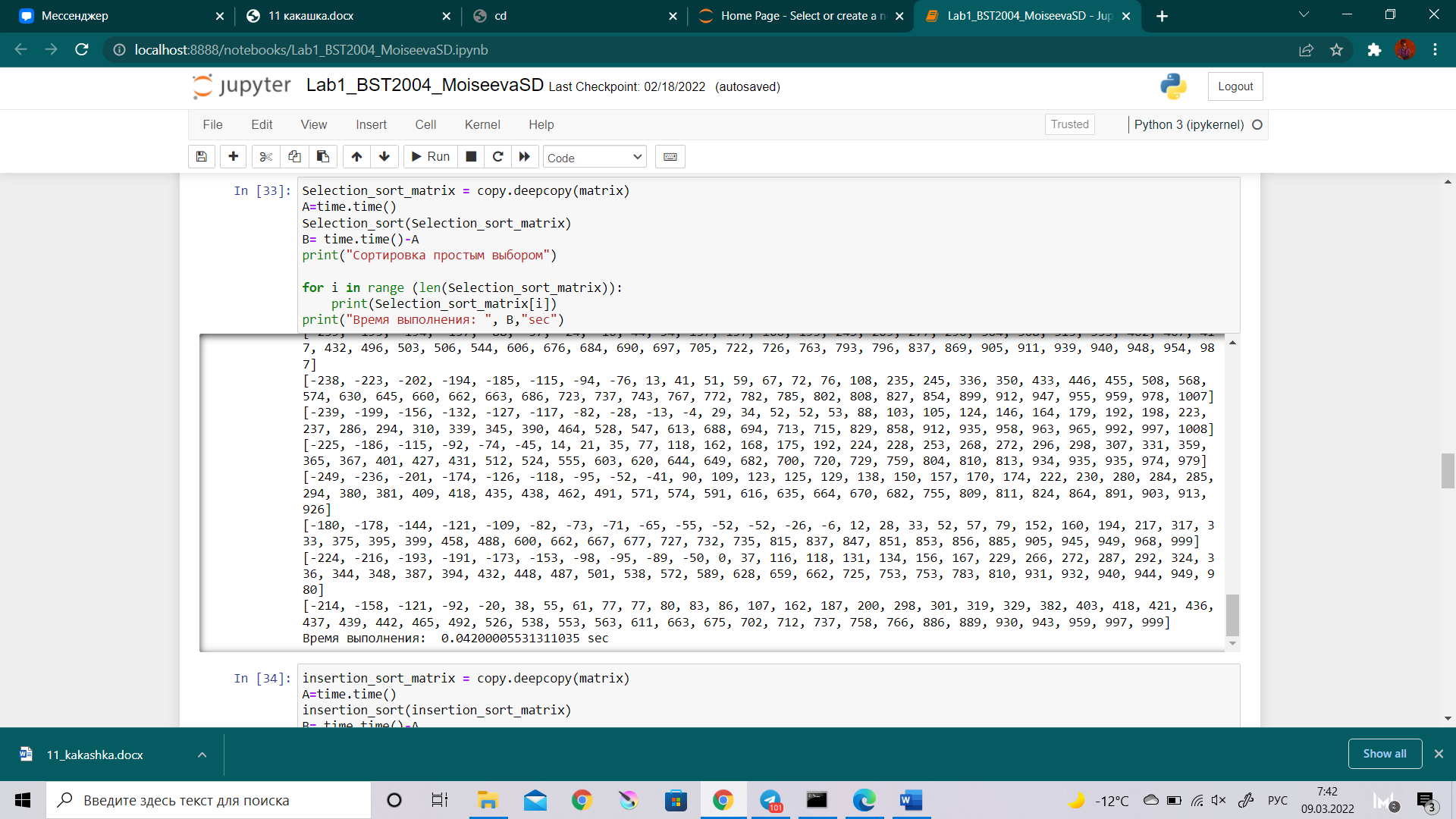


Рисунок 4 – Сортировка выбором

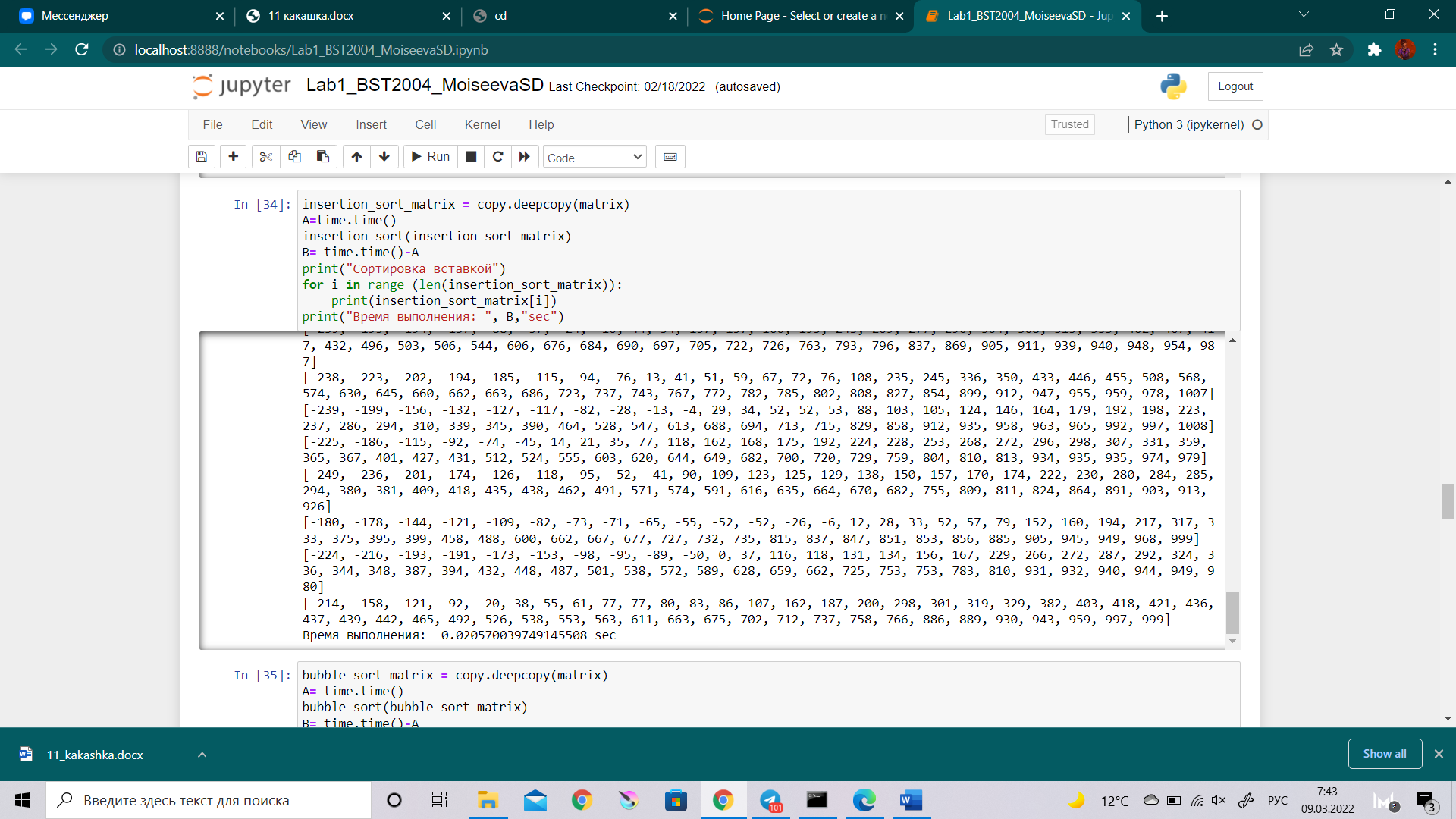


