

***Кафедра \_\_информационных систем\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_***

***Рейтинговая работа "Расчетно-аналитическое задание"***

***по дисциплине Алгоритмизация и программирование***

***Выполнена обучающимся группы з.ИЗДт 23.1/Б3-22***

**Герасимовой Анастасией Сергеевной**

(фамилия, имя, отчество)

***Преподаватель Камолова Анастасия Николаевна*** (фамилия, имя, отчество)

Москва – 2024 г.

ОГЛАВЛЕНИЕ

[ВВЕДЕНИЕ 3](#_Toc166347388)

[РЕШЕНИЕ 4](#_Toc166347389)

[1. Создание «source\_data.txt», чтение и запись данных 4](#_Toc166347390)

[2. Вычисление значения, полученного в результате деления ID на количество символов в ФИО 6](#_Toc166347391)

[3. Определение направления сортировки 7](#_Toc166347392)

[4. Формирование набора данных 8](#_Toc166347393)

[5. Сортировка набора данных 9](#_Toc166347394)

[6. Среднее арифметическое значение 15](#_Toc166347395)

[7. Среднее квадратическое значение 16](#_Toc166347396)

[ЗАКЛЮЧЕНИЕ 18](#_Toc166347397)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 19](#_Toc166347398)

[ЛИСТИНГ 20](#_Toc166347399)

# ВВЕДЕНИЕ

Данная рейтинговая работа предусматривает задание на отработку чтения файла, записи в файл, а также базовых алгоритмов на языке программирования Python. Из изначально подготовленных данных подготовим файл «source\_data.txt», который содержит в себе данные о выполняющем работу на двух строках. На первой строке - фамилия, имя и отчество, а на второй - ID. Задание состоит из нескольких пунктов, результат решения которых будет записан в файл «result.txt».

В первом пункте задания необходимо считать данные из текстового файла «source\_data.txt» и записать их в «result.txt».

Во втором пункте нужно получить число, получившееся в результате деления ID на количество символов в ФИО без учета пробелов.

В третьем пункте, в зависимости от предыдущего результата, необходимо выбрать метод сортировки, который будет использоваться в дальнейшем. Если число во втором пункте получилось четное – сортировка будет по возрастанию, в противном случае – по убыванию.

В четвертом пункте предстоит сформировать набор данных, получаемый из кодов Юникода каждого символа ФИО без учета пробелов и переведенный в десятичную форму.

В пятом пункте необходимо выполнить сортировку получившегося набора данных по методу из пункта три.

В шестом и седьмом пунктах нужно вычислить среднее арифметическое и среднее квадратическое значения набора данных соответственно.

При решении запрещается использовать встроенные методы сортировки. Также требуется написать два вида сортировки: пузырьком и шейкерную.

# РЕШЕНИЕ

## **Создание «source\_data.txt», чтение и запись данных**

Первым делом создаем файл «source\_data.txt». В условиях задачи четко оговорено, что всего строк в файле должно быть две (Рисунок 1).

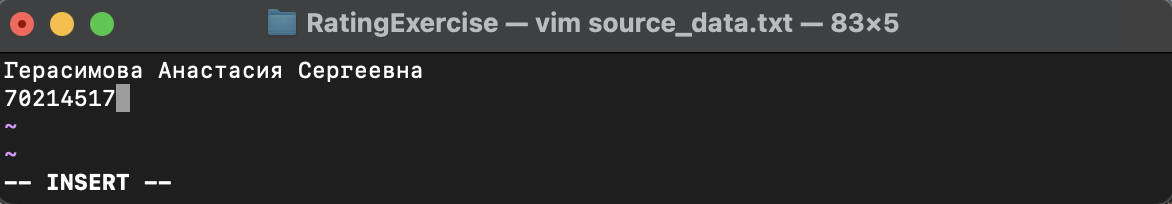


Рисунок 1 – Создание «source\_data.txt»

Далее создаем файл «main.py», откуда будет запускаться программа, в той же директории, что и файл «source\_data.txt» (Рисунок 2).

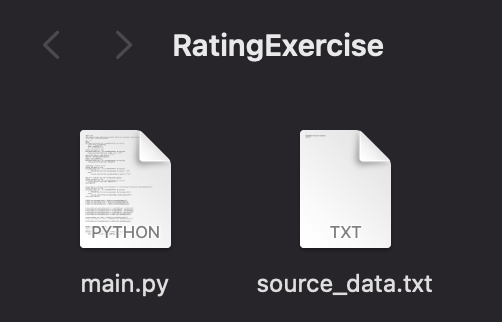


Рисунок 2 – Добавление main.py

Перейдем к непосредственному выполнению первого пункта задания. Нас просят прочитать данные из «source\_data.txt» и записать их в файл «result.txt». С этим нам поможет встроенная функция open(). После работы с файлом, его обязательно нужно закрыть с помощью метода close(). Но можно воспользоваться менеджером контекста with, который сделает это за нас. Открываем файл, получаем из него информацию, сохраняем ФИО в name, а номер ID – в id\_ (Рисунок 3):

with open("source\_data.txt", encoding="UTF-8") as file\_in:  
 lines = file\_in.readlines()  
 name = lines[0][:-1]  
 id\_ = int(lines[1])

Рисунок 3 – Чтение файла и сохранение данных в переменные

В функции open() первым аргументом передается относительный путь к файлу «source\_data.txt», вторым аргументом – именованным – передается тип кодировки – по условиям задачи, это Unicode. С помощью метода readlines() в переменную lines сохранится список с, в нашем случае, двумя элементами. Если выведем lines в консоль, то увидим (Рисунок 4):



Рисунок 4 – Вывод lines в консоль

Первый элемент – строка, содержащая ФИО (первая строчка из «source\_data.txt»), второй элемент – строка с ID (вторая строчка из «source\_data.txt»). Чтобы достать эти элементы из списка и поместить их в заранее созданные переменные name и id\_, используем индексы. Также стоит помнить, что в конце первого элемента стоит \n – переход на новую строку. Нам этот символ в дальнейшем здесь не понадобится, поэтому стоит обрезать строку с ФИО на один элемент с конца. Делаем это с помощью среза [:-1]. lines[1] мы привели к типу int, так как в дальнейшем нам понадобится производить расчеты с ID в виде числа, а не строки. Стоит заметить, что переменная id\_ названа с нижним подчеркиванием на конце, так как в Python уже имеется системная одноименная функция и правилом хорошего тона является использование уже имеющихся названий с добавлением нижнего подчеркивания в конце.

