Министерство образования Российской Федерации

Пензенский государственный университет

Кафедра «Системы автоматизированного проектирования»

**Отчет**

По лабораторной работе №2

По курсу «Логика и основы алгоритмизации в инженерных задачах»

На тему «Оценка времени выполнения программ»

Выполнил студент гр.20ВВ4

Горбунов Н. 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:**

Переписал алгоритм, данный в задании, на языке Python.

import time

from random import randint

start\_time = time.time()

# Задаём время начала выполнения программы

def Main():

A = []

B = []

C = []

for i in range(SIZE\_MATRIX):

A.append([])

B.append([])

for j in range(SIZE\_MATRIX):

A[i].append(randint(0,100))

B[i].append(randint(0,100))

for i in range(SIZE\_MATRIX):

C.append([])

for j in range(SIZE\_MATRIX):

C[i].append(A[i][j] \* B[i][j])

print("--- %s seconds ---" % (time.time() - start\_time))

1. O(N2)
2. В SIZE\_MATRIX я храню размер матрицы, перед выполнением функции изменяю значение. Вызываю главный метод 7 раз, с разными значениями и получаю разное время выполнения метода.  
   SIZE\_MATRIX = 100

Main()

SIZE\_MATRIX = 200

Main()

SIZE\_MATRIX = 400

Main()

SIZE\_MATRIX = 1000

Main()

SIZE\_MATRIX = 2000

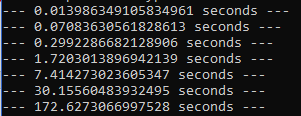
Main()

SIZE\_MATRIX = 4000

Main()

SIZE\_MATRIX = 10000  
Main()

Получаю такой результат:



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

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

**Задание 2**

Переписал алгоритмы сортировки:

import random

import time

#Shell Sort Algorithm

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:

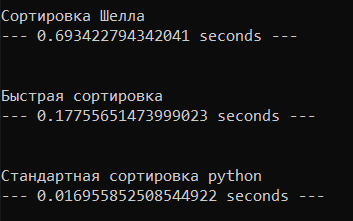
m\_nums.append(n)

else:

e\_nums.append(n)

return quickSort(s\_nums) + e\_nums + quickSort(m\_nums)

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



data1 = []

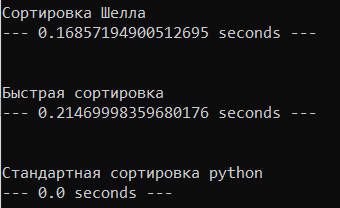
for i in range(100000):

data1.append(random.randint(0,10000))

data2 = data1.copy()

data3 = data1.copy()

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



data1 = []

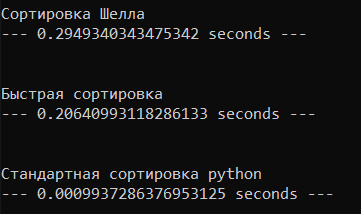
for i in range(100000):

data1.append(i)

data2 = data1.copy()

data3 = data1.copy()

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



data1 = []

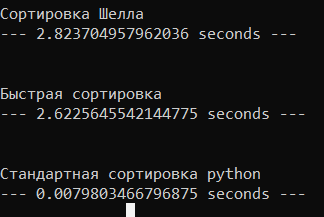
for i in range(100000):

data1.append(100000 - i)

data2 = data1.copy()

data3 = data1.copy()

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



data1 = []

for i in range(int(1000000/2)):

data1.append(i)

for i in range(int(1000000/2)):

data1.append(100000-i)

data2 = data1.copy()

data3 = data1.copy()



Вывод: Исходя их результатов выполнения алгоритмов и подсчета их времени, я пришел к выводу, что стандартная сортировка, встроенная в язык python, намного быстрее и эффективнее, чем остальные предложенные.