Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Российский химико-технологический университет имени Д.И. Менделеева»

Кафедра информационных компьютерных технологий

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ № 1

Выполнил студент группы КС-38 (Нергарян Геворг Гарегинович)

Ссылка на репозиторий: (https://github.com/MUCTR-IKT-CPP/Nergaryan\_KC-38\_Algos)

Приняли: Пысин Максим Дмитриевич

Краснов Дмитрий Олегович

Лобанов Алексей Владимирович

Крашенинников Роман Сергеевич

Дата сдачи: …………………….04.03.2023…………………………………………………………………..

Оглавление

[Описание задачи. 2](#_Toc63548272)

[Описание метода/модели. 2](#_Toc63548273)

[Выполнение задачи. 3](#_Toc63548274)

[Заключение. 8](#_Toc63548275)

# Описание задачи.

Вариант 1. Сортировка пузырьком.

В рамках лабораторной работы необходимо изучить и реализовать метод сортировки соответствующий вашему варианту (приведены ниже).  
Для реализованного метода сортировки необходимо провести серию тестов для всех значений N из списка (1000, 2000, 4000, 8000, 16000, 32000, 64000, 128000), при этом:

* в каждом тесте необходимо по 20 раз генерировать вектор, состоящий из N элементов
* каждый элемент массива заполняется случайным числом с плавающей запятой от -1 до 1, для этого можно использовать как C функцию rand(), так и С++ генераторы
* каждый массив после генерации необходимо отсортировать и замерить время, требуемое на сортировку, для замера времени использовать следующий код
* Результат замера для каждой попытки каждого теста записать в файл общий файл.
* При замере времени выбираем только один из варивантов (nano\_diff, micro\_diff, milli\_diff, sec\_diff).

По окончанию всех тестов необходимо нанести все точки, полученные в результате замеров времени на график где на ось абсцисс(Х) нанести N, а на ось ординат(Y) нанести значения времени на сортировку. По полученным точкам построить график лучшего (минимальное время для каждого N), худшего (максимальное время для каждого N) и среднего (среднее время для каждого N) случая.

# Описание метода/модели.

# Сортировка пузырьком — один из самых известных алгоритмов сортировки. Здесь нужно последовательно сравнивать значения соседних элементов и менять числа местами, если предыдущее оказывается больше последующего.

# Нам нужно создать два цикла for — внешний и внутренний. Один будет вложен в другой. При каждом проходе алгоритма по внутреннему циклу очередной наибольший элемент массива ставится на своё место в конце массива рядом с предыдущим наибольшим элементом («всплывает» до нужной позиции, как пузырёк в воде), а наименьший элемент перемещается на одну позицию к началу массива. Внешний цикл отвечает за то, чтобы данные перестановки были выполнены в итоге для всех элементов массива.

# В сортировке методом пузырька количество итераций внешнего цикла определяется длинной списка минус единица, так как когда второй элемент становится на свое место, то первый уже однозначно минимальный и находится на своем месте.

Количество итераций внутреннего цикла зависит от номера итерации внешнего цикла, так как конец списка уже отсортирован, и выполнять проход по этим элементам смысла нет.

Если мы говорим, что пузырьковая сортировка выполняется за время O(n^2), это значит, что медленнее она точно работать не будет. Здесь n это количество операций, которое предстоит выполнить алгоритму.

В среднем, сортировка пузырьком выполняется за квадратичное время, поэтому ожидаемая производительность алгоритма будет Θ(n^2). В данном случае, сортировка пузырьком выполняется за квадратичное время, потому что мы имеем два вложенных друг в друга цикла for, которые выполняют одинаковое количество операций.

Сортировка пузырьком — крайне неэффективный алгоритм.

Сортировка пузырьком расходует постоянное количество памяти. Дело в том, что мы сортируем уже имеющийся массив и не создаём дополнительных структур данных, в которых теоретически могли что-нибудь хранить.

К плюсам сортировки пузырьком относится простота реализации алгоритма, но такое подходит только в учебных целях.

Основной минус алгоритма сортировки «пузырьком» - его медленная скорость

# Выполнение задачи.

Использовался язык Python. Время измеряли в миллисекундах.

1. Код:

import random

import time

import matplotlib.pyplot as plt

plt.style.use('dark\_background')

k = 20 #кол-во прогонов

l = 7 #заканчиваем на 64000 (степени 2)

n = 1000

vector1000 = [] #пустые массивы для записи времени интерации

vector2000 = [] #будущие значения у

vector4000 = []

vector8000 = []

vector16000 = []

vector32000 = []

vector64000 = []

f = open('time.txt', 'w+') #создаем/открываем пустой файл

f.close()