Таким образом мы поместили из файла «source\_data.txt» ФИО и ID в переменные name и id\_ соответственно. Теперь нам нужно создать файл «result.txt» и поместить туда полученную информацию.

Перед тем, как сделать это, создадим файл «text.py», куда мы положим весь текст, который нам предстоит записать в «result.txt». Получим (Рисунок 5):

ex\_1 = "1. Исходные данные: %s; ID: %d"  
ex\_2 = "2."  
ex\_3\_ascending = ("3. Направление сортировки: "  
 "по возрастанию, так как число %d - четное")  
ex\_3\_descending = ("3. Направление сортировки: "  
 "по убыванию, так как число %d - нечетное")  
ex\_4 = "4. Набор данных:"  
ex\_5\_ascending = "5. Отсортированный по возрастанию набор данных:"  
ex\_5\_descending = "5. Отсортированный по убыванию набор данных:"  
ex\_6 = "6. Среднее арифметическое значение:"  
ex\_7 = "7. Среднее квадратическое значение:"

Рисунок 5 – Файл text.py

Чтобы эти переменные были доступны в «main.py», необходимо сделать импорт «text». Сделаем это вначале файла (Рисунок 6):

import text

Рисунок 6 – Импорт text

Чтобы записать полученные данные в файл, воспользуемся уже знакомым with open, только изменим режим доступа к файлу, передав аргумент «w» - write (Рисунок 7):

with open("result.txt", "w", encoding="UTF-8") as file\_out:  
 file\_out.write(str(text.ex\_1 % (name, id\_)) + "\n")

Рисунок 7 – Запись данных в файл

Функция open(), если файла, с переданным ей названием, не существует, создаст его. Если файл с таким названием существует, то он перезапишется. Воспользовавшись методом write, передаем строку с нужным нам текстом и данными, добавляя в конец перенос на новую строку. Если мы запустим программу, то убедимся, что в этой же директории появился файл «result.txt», внутри которого записана строчка с нужной нам информацией (Рисунок 8):

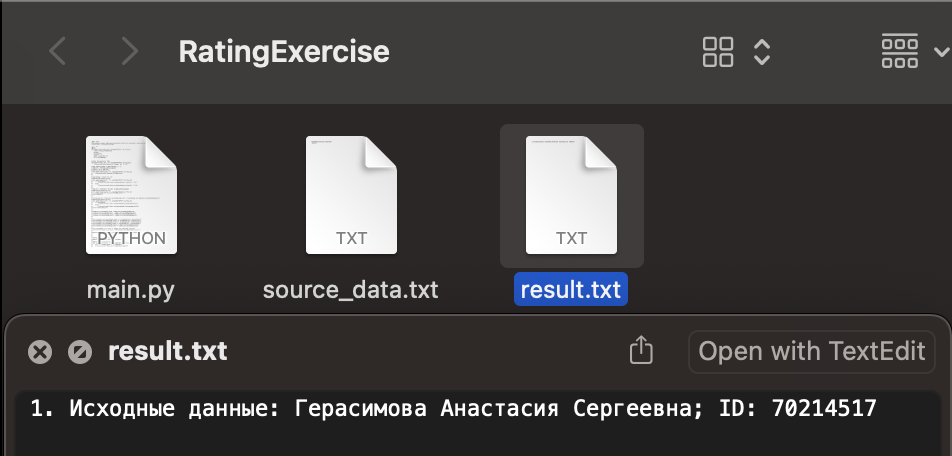


Рисунок 8 – Результат работы программы, пункт 1

Таким образом выполняется первый пункт задания.

## **Вычисление значения, полученного в результате деления ID на количество символов в ФИО**

ID было получено в пункте 1, осталось посчитать количество символов в ФИО. По условиям задачи, пробелы считаться не должны. С помощью метода строки replace мы можем заменить пробелы на пустоту. Затем, с помощью функции len() узнаем количество символов в получившейся строке. После вычислим результат деления ID на количество символов, с помощью операции «//» - целочисленного деления. Выглядеть это будет следующим образом (Рисунок 9):

name\_without\_spaces = name.replace(" ", "")  
name\_len = len(name\_without\_spaces)  
result = id\_ // name\_len

Рисунок 9 – Получение результата деления ID на количество символов в ФИО

Теперь нужно значение переменной result добавить к имеющейся информации в «result.txt». Воспользуемся with open, но теперь режим доступа к файлу будет «a» - append: добавим строчки в файл, а не перезапишем его. Не забываем добавить \n в конец строки. В результате получился код (Рисунок 10):

with open("result.txt", "a", encoding="UTF-8") as file\_out:  
 file\_out.write(f"{text.ex\_2} {result}\n")

Рисунок 10 – Записываем результат деления в файл «result.txt»

На этот момент, после запуска программы имеется результат (Рисунок 11):

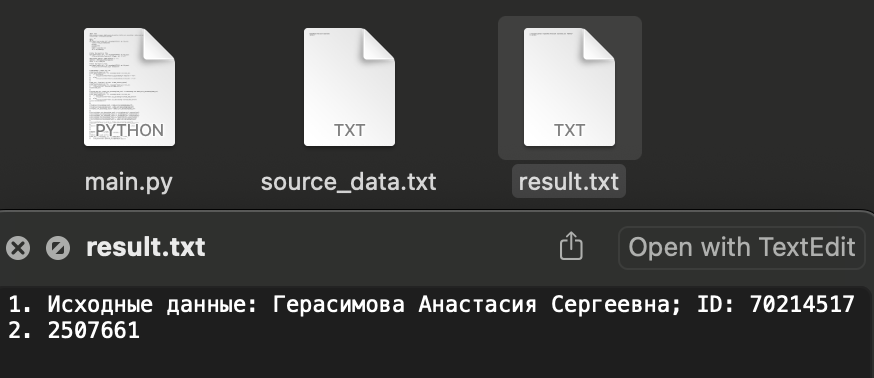


Рисунок 11 – Результат работы программы, пункт 2

Второй пункт задания выполнен.

