Министерство образования Российской Федерации

Пензенский государственный университет

Кафедра «Вычислительная техника»

**ОТЧЕТ**

по лабораторной работе №2

по курсу «Логика и основы алгоритмизации в инженерных задачах»

на тему «Оценка времени выполнения программ»

Выполнил:

студент группы 20ВВ2

Барсуков Н.И.

Проверили:

Митрохин М.А.

Юрова О.В.

Пенза 2021

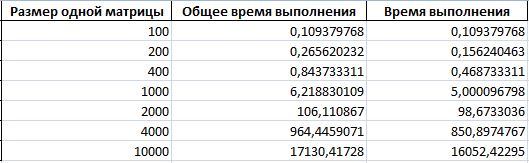
**Задание 1:**

1. Вычислить порядок сложности программы (О-символику).

2. Оценить время выполнения программы и кода, выполняющего перемножение матриц, используя функции библиотеки time.h для матриц размерами от 100, 200,400, 1000, 2000, 4000, 10000.

3. Построить график зависимости времени выполнения программы от размера матриц и сравнить полученный результат с теоретической оценкой.

1. O(n3)



**Листинг**

import random  
import numpy as p  
import time  
import pandas as pd  
import openpyxl  
  
final\_time\_all = []  
final\_time = []  
star\_time\_all = time.time()  
column = [100, 200, 400, 1000, 2000, 4000, 10000]  
  
for k in range(len(column)):  
 star\_time = time.time()  
 matrixA = [[random.randint(1, 100) for j in range(column[k])] for i in range(column[k])]  
 matrixB = [[random.randint(1, 100) for j in range(column[k])] for i in range(column[k])]  
  
 print(p.dot(matrixA, matrixB))  
 final\_time.append(time.time() - star\_time)  
 final\_time\_all.append(time.time() - star\_time\_all)  
  
result = pd.DataFrame({'Размер одной матрицы': column,  
 'Время выполнения': final\_time,  
 'Общее время выполнения': final\_time\_all})  
result.to\_excel("./results1.xlsx")

**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,

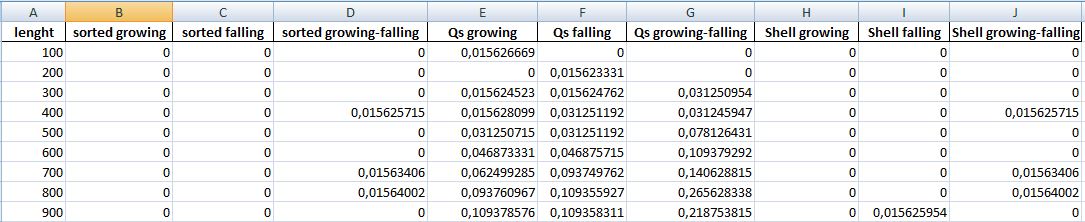
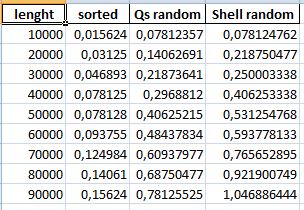
представляющем собой возрастающую последовательность чисел.

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 стандартной функции qsort, реализующей алгоритм быстрой сортировки на выше указанных наборах данных.



**Листинг**

import random  
import time  
import pandas as pd  
import xlsxwriter  
  
def quick\_sort\_normal(array):  
 if len(array) <= 1:  
 return array  
  
 support\_element = array[0]  
  
 left = list(filter(lambda x: x < support\_element, array))  
 center = [i for i in array if i == support\_element]  
 right = list(filter(lambda x: x > support\_element, array))  
  
 return quick\_sort\_normal(left) + center + quick\_sort\_normal(right)  
  
def shell\_sort(data):  
 data = data.copy()  
 last\_index = len(data) - 1  
 step = len(data)//2  
 while step > 0:  
 for i in range(step, last\_index + 1, 1):  
 j = i  
 delta = j - step  
 while delta >= 0 and data[delta] > data[j]:  
 data[delta], data[j] = data[j], data[delta]  
 j = delta  
 delta = j - step  
 step //= 2  
 return data  
  
def randomNumbers(lenght):  
 return [random.randint(1, lenght) for i in range(lenght)]  
  
def growingNumbers(lenght):  
 return [i for i in range(lenght)]  
  
def fallingNumbers(lenght):  
 return [i for i in range(lenght, 0, -1)]  
  
def growingAndFallingNumbers(lenght):  
 return [i for i in range(lenght)] + [i for i in range(lenght, 0, -1)]  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 time\_qs\_random = []  
 time\_qs\_growing = []  
 time\_qs\_falling = []  
 time\_qs\_gf = []  
 time\_shell\_random = []  
 time\_shell\_growing = []  
 time\_shell\_falling = []  
 time\_shell\_gf = []  
 time\_sort\_random = []  
 time\_sort\_growing = []  
 time\_sort\_falling = []  
 time\_sort\_gf = []  
  