Рисунок 5 – Сортировка вставкой

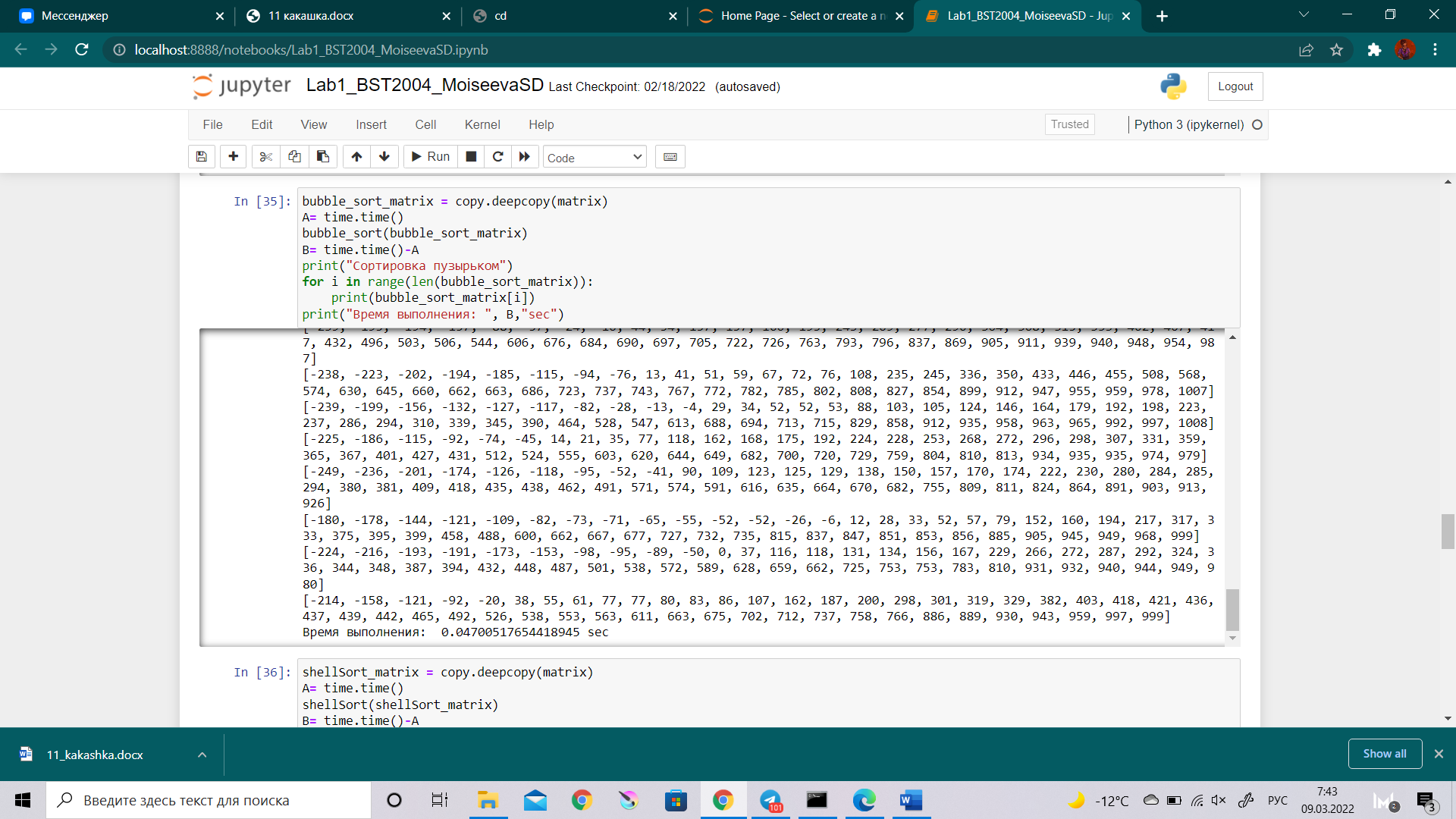


Рисунок 6 – Сортировка пузырьком