## **Определение направления сортировки**

От результата из прошлого пункта зависит метод сортировки предстоящего задания. Если число четное – сортировка будет по возрастанию, в противном случае – по убыванию. Определим переменную is\_ascending, которая будет хранить в себе значение типа bool – четным числом является result или нет (Рисунок 12):

is\_ascending = result % 2 == 0

Рисунок 12 – Определение is\_ascending

Далее нужно записать соответствующую информацию в «result.txt». Если is\_ascending равно True – записываем, что сортировка будет выполнена по возрастанию, если is\_ascending равно False – по убыванию. Осуществим эту проверку в теле with open (Рисунок 13):

with open("result.txt", "a", encoding="UTF-8") as file\_out:  
 if is\_ascending:  
 file\_out.write(str(text.ex\_3\_ascending % result) + "\n")  
 else:  
 file\_out.write(str(text.ex\_3\_descending % result) + "\n")

Рисунок 13 – Запись информации в «result.txt» в зависимости от значения is\_ascending

После запуска программы в «result.txt» в результате выполнения программы получится (Рисунок 14):

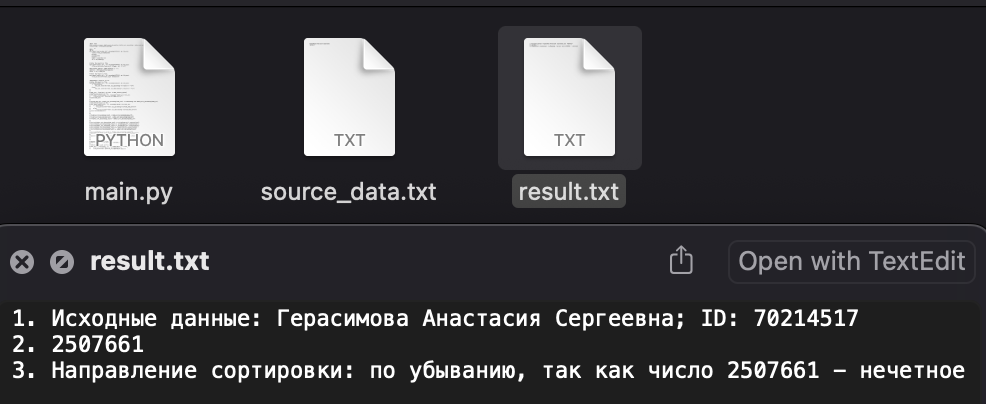


Рисунок 14 - Результат работы программы, пункт 3

Завершено выполнение третьего пункта задания.

## **Формирование набора данных**

Целью этого пункта является сформировать список целых чисел из кодов Юникода в десятичной системе счисления каждого символа ФИО, исключая пробелы. В пункте два мы уже создали name\_without\_spaces – это и есть ФИО без пробелов. Для получения Юникода из символа в десятичной форме существует встроенная функция ord(). Сформировать набор данных можно в одну строку, используя списочное выражение (Рисунок 15):

name\_utf = [ord(char) for char in name\_without\_spaces]

Рисунок 15 – Формирование набора данных и сохранение его в переменную name\_utf

Теперь переменная name\_utf хранит список кодов Юникода для каждой буквы из ФИО. Осталось только записать набор данных в «result.txt» (Рисунок 16):

with open("result.txt", "a", encoding="UTF-8") as file\_out:  
 file\_out.write(f"{text.ex\_4} {name\_utf}\n")

Рисунок 16 – Запись набора данных в «result.txt»

«result.txt» после отработки программы (Рисунок 17):

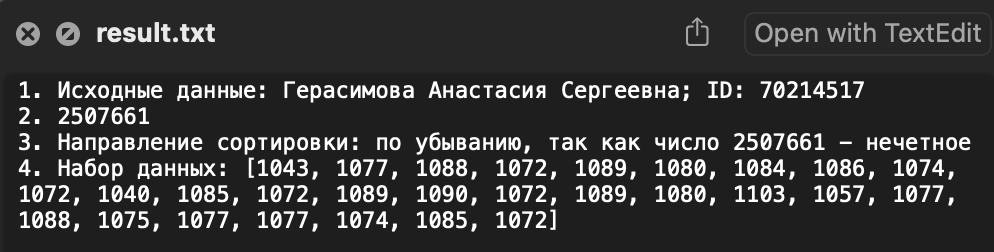


Рисунок 17 - Результат работы программы, пункт 4

Выполнен четвертый пункт задания.

## **Сортировка набора данных**

Теперь набор данных, который был получен в четвертом пункте, нужно отсортировать методом из условия пункта три. Создадим два алгоритма сортировки: пузырьком и шейкерную. Начнем с первой.