 lenght\_random = [i for i in range(10000, 100000, 10000)]  
 lenght = [i for i in range(100, 1000, 100)]  
  
 for i in range(len(lenght\_random)):  
 start\_time\_sort\_random = time.time()  
 rMassive = randomNumbers(lenght\_random[i])  
 sorted(rMassive)  
 time\_sort\_random.append(time.time() - start\_time\_sort\_random)  
  
 start\_time\_qs\_random = time.time()  
 rMassive = randomNumbers(lenght\_random[i])  
 quick\_sort\_normal(rMassive)  
 time\_qs\_random.append(time.time() - start\_time\_qs\_random)  
  
 start\_time\_shell\_random = time.time()  
 rMassive = randomNumbers(lenght\_random[i])  
 shell\_sort(rMassive)  
 time\_shell\_random.append(time.time() - start\_time\_shell\_random)  
  
 for i in range(len(lenght)):  
 start\_time\_qs\_growing = time.time()  
 gMassive = growingNumbers(lenght[i])  
 quick\_sort\_normal(gMassive)  
 time\_qs\_growing.append(time.time() - start\_time\_qs\_growing)  
  
 start\_time\_qs\_falling = time.time()  
 fMassive = fallingNumbers(lenght[i])  
 quick\_sort\_normal(fMassive)  
 time\_qs\_falling.append(time.time() - start\_time\_qs\_falling)  
  
 start\_time\_qs\_gf = time.time()  
 gfMassive = growingAndFallingNumbers(lenght[i])  
 quick\_sort\_normal(gfMassive)  
 time\_qs\_gf.append(time.time() - start\_time\_qs\_gf)  
  
 start\_time\_shell\_growing = time.time()  
 gMassive = growingNumbers(lenght[i])  
 shell\_sort(gMassive)  
 time\_shell\_growing.append(time.time() - start\_time\_shell\_growing)  
  
 start\_time\_shell\_falling = time.time()  
 fMassive = fallingNumbers(lenght[i])  
 shell\_sort(fMassive)  
 time\_shell\_falling.append(time.time() - start\_time\_shell\_falling)  
  
 start\_time\_shell\_gf = time.time()  
 gfMassive = growingAndFallingNumbers(lenght[i])  
 shell\_sort(gfMassive)  
 time\_shell\_gf.append(time.time() - start\_time\_shell\_gf)  
  
 start\_time\_sort\_falling = time.time()  
 fMassive = fallingNumbers(lenght[i])  
 sorted(fMassive)  
 time\_sort\_falling.append(time.time() - start\_time\_sort\_falling)  
  
 start\_time\_sort\_growing = time.time()  
 gMassive = growingNumbers(lenght[i])  
 sorted(gMassive)  
 time\_sort\_growing.append(time.time() - start\_time\_sort\_growing)  
  
 start\_time\_sort\_gf = time.time()  
 gfMassive = growingAndFallingNumbers(lenght[i])  
 sorted(gfMassive)  
 time\_sort\_gf.append(time.time() - start\_time\_sort\_gf)  
  
 result\_random = pd.DataFrame({'lenght': lenght\_random,  
 'sorted': time\_sort\_random,  
 'Qs random': time\_qs\_random,  
 'Shell random': time\_shell\_random})  
  
 result\_other = pd.DataFrame({'lenght': lenght,  
 'sorted growing': time\_sort\_growing,  
 'sorted falling': time\_sort\_falling,  
 'sorted growing-falling': time\_shell\_gf,  
 'Qs growing': time\_qs\_growing,  
 'Qs falling': time\_qs\_falling,  
 'Qs growing-falling': time\_qs\_gf,  
 'Shell growing': time\_shell\_growing,  
 'Shell falling': time\_shell\_falling,  
 'Shell growing-falling': time\_shell\_gf})  
  
 sheet = {'random': result\_random, 'other': result\_other}  
 writer = pd.ExcelWriter('./results2.xlsx', engine='xlsxwriter')  
  
 for sheet\_name in sheet.keys():  
 sheet[sheet\_name].to\_excel(writer, sheet\_name=sheet\_name, index=False)  
  
 writer.save()

**Вывод:**

В первом задании сложность алгоритма составила O(n3). В хоте тестирования на различных размерах матриц, было выявлено, что рассчитанная сложность совпадает с фактической, что подтверждает построенный график.

Во втором задании производилось сравнивание двух алгоритмов сортировки и одной стандартной функции на различных типах данных. В ходе тестирования было выявлено, что стандартная функция оказалась быстрее во всех случаях, алгоритм quick\_sort на не отсортированном массиве оказался быстрее алгоритма Шелла при больших значениях, однако, на отсортированных массивах алгоритм Шелла показал лучшие результаты.