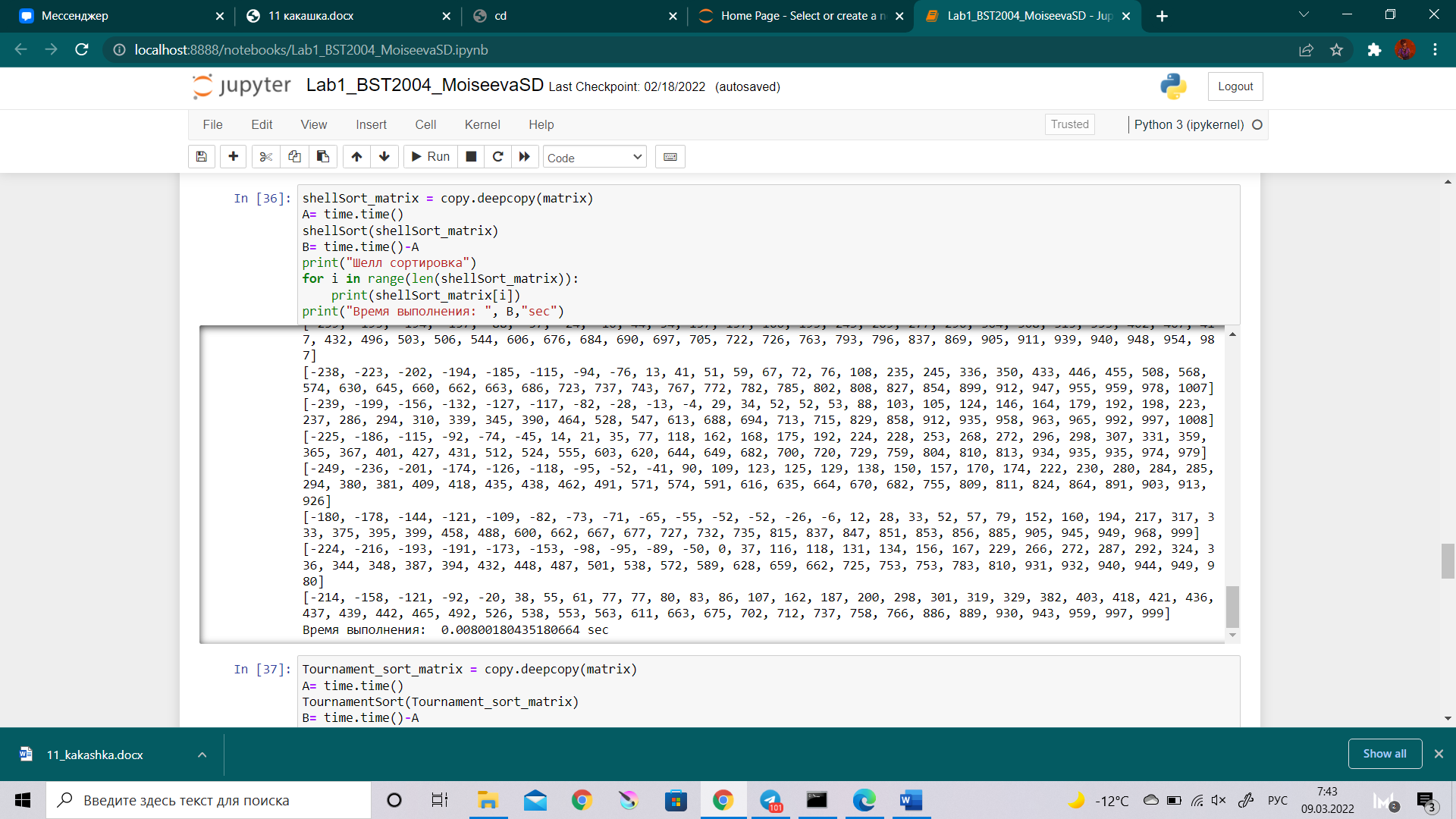


Рисунок 7 – Пирамидальная сортировка

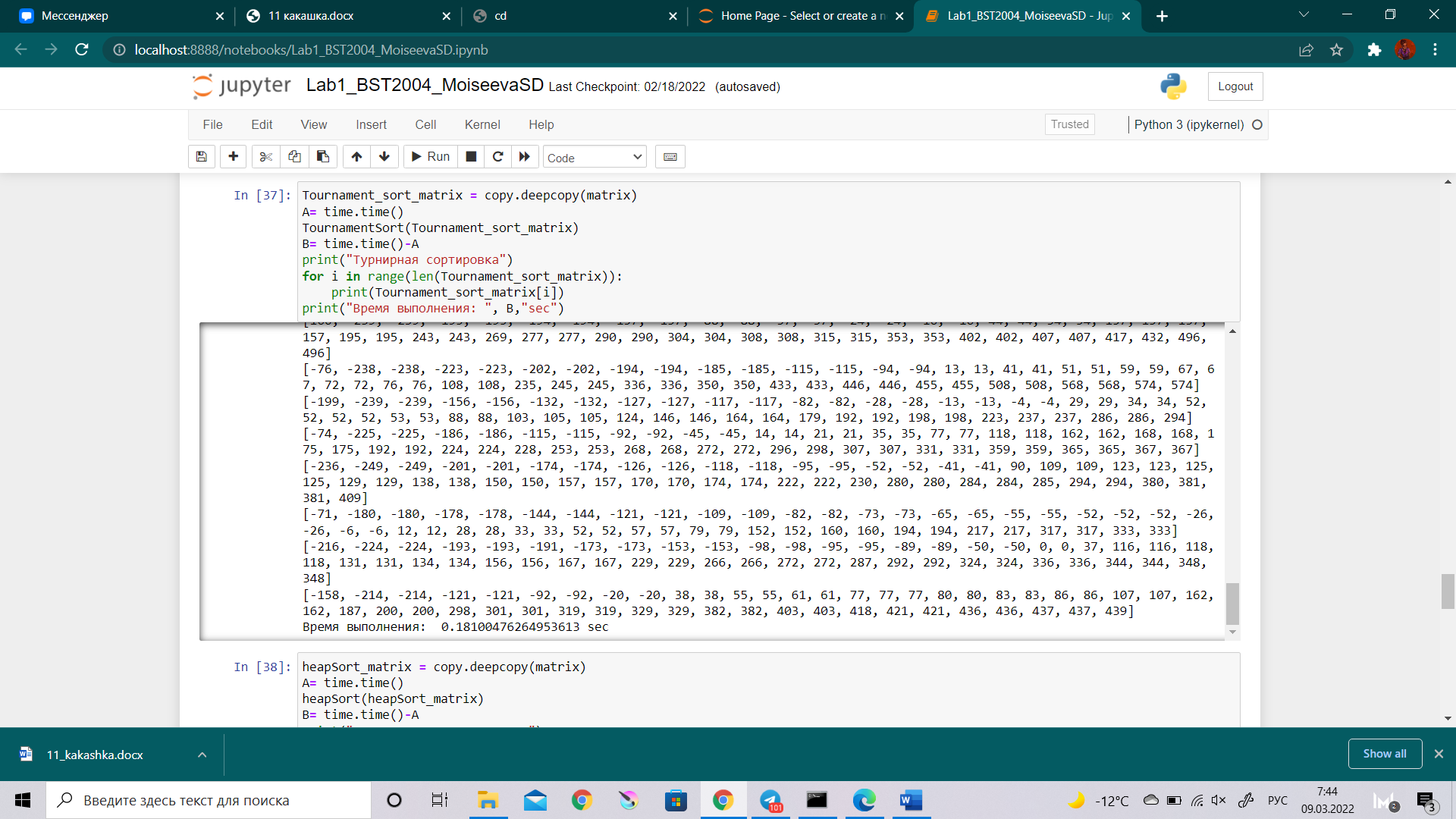