Создадим еще один дополнительный файл «resources.py», где будут находиться все алгоритмические функции. Определим функцию buble\_sort\_descending, в теле которой напишем алгоритм сортировки по убыванию. Функция будет принимать один аргумент типа list. Суть сортировки пузырьком заключается в том, что последовательно сравниваются два соседствующих элемента и если второй элемент больше (в нашем случае), чем первый, то они меняются местами. Реализация алгоритма включает в себя цикл внутри цикла. После каждого отрабатывания внутреннего цикла как минимум одно значение встанет на свое место. Блок-схема к алгоритму сортировки по убыванию выглядит так (Рисунок 18):

list lst

Вход

Рисунок 18 – Блок-схема алгоритма сортировки пузырьком (по убыванию)

Выход

Нет

Нет

Да

Да

i < len(lst) - 1

i += 1

j < len(lst) - 1 - i

j += 1

lst[j + 1], lst[j] = lst[j], lst[j + 1]

Да

Нет

lst[j] < lst[j + 1]

j = 0

i = 0

Переносим алгоритм на код, выглядеть это будет следующим образом (Рисунок 19):

def buble\_sort\_descending(lst):  
 for i in range(len(lst) - 1):  
 for j in range(len(lst) - 1 - i):  
 if lst[j + 1] > lst[j]:  
 lst[j + 1], lst[j] = lst[j], lst[j + 1]  
 return lst

Рисунок 19 – Реализация алгоритма сортировки пузырьком (по убыванию)

Первый цикл идет до значения range(len(lst) – 1) (до последнего не включая), так как условие внутреннего цикла построено таким образом, что с i равное индексу последнего элемента это условие не выполнится и внутренний цикл не отработает (range(len(lst – 1 - i)) станет равно нулю). Поэтому, чтобы избежать бессмысленной отработки внешнего цикла, ставим условие до предпоследнего индекса включительно. Второй, внутренний, цикл идет до значения range(len(lst) – 1 – i), таким образом в цикле не будут перебираться те значения, которые уже стоят на своем месте. При i = 0 – последний элемент в списке окажется на своем месте, при i = 1 – предпоследний элемент окажется на своем месте и так далее. Также условие внутреннего цикла должно быть строго меньше как минимум на единицу размера листа, так как в теле цикла идет проверка следующего за j элемента, а если j станет равно размеру листа, то проверка выйдет за пределы листа и программа прекратит свое выполнение досрочно с критической ошибкой.

Для сортировки пузырьком по возрастанию достаточно поменять условие внутри внутреннего цикла (Рисунок 20):

def buble\_sort\_ascending(lst):  
 for i in range(len(lst) - 1):  
 for j in range(len(lst) - 1 - i):  
 if lst[j] > lst[j + 1]:  
 lst[j + 1], lst[j] = lst[j], lst[j + 1]  
 return lst

Рисунок 20 – Реализация алгоритма сортировки пузырьком (по возрастанию)

Теперь мы проверяем, что следующий за j элемент меньше элемента j. Все остальное неизменно.

Сортировка пузырьком хорошо работает со списками, в которых небольшое количество элементов. В рабочей практике не применяется.

Разберем шейкерную сортировку. Эта сортировка отличается от предыдущей только тем, что сравнивание элементов идет и из начала списка в конец, и с конца в начало. Работает несущественно быстрее пузырьковой сортировки. Рассмотрим подробнее на блок-схеме (Рисунок 21):

Рисунок 21 – Блок-схема алгоритма шейкерной сортировки (по убыванию)

Выход

Нет

Нет

Нет

i -= 1

lst[i - 1], lst[i] = lst[i], lst[i - 1]

Да

Да

lst[i] > lst[i - 1]

i > start

i = end

end -= 1

Нет

Да

i += 1

lst[i + 1], lst[i] = lst[i], lst[i + 1]

Да

lst[i] < lst[i + 1]

i < end

i = start

Вход

list lst

start = 0

end = len(lst) - 1

Да

start < end

Нет

start += 1

В начале задаем указатели start и end, первый указывает на нулевой индекс, второй – на последний индекс списка. Входим в первый цикл: пока start < end. Задаем i = start и вторым, внутренним, циклом идем с начала списка до конца (до end) и попарно сравниваем элементы (как было в сортировке пузырьком). Если следующий от i элемент больше текущего – меняем эти два элемента местами. Доходим до конца внутреннего цикла, уменьшаем end на единицу (так как наименьшее число уже гарантировано находится на последнем месте списка, далее сравнивать его с другими элементами не имеет смысла). Теперь i берет значение end и происходит вход в третий цикл: пока i > start. Если предыдущий от i элемент меньше текущего, то меняем эти два элемента местами. После выхода из этого цикла увеличиваем start на единицу, так как наибольшее число уже гарантировано находится в начале списка. К моменту, как start станет равно end список гарантировано будет отсортирован.

В коде реализация выглядит следующим образом (Рисунок 22):

def shaker\_sort\_descending(lst):  
 start = 0  
 end = len(lst) - 1  
 while start < end:  
 for i in range(start, end):  
 if lst[i] < lst[i + 1]:  
 lst[i + 1], lst[i] = lst[i], lst[i + 1]  
 end -= 1  
 for i in range(end, start, -1):  
 if lst[i] > lst[i - 1]:  
 lst[i - 1], lst[i] = lst[i], lst[i - 1]  
 start += 1  
 return lst

Рисунок 22 - Реализация алгоритма шейкерной сортировки (по убыванию)

Для реализации шейкерной сортировки по возрастанию, следует во вложенных циклах поменять условия (Рисунок 23):

def shaker\_sort\_ascending(lst):  
 start = 0  
 end = len(lst) - 1  
 while start < end:  
 for i in range(start, end):  
 if lst[i] > lst[i + 1]:  
 lst[i + 1], lst[i] = lst[i], lst[i + 1]  
 end -= 1  
  
 for i in range(end, start, -1):  
 if lst[i] < lst[i - 1]:  
 lst[i - 1], lst[i] = lst[i], lst[i - 1]  
 start += 1  
 return lst

Рисунок 23 - Реализация алгоритма шейкерной сортировки (по возрастанию)

