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.

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 C++ 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(X) 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 y
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)
```

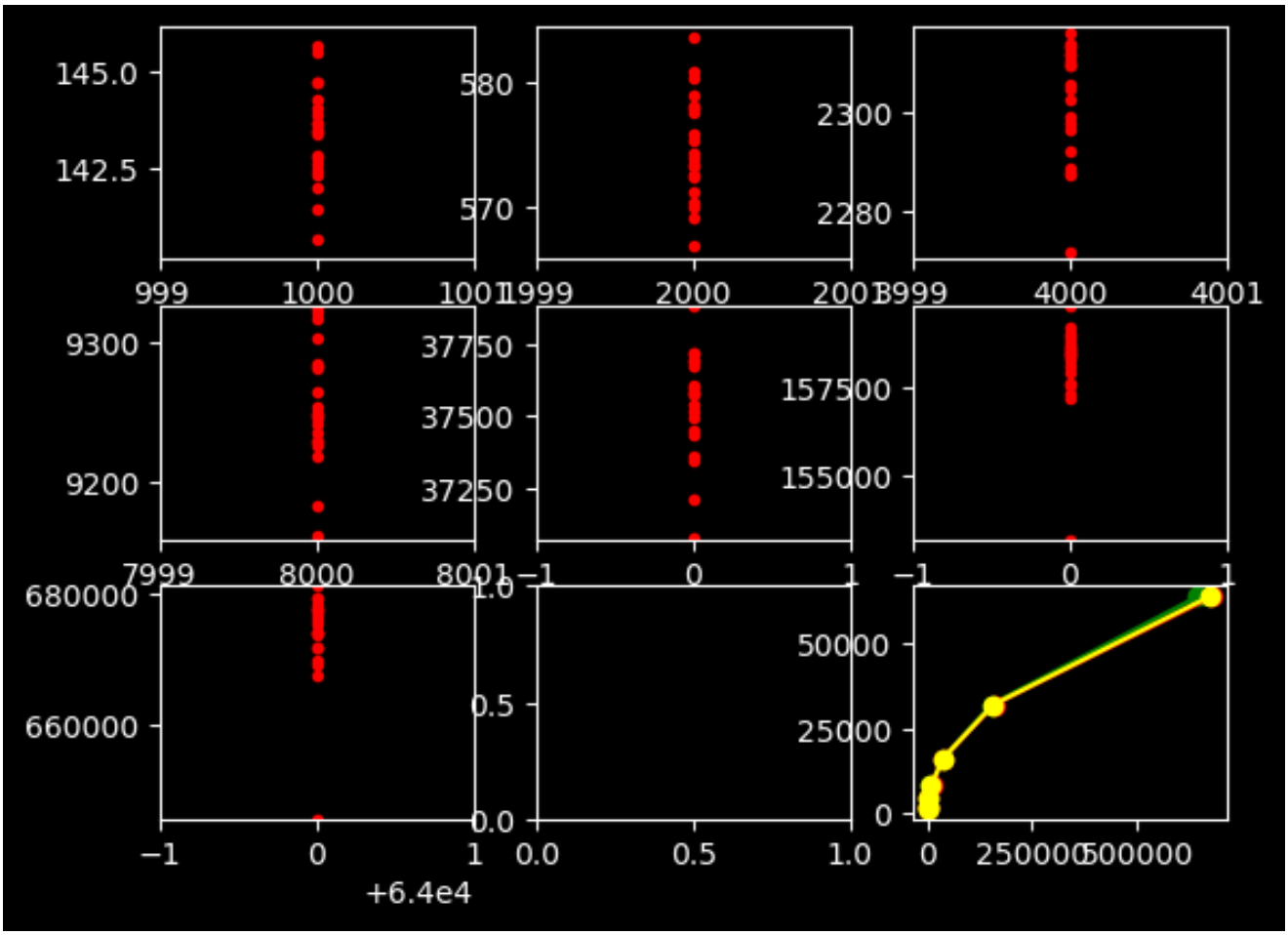
```
m = len(vector8000)
x = [8000] * m
graf4.set_xlim(7999, 8001)
graf4.set_ylim(min_time8k - 2, max_time8k + 2)
for i in range(len(vector8000)):
    graf4.plot(x[i], vector8000[i], marker=".", color="red")

#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.

Count 1000	Count 4000	Count 16000	Count 64000
145.6775665283203	2271.9123363494873	37074.31602478027	645866.0655021667
142.41909980773926	2316.396713256836	37580.33466339111	675910.2733135223
145.4598903656006	2312.0594024658203	37213.38486671448	673616.1460876465
144.02008056640625	2305.9494495391846	37515.49530029297	678265.2015686035
143.3866024017334	2309.8838329315186	37433.894872665405	681072.8526115417
144.72484588623047	2288.8529300689697	37343.793869018555	676830.1432132721
143.87965202331543	2287.365436553955	37536.558628082275	679167.1624183655
143.67246627807617	2298.0661392211914	37606.55665397644	677464.4756317139
140.65289497375488	2287.9765033721924	37491.37330055237	677798.6676692963
142.7631378173828	2314.1884803771973	37452.521085739136	673993.8359260559
144.25277709960938	2299.1936206817627	37360.08977890015	677491.0464286804
141.43133163452148	2313.4448528289795	37691.52069091797	679121.4146614075
142.84157752990723	2305.016040802002	37578.720808029175	669153.7961959839
143.63479614257812	2311.8069171905518	37668.70594024658	667721.2936878204
143.5105800628662	2296.557664871216	37719.26760673523	671916.6362285614
142.32707023620605	2310.5196952819824	37598.63376617432	675529.7338962555
144.72723007202148	2292.1597957611084	37716.09091758728	669974.0610122681
142.60101318359375	2309.6835613250732	37878.86357307434	674016.0601139069
143.42164993286133	2302.9820919036865	37572.40605354309	671927.1450042725
142.00639724731445	2313.624858856201	37683.979988098145	674003.3016204834
Count 2000	Count 8000	Count 32000	
575.3827095031738	9235.67271232605	153206.294298172	
570.411205291748	9323.585033416748	159169.57569122314	
574.3374824523926	9302.714109420776	158796.9424724579	
573.9424228668213	9241.117238998413	159788.39898109436	
580.3422927856445	9282.60087966919	158245.3396320343	
577.5320529937744	9247.792959213257	158563.14206123352	
580.8508396148682	9253.755807876587	157154.7350883484	
571.260929107666	9246.04344367981	158977.53477096558	
577.9838562011719	9264.868259429932	157324.78213310242	
570.0521469116211	9320.621013641357	157536.95154190063	
573.2040405273438	9248.56185913086	158432.91759490967	
578.9749622344971	9228.527784347534	158695.70875167847	
577.8293609619141	9218.6598777771	158512.6509666443	
575.8333206176758	9183.187007904053	157900.00247955322	
566.8749809265137	9316.871166229248	158620.31769752502	
583.4565162658691	9225.904941558838	158396.625995636	
572.5917816162109	9161.086082458496	158417.30570793152	
572.3536014556885	9280.465841293335	158917.14763641357	
569.1914558410645	9229.739427566528	157613.05975914001	
573.453426361084	9283.883810043335	158080.7764530182	

2) 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).