Рисунок 8 – Турнирная сортировка

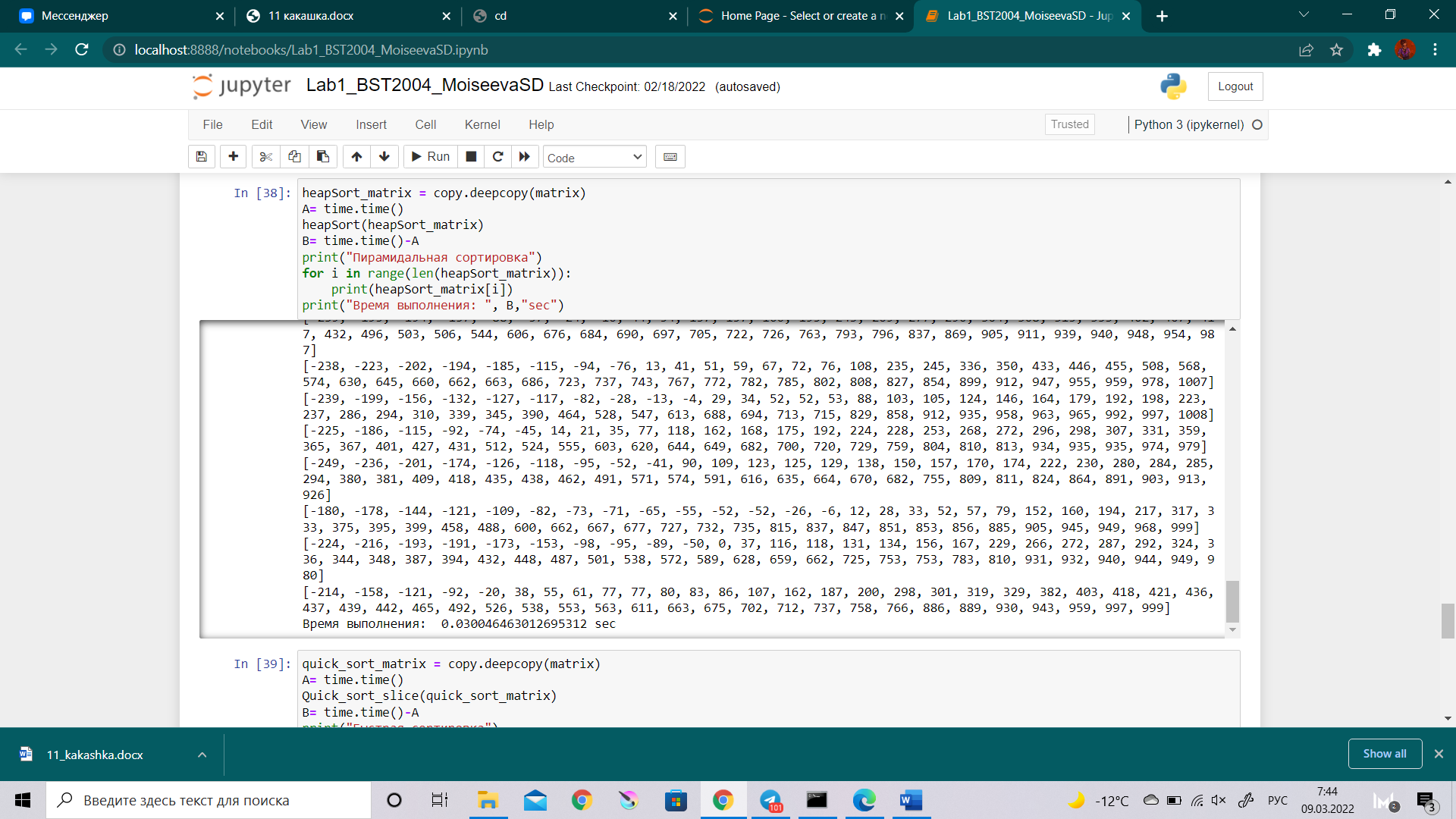


Рисунок 9 – Пирамидальная сортировка