Чтобы убедиться в том, что алгоритмы работают корректно, стоит их проверить. В этом же файле напишем небольшую логику для тестирования. Нам не нужно, чтобы тесты отрабатывали при запуска файла main.py, поэтому воспользуемся проверкой специальной переменной \_\_name\_\_. Если она будет равна «\_\_main\_\_» — значит мы запускаем файл resources.py, если же мы запустим файл main.py, то \_\_name\_\_ в resources.py будет равна «resources». В конце файла «resources.py» делаем проверку \_\_name\_\_ на “\_\_main\_\_” и запускаем тесты (Рисунок 24):

def test(test\_func, is\_descending =False):  
 name = f"{test\_func.\_\_name\_\_} test"  
 lst\_1 = [3, 2, 5, 10, 15, 20, 0, -1]  
 lst\_2 = [0, 1, 2, 3]  
 lst\_3 = [9, 6, 5, 0, -1]  
 lst\_4 = [1]  
 for i, lst in enumerate([lst\_1, lst\_2, lst\_3, lst\_4]):  
 test\_sort\_result = test\_func(lst)  
 sorted\_result = sorted(lst, reverse=is\_descending)  
 result = test\_sort\_result == sorted\_result  
 if not result:  
 print(f"\n {name} {i + 1} is failed ❌")  
 print(f"\tExpected: {sorted\_result}")  
 print(f"\t{name} returned: {test\_sort\_result}\n")  
 else:  
 print(f"\n {name} {i + 1} is passed ✅", end="")  
 print()

def start\_tests():  
 test(buble\_sort\_ascending)  
 test(buble\_sort\_descending, True)  
 test(shaker\_sort\_ascending)  
 test(shaker\_sort\_descending, True)  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 start\_tests()

Рисунок 24 – Тестирование функций в «resources.py»

Функция test() принимает два аргумента: функцию, которую надо протестировать и информацию, по убыванию ли надо отсортировать список. По умолчанию is\_descending равен False, что значит сортирование будет по возрастанию. В теле записаны четыре списка, которые будут отправлены в тестируемую функцию и во встроенную функцию sorted(), затем их результаты сравниваются.

Запускаем файл «resources.py», получим (Рисунок 25):

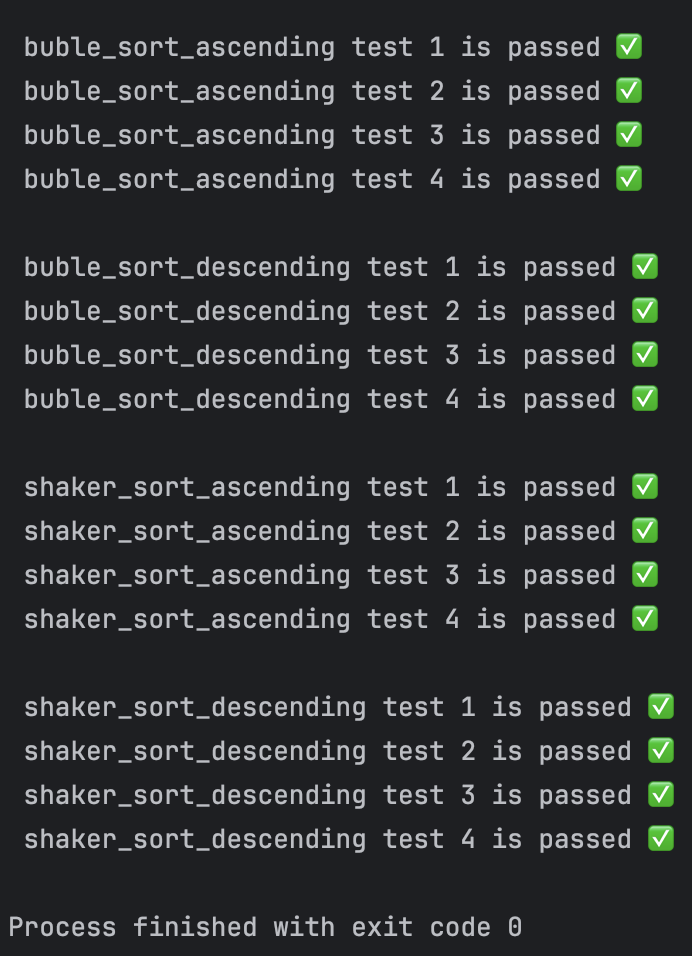


Рисунок 25 – Результат тестов

Результаты обоих методов сортировки в обоих направлениях соответствуют результатам встроенной функции sorted().

Функции для сортировки написаны и проверены, осталось их применить. В «main.py» в начале файла импортируем нужные нам функции из файла «resources.py» (Рисунок 26):

from resources import (buble\_sort\_ascending, buble\_sort\_descending,  
 shaker\_sort\_ascending, shaker\_sort\_descending)

Рисунок 26 – Импортирование нужных функций

В задании просят отсортировать набор данных двумя способами. Сделаем это, а затем создадим переменную sorted\_name\_utf, в которую положим результат сортировки (например, шейкером). Направление сортировки будет выбрано исходя из значения is\_ascending, которое мы получили в третьем пункте (Рисунок 27):

shaker\_sort\_ascending\_name\_utf = shaker\_sort\_ascending(name\_utf)  
buble\_sort\_ascending\_name\_utf = buble\_sort\_ascending(name\_utf)  
shaker\_sort\_descending\_name\_utf = shaker\_sort\_descending(name\_utf)  
buble\_sort\_descending\_name\_utf = buble\_sort\_descending(name\_utf)  
sorted\_name\_utf = shaker\_sort\_ascending\_name\_utf if is\_ascending \  
 else shaker\_sort\_descending\_name\_utf