#массива из случайных чисел от -1.0 до 1.0 k-раз для каждого n

for l in range(l):

f = open('time.txt', 'a') #записываем в файл с каким n работаем

c = " Count " +str(n) + "\n"

f.write(c)

f.close()

for i in range (k):

vector = []

for i in range(n):

vector.append(random.uniform(-1, 1)) #массив от -1 до 1

start = time.time() # начало таймера

for i in range(n - 1): # метод пузырька

for j in range(n - i - 1):

if vector[j] > vector[j + 1]:

vector[j], vector[j + 1] = vector[j + 1], vector[j]

end = time.time() # конец таймера

finaltime = (end - start) \* 10 \*\* 3

#в зависимости от n на текущей итерации заносим значения времени в соотв.массив

if n == 1000:

vector1000.append(finaltime)

elif n == 2000:

vector2000.append(finaltime)

elif n == 4000:

vector4000.append(finaltime)

elif n == 8000:

vector8000.append(finaltime)

elif n == 16000:

vector16000.append(finaltime)

elif n == 32000:

vector32000.append(finaltime)

elif n == 64000:

vector64000.append(finaltime)

finaltime = str(finaltime) + "\n"

f = open('time.txt', 'a') # время текущей итерации

f.write(finaltime)

f.close()

n \*= 2

#создаём общее окно для графиков и отдельное поле для каждого из 9 графиков

figure = plt.figure()

graf1 = figure.add\_subplot(3, 3, 1)

graf2 = figure.add\_subplot(3, 3, 2)

graf3 = figure.add\_subplot(3, 3, 3)

graf4 = figure.add\_subplot(3, 3, 4)

graf5 = figure.add\_subplot(3, 3, 5)

graf6 = figure.add\_subplot(3, 3, 6)

graf7 = figure.add\_subplot(3, 3, 7)

graf8 = figure.add\_subplot(3, 3, 8) #для 128к, но оно нам не надо

graf9 = figure.add\_subplot(3, 3, 9)

#время для n=1000 на первый график

avg\_time1k = 0

min\_time1k = min(vector1000)

max\_time1k = max(vector1000)

for i in range(len(vector1000)):

avg\_time1k += vector1000[i]

avg\_time1k /= len(vector1000)

m = len(vector1000)

x = [1000] \* m

graf1.set\_xlim(999, 1001)

graf1.set\_ylim(min\_time1k - 0.5, max\_time1k + 0.5)

for i in range(len(vector1000)):

graf1.plot(x[i], vector1000[i], marker=".", color="red")

#время для n=2000 на второй график

avg\_time2k = 0

min\_time2k = min(vector2000)

max\_time2k = max(vector2000)

for i in range(len(vector2000)):

avg\_time2k += vector2000[i]

avg\_time2k /= len(vector2000)

m = len(vector2000)

x = [2000] \* m

graf2.set\_xlim(1999, 2001)

graf2.set\_ylim(min\_time2k - 1, max\_time2k + 1)

for i in range(len(vector2000)):

graf2.plot(x[i], vector2000[i], marker=".", color="red")

#время для n=4000 на третий график

avg\_time4k = 0

min\_time4k = min(vector4000)

max\_time4k = max(vector4000)

for i in range(len(vector4000)):

avg\_time4k += vector4000[i]

avg\_time4k /= len(vector4000)

m = len(vector4000)

x = [4000] \* m

graf3.set\_xlim(3999, 4001)

graf3.set\_ylim(min\_time4k - 1.5, max\_time4k + 1.5)

for i in range(len(vector4000)):

graf3.plot(x[i], vector4000[i], marker=".", color="red")

#время для n=8000 на четвертый график

vector8000 = vector8000

avg\_time8k = 0

min\_time8k = min(vector8000)

max\_time8k = max(vector8000)

for i in range(len(vector8000)):

avg\_time8k += vector8000[i]

avg\_time8k /= len(vector8000)

m = len(vector8000)

x = [8000] \* m

graf4.set\_xlim(7999, 8001)

graf4.set\_ylim(min\_time8k - 2, max\_time8k + 2)

for i in range(len(vector8000)):

graf4.plot(x[i], vector8000[i], marker=".", color="red")

#время для n=16000 на пятый график

avg\_time16k = 0

min\_time16k = min(vector16000)

max\_time16k = max(vector16000)

for i in range(len(vector16000)):

avg\_time16k += vector16000[i]

avg\_time16k /= len(vector16000)

m = len(vector16000)

x = [16000] \* m

graf5.set\_xlim(15999, 16001)

graf5.set\_ylim(min\_time16k - 2, max\_time16k + 2)

for i in range(len(vector16000)):

graf5.plot(x[i], vector16000[i], marker=".", color="red")

#время для n=32000 на шестой график

avg\_time32k = 0

min\_time32k = min(vector32000)

max\_time32k = max(vector32000)

for i in range(len(vector32000)):

avg\_time32k += vector32000[i]

avg\_time32k /= len(vector32000)

m = len(vector32000)

x = [32000] \* m

graf6.set\_xlim(31999, 32001)

graf6.set\_ylim(min\_time32k - 2, max\_time32k + 2)

for i in range(len(vector32000)):

graf6.plot(x[i], vector32000[i], marker=".", color="red")