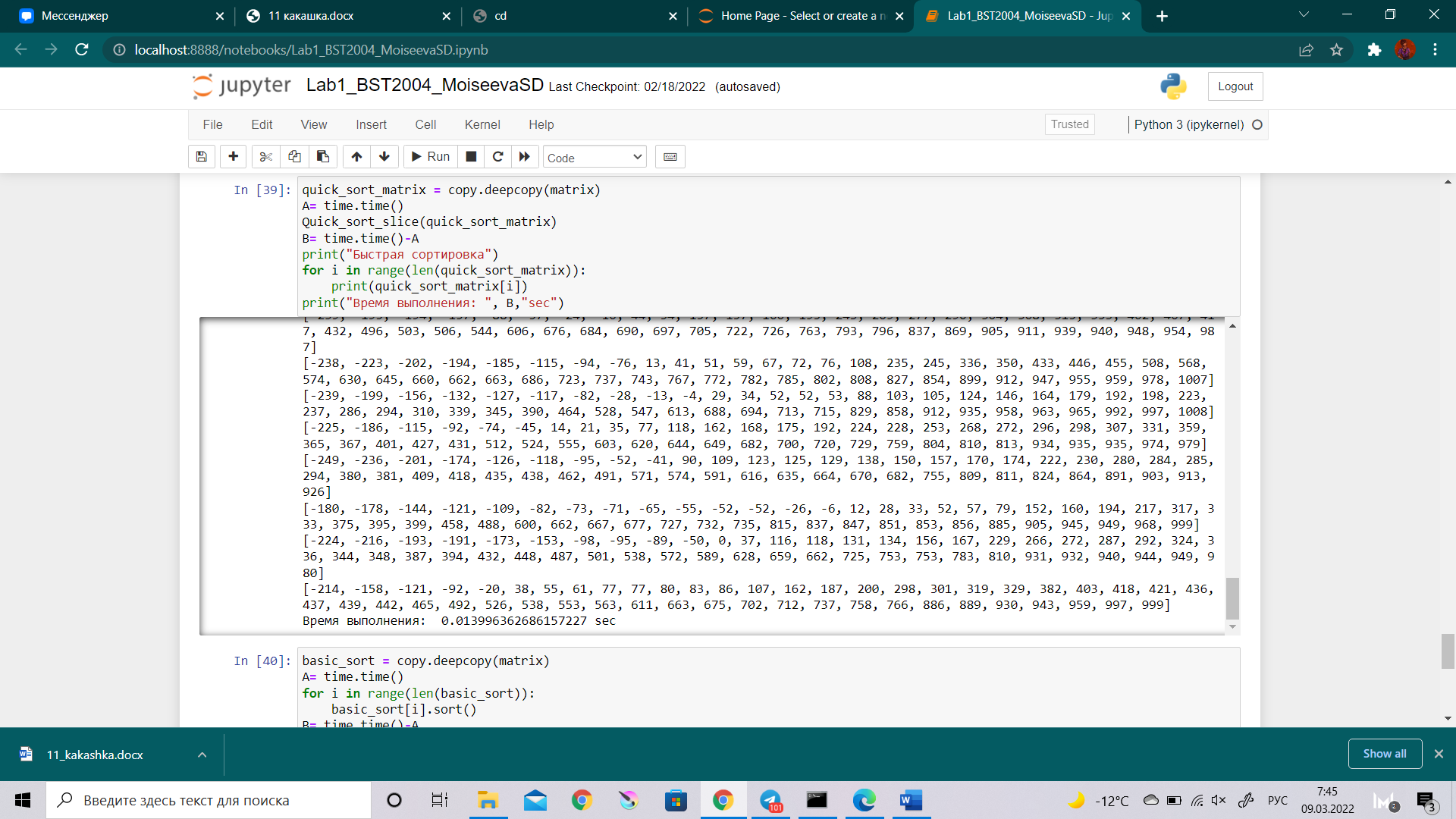


Рисунок 10 – Быстрая сортировка

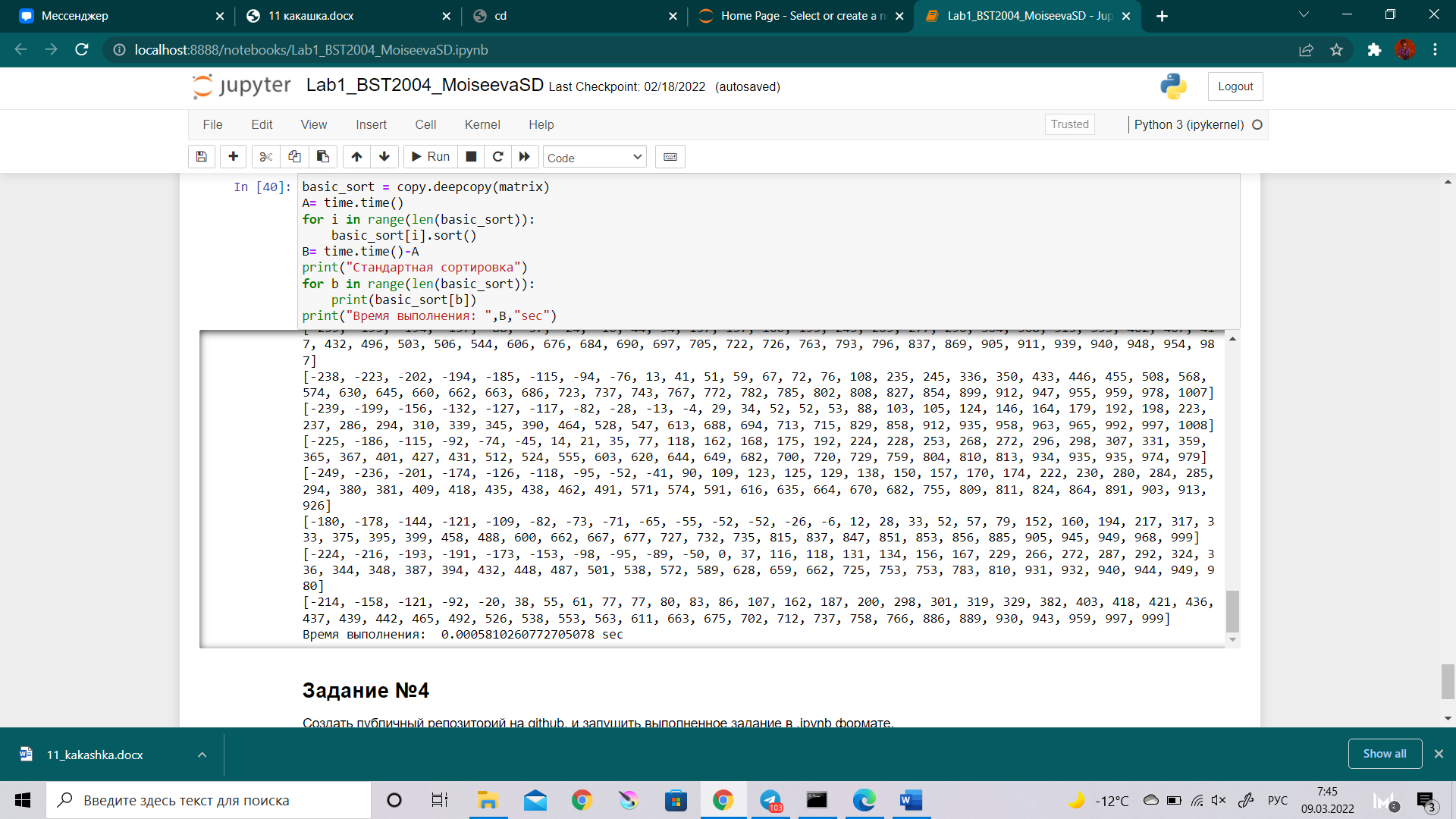


Рисунок 11 – Встроенная сортировка

Ссылка на репозиторий <https://github.com/mmaider/Lab1_SAOD>

**Список литературы**

1. Камаев В.А., Костерин В.В. Технологии программирования. М.: Высшая школа,

2006.

2. Жоголев Е.А.Технология программирования. – М.: Научный мир, 2004.

Git:

3. Scott Chacon, Ben Straub «Pro Git»

4. git-scm.com