Рисунок 27 – Сортирование набора данных

Осталось записать результат в файл «result.txt». В зависимости от выбранного направления сортировки, текст в файле будет отличаться (Рисунок 28):

with open("result.txt", "a", encoding="UTF-8") as file\_out:  
 if is\_ascending:  
 file\_out.write(f"{text.ex\_5\_ascending} {sorted\_name\_utf}\n")  
 else:  
 file\_out.write(f"{text.ex\_5\_descending} {sorted\_name\_utf}\n")

Рисунок 28 – Запись в файл результат сортировки

Теперь в файле «result.txt» записан следующий набор данных (Рисунок 29):

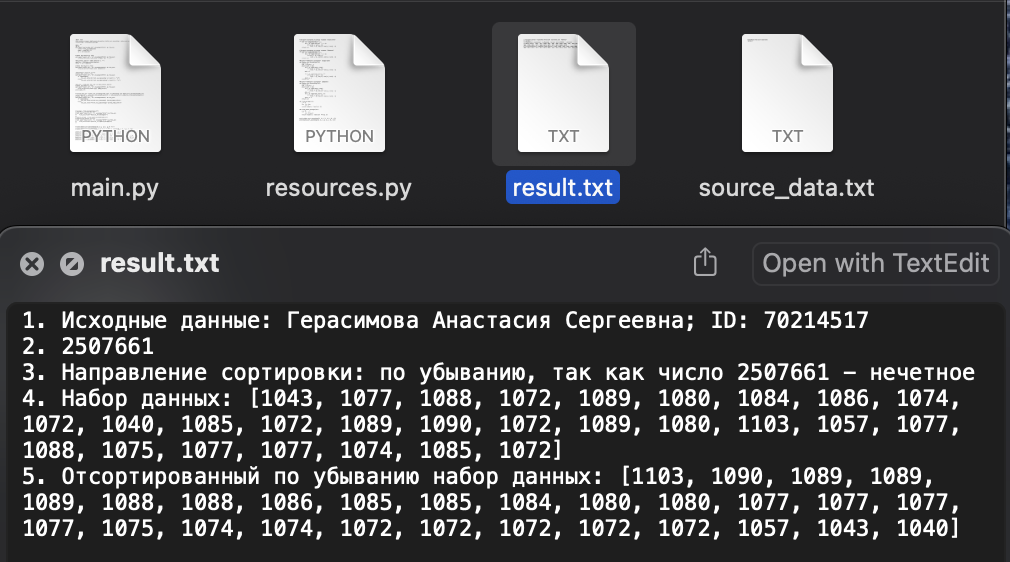


Рисунок 29 – Результат работы программы, пункт 5

Оба метода сортировки в двух направления реализованы, результаты записаны.

## **Среднее арифметическое значение**

В текущем пункте необходимо вычислить среднее арифметическое набора данных. Это значение находится по формуле:

где: – среднее арифметическое;

n – количество элементов.

Сумму всех элементов можно узнать с помощью встроенной функции sum(), количество элементов – len(). По условиям задачи, результат необходимо округлить до трех цифр после запятой в этом нам поможет round(). Получим (Рисунок 30):

average = round(sum(name\_utf) / len(name\_utf), 3)

Рисунок 30 – Нахождение среднего арифметического набора данных

Запишем результат (Рисунок 31):

with open("result.txt", "a", encoding="UTF-8") as file\_out:  
 file\_out.write(f"{text.ex\_6} {average}\n")

Рисунок 31 – Запись среднего арифметического в файл

Итог (Рисунок 32):

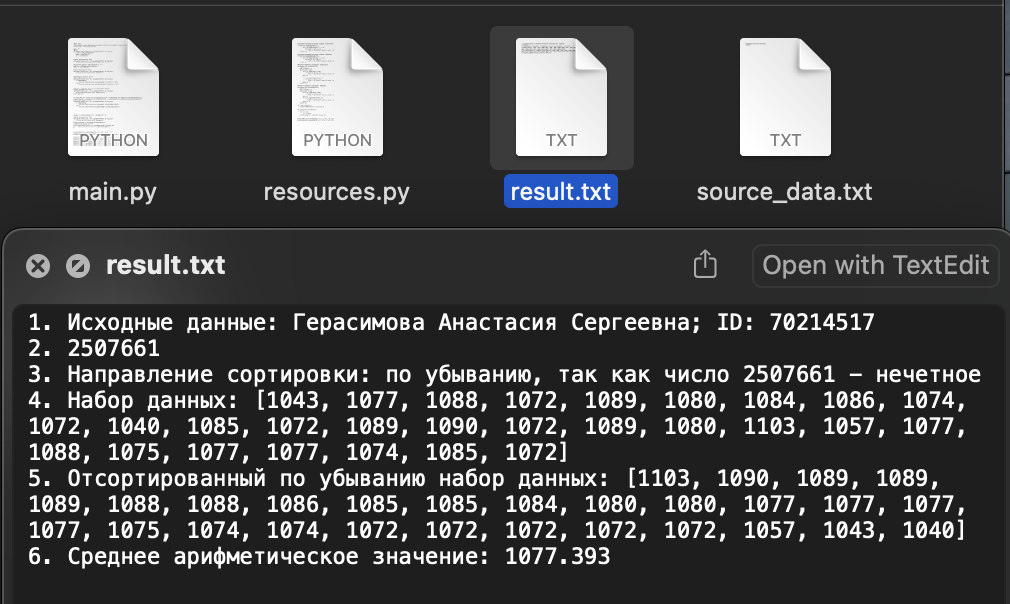


Рисунок 32 - Результат работы программы, пункт 6

Поставленная задача пункта шесть выполнена, переходим к финальному пункту.

## **Среднее квадратическое значение**

В завершающем пункте требуется вычислить среднее квадратическое значение набора данных, которое ищется по формуле:

где: s – среднее квадратическое;

– элемент;

n – количество элементов.