#время для n=64000 на седьмой график

avg\_time64k = 0

min\_time64k = min(vector64000)

max\_time64k = max(vector64000)

for i in range(len(vector64000)):

avg\_time64k+=vector64000[i]

avg\_time64k/=len(vector64000)

m = len(vector64000)

x = [64000] \* m

graf7.set\_xlim(63999, 64001)

graf7.set\_ylim(min\_time64k-2, max\_time64k+2)

for i in range(len(vector64000)):

graf7.plot(x[i], vector64000[i], marker=".", color="red")

#выводим график лучшего времени-n на девятый график

graphbest = [min\_time1k, min\_time2k, min\_time4k, min\_time8k, min\_time16k, min\_time32k, min\_time64k]

ybestgraph = [1000, 2000, 4000, 8000, 16000, 32000, 64000]

graf9.plot(graphbest, ybestgraph, color="green", marker='o')

#выводим график худшего времени-n на девятый график

graphworst = [max\_time1k, max\_time2k, max\_time4k, max\_time8k, max\_time16k, max\_time32k, max\_time64k]

yworstgraph = [1000, 2000, 4000, 8000, 16000, 32000, 64000]

graf9.plot(graphworst, yworstgraph, color="red", marker='o')

#выводим график среднего времени-n на девятый график

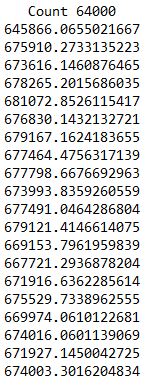
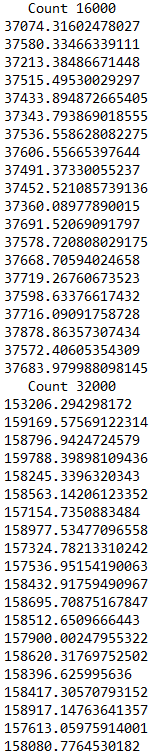
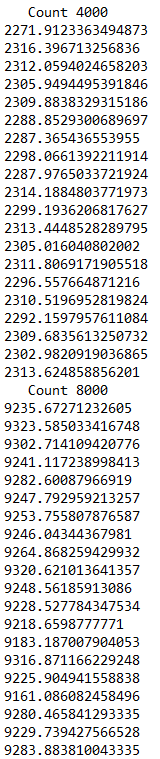
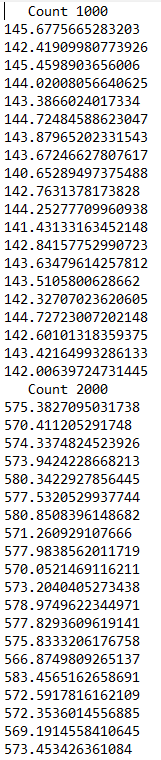
graphavg = [avg\_time1k, avg\_time2k, avg\_time4k, avg\_time8k, avg\_time16k, avg\_time32k, avg\_time64k]

yavggraph = [1000, 2000, 4000, 8000, 16000, 32000, 64000]

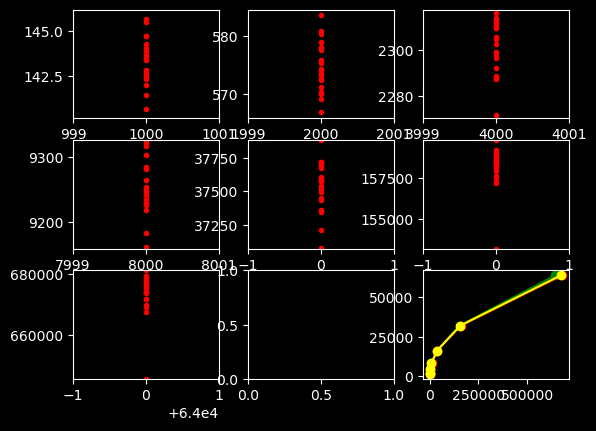
graf9.plot(graphavg, yavggraph, color="yellow", marker='o')

plt.show()

Данные выводятся в файл.



1. Графики выводятся в отдельном окне по окончанию выполнения программы. Графики идут по порядку от n = 1000 до n = до n = 64000. Предпоследнее поле для графика n = 128000, но из-за большого количества вычислений и затрат по времени оно пустует (без него не красиво, поэтому оставил). На последнем окне показаны графики лучшего (зелёный), среднего (жёлтый) и худшего (красный) замера времени в соответствии с его N.



# Заключение.

В ходе проделанной работы мы приходим к выводу, что этот алгоритм считается учебным и в чистом виде на практике почти не применяется, ведь среднее количество проверок и перестановок в массиве — это количество элементов в квадрате. Например, для массива из 10 элементов потребуется 100 проверок, а для массива из 100 элементов — уже в сто раз больше, 10 000 проверок. Каждый раз, когда увеличиваем количество элементов в массиве для его сортировки, разница во времени значительно увеличивается. К примеру, при увеличении N в 2 раза время увеличивается в 2^2 раза (в 4 раза в среднем).