Результат так же нужно округлить до трех знаков. Сформируем список из квадратов набора данных, просуммируем все значения с помощью sum() разделим на количество символов, возведем в степень 0.5 и округлим результат до трех знаков. Получим код (Рисунок 33):

square\_average = round((sum([i \*\* 2 for i in name\_utf]) / len(name\_utf)) \*\* 0.5, 3)

Рисунок 33 - Нахождение среднего квадратического набора данных

Осталось записать результат (Рисунок 34):

with open("result.txt", "a", encoding="UTF-8") as file\_out:  
 file\_out.write(f"{text.ex\_7} {square\_average}")

Рисунок 34 - Запись среднего квадратического в файл

Поскольку это последняя запись, которую мы делаем в файле. Можно обойтись без \n на конце. Имеем (Рисунок 35):

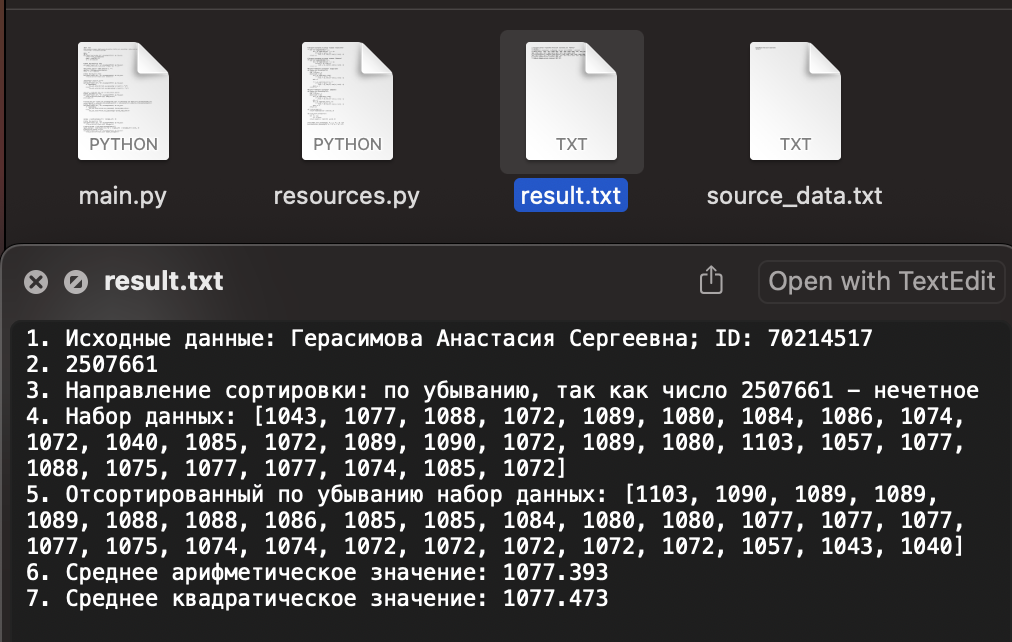


Рисунок 35 - Результат работы программы, пункт 7

Финальный пункт выполнен, задание завершено.

# ЗАКЛЮЧЕНИЕ

В процессе выполнения задания были проведены следующие работы:

- считывание данных из файла «source\_data.txt»

- разбиение полученных данных на нужные компоненты – name и id

- созданы два вспомогательных файла – «text.py» и «resources.py»

- задействованы встроенные функции и методы по работе со строками и числами

- использованы списочные выражения для формирования набора данных

- реализованы два алгоритма сортировки в двух направлениях

- произведена запись полученных результатов в файл «result.txt».

Все поставленные задачи были выполнены, был получен требуемый результат работы программы.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Сортировки. Сортировка пузырьком [электронный ресурс]. Режим доступа: http://sortings.github.io/sort\_types/bubble.html#:~:text=Сложность%20алгоритма%3A%20O(n²), (дата обращения 08.05.2024)
2. Хабр. Пузырьковая сортировка и все-все-все [электронный ресурс] / В. Макаров. Режим доступа: https://habr.com/ru/articles/204600/, (дата обращения 11.05.2024)
3. Яндекс Образование. Основные виды сортировок и примеры их реализации [электронный ресурс]. Режим доступа: https://education.yandex.ru/journal/osnovnye-vidy-sortirovok-i-primery-ikh-realizatsii, (дата обращения 08.05.2024)
4. Яндекс Образование. Основные виды сортировок и примеры их реализации [электронный ресурс]. Режим доступа: https://education.yandex.ru/journal/osnovnye-vidy-sortirovok-i-primery-ikh-realizatsii, (дата обращения 08.05.2024)
5. Яндекс Образование. Основы Python [электронный ресурс]. Режим доступа: https://education.yandex.ru/handbook/python, (дата обращения 07.05.2024)
6. CodeLessons. Шейкер сортировка в C++: принцип работы [электронный ресурс] / Д. Тилюпо. Режим доступа: https://codelessons.dev/ru/shejker-sortirovka-v-c-princip-raboty/, (дата обращения 08.05.2024)
7. Python 3 для начинающих. PEP 8 - руководство по написанию кода на Python работы [электронный ресурс]. Режим доступа: https://pythonworld.ru/osnovy/pep-8-rukovodstvo-po-napisaniyu-koda-na-python.html/, (дата обращения 11.05.2024)

# ЛИСТИНГ

«text.py»

ex\_1 = "1. Исходные данные: %s; ID: %d"  
ex\_2 = "2."  
ex\_3\_ascending = ("3. Направление сортировки: "  
 "по возрастанию, так как число %d - четное")  
ex\_3\_descending = ("3. Направление сортировки: "  
 "по убыванию, так как число %d - нечетное")  
ex\_4 = "4. Набор данных:"  
ex\_5\_ascending = "5. Отсортированный по возрастанию набор данных:"  
ex\_5\_descending = "5. Отсортированный по убыванию набор данных:"  
ex\_6 = "6. Среднее арифметическое значение:"  
ex\_7 = "7. Среднее квадратическое значение:"

«resources.py»

# Алгоритм сортировки по методу пузырька (возрастание)  
def buble\_sort\_ascending(lst):  
 for i in range(len(lst) - 1):  
 for j in range(len(lst) - 1 - i):  
 if lst[j] > lst[j + 1]:  
 lst[j + 1], lst[j] = lst[j], lst[j + 1]  
 return lst  
  
  
# Алгоритм сортировки по методу пузырька (убывание)  
def buble\_sort\_descending(lst):  
 for i in range(len(lst) - 1):  
 for j in range(len(lst) - 1 - i):  
 if lst[j + 1] > lst[j]:  
 lst[j + 1], lst[j] = lst[j], lst[j + 1]  
 return lst  
  
  
#Алгоритм "Шейкерная сортировка" (возрастание)  
def shaker\_sort\_ascending(lst):  
 start = 0  
 end = len(lst) - 1  
 while start < end:  
 for i in range(start, end):  
 if lst[i] > lst[i + 1]:  
 lst[i + 1], lst[i] = lst[i], lst[i + 1]  
 end -= 1  
  
 for i in range(end, start, -1):  
 if lst[i] < lst[i - 1]:  
 lst[i - 1], lst[i] = lst[i], lst[i - 1]  
 start += 1  
 return lst  
  
  
#Алгоритм "Шейкерная сортировка" (убывание)  
def shaker\_sort\_descending(lst):  
 start = 0  
 end = len(lst) - 1  
 while start < end:  
 for i in range(start, end):  
 if lst[i] < lst[i + 1]:  
 lst[i + 1], lst[i] = lst[i], lst[i + 1]  
 end -= 1  
 for i in range(end, start, -1):  
 if lst[i] > lst[i - 1]:  
 lst[i - 1], lst[i] = lst[i], lst[i - 1]  
 start += 1  
 return lst  
  
  
# Tests  
def test(test\_func, is\_descending=False):  
 name = f"{test\_func.\_\_name\_\_} test"  
 lst\_1 = [3, 2, 5, 10, 15, 20, 0, -1]  
 lst\_2 = [0, 1, 2, 3]  
 lst\_3 = [9, 6, 5, 0, -1]  
 lst\_4 = [1]  
 for i, lst in enumerate([lst\_1, lst\_2, lst\_3, lst\_4]):  
 test\_sort\_result = test\_func(lst)  
 sorted\_result = sorted(lst, reverse=is\_descending)  
 result = test\_sort\_result == sorted\_result  
 if not result:  
 print(f"\n {name} {i + 1} is failed ❌")  
 print(f"\tExpected: {sorted\_result}")  
 print(f"\t{name} returned: {test\_sort\_result}\n")  
 else:  
 print(f"\n {name} {i + 1} is passed ✅", end="")  
 print()  
  
  
def start\_tests():  
 test(buble\_sort\_ascending)  
 test(buble\_sort\_descending, True)  
 test(shaker\_sort\_ascending)  
 test(shaker\_sort\_descending, True)  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 start\_tests()

«main.py»

import text  
from resources import (buble\_sort\_ascending, buble\_sort\_descending,  
 shaker\_sort\_ascending, shaker\_sort\_descending)  
  
# №1  
with open("source\_data.txt", encoding="UTF-8") as file\_in:  
 lines = file\_in.readlines()  
 name = lines[0][:-1]  
 id\_ = int(lines[1])  
  
with open("result.txt", "w", encoding="UTF-8") as file\_out:  
 file\_out.write(str(text.ex\_1 % (name, id\_)) + "\n")  
  
  
# №2  
name\_without\_spaces = name.replace(" ", "")  
name\_len = len(name\_without\_spaces)  
result = id\_ // name\_len  
  
with open("result.txt", "a", encoding="UTF-8") as file\_out:  
 file\_out.write(f"{text.ex\_2} {result}\n")  
  
  
# №3  
is\_ascending = result % 2 == 0  
  
with open("result.txt", "a", encoding="UTF-8") as file\_out:  
 if is\_ascending:  
 file\_out.write(str(text.ex\_3\_ascending % result) + "\n")  
 else:  
 file\_out.write(str(text.ex\_3\_descending % result) + "\n")  
  
  
# №4  
name\_utf = [ord(char) for char in name\_without\_spaces]  
  
with open("result.txt", "a", encoding="UTF-8") as file\_out:  
 file\_out.write(f"{text.ex\_4} {name\_utf}\n")  
  
  
# №5  
shaker\_sort\_ascending\_name\_utf = shaker\_sort\_ascending(name\_utf)  
buble\_sort\_ascending\_name\_utf = buble\_sort\_ascending(name\_utf)  
shaker\_sort\_descending\_name\_utf = shaker\_sort\_descending(name\_utf)  
buble\_sort\_descending\_name\_utf = buble\_sort\_descending(name\_utf)  
sorted\_name\_utf = shaker\_sort\_ascending\_name\_utf if is\_ascending \  
 else shaker\_sort\_descending\_name\_utf  
  
with open("result.txt", "a", encoding="UTF-8") as file\_out:  
 if is\_ascending:  
 file\_out.write(f"{text.ex\_5\_ascending} {sorted\_name\_utf}\n")  
 else:  
 file\_out.write(f"{text.ex\_5\_descending} {sorted\_name\_utf}\n")  
  
  
# №6  
average = round(sum(name\_utf) / len(name\_utf), 3)  
  
with open("result.txt", "a", encoding="UTF-8") as file\_out:  
 file\_out.write(f"{text.ex\_6} {average}\n")  
  
  
# №7  
square\_average = round((sum([i \*\* 2 for i in name\_utf])  
 / len(name\_utf)) \*\* 0.5, 3)  
  
with open("result.txt", "a", encoding="UTF-8") as file\_out:  
 file\_out.write(f"{text.ex\_7} {square\_average}")