МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение

высшего образования

«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Департамент цифровых, робототехнических  
систем и электроники

# Отчет к практической работе №1

Тема:«Решение алгоритмических задач. Введение в инструменты и критерии оценки»  
Дисциплина: **«Анализ сложности алгоритмов»**

|  |  |  |
| --- | --- | --- |
|  |  | **Выполнил:** |
|  |  | Студент группы ПИЖ-б-о-23-1 Направление подготовки: 09.03.04 «Программная инженерия»  Герасименко Константин Васильевич  **Проверил:**  Щеголев Алексей Алексеевич |

**Цель работы**: Настроить рабочее окружение, освоить базовые операции ввода/вывода, написать и протестировать первую программу. Научиться оценивать сложность отдельных операций и всей программы, проводить эмпирические замеры времени выполнения и визуализировать результаты.

Для выполнения практической работы необходимо выполнить следующие шаги:

1. Выбор среды разработки
2. Создание проекта
3. Написание и анализ кода
4. Тестирование
5. Эмпирический анализ производительности
6. Анализ результатов
7. Вывод результатов
8. Требования к коду
9. Контроль версий

Шаг 1

Для решения практической работы была выбрана среда разработки VS CODE

Шаг 2

Создание файла sum\_analisis.py и numbers.txt, необходимый для хранения суммированный чисел (рисунок 1)



Рисунок 1. Создание проекта

Шаг 3

Была написана функция sum\_from\_file, открывающая файл и считывающая с него строку. Далее функция разделяет строку на части по пробелам, преобразуя первые два элемента в целые числа и возвращая их сумму. В основном блоке программа вызывает эту функцию и выводит результат на экран (рисунок 2)

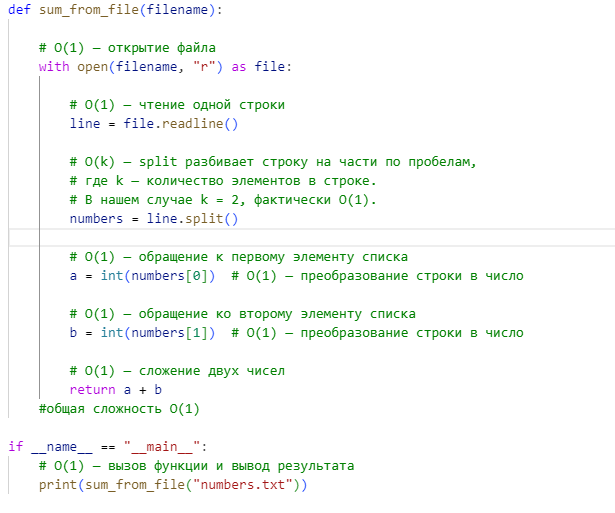


Рисунок 2. Код программы

Асимптотическая сложность алгоритма:

1. Открытие файла — **O(1)**
2. Чтение одной строки — **O(1)**
3. Разделение строки на элементы (split) — **O(k),** где k — количество чисел в строке. В данной задаче k = 2, поэтому **O(1).**
4. Преобразование двух строк в числа — **O(1)**
5. Сложение двух чисел — **O(1)**

Шаг 4

Чтобы протестировать алгоритм сложения чисел, была написана функция для замера времени выполнения (рисунок 3)

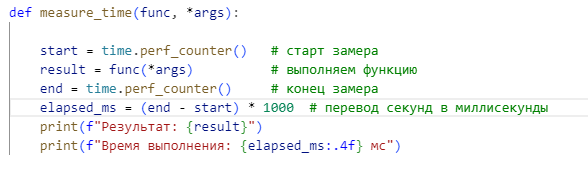


Рисунок 3. Функция замера времени

Сложение малых положительных чисел – 5 и 6. Результат: 0.1409мс (рисунок 4)

Сложение больших положительных чисел – 106578942 и 9986512340. Результат: 0.1563мс (рисунок 5)

Сложение отрицательных малых чисел – -12 и -9. Результат: 0.1519мс (рисунок 6)

Сложение отрицательных больших чисел – -125596421378 и -96661485123015. Результат: 0.1703мс (рисунок 7)

Сложение отрицательного малого и положительного малого чисел – -12 и 9. Результат: 0.1178мс(рисунок 8)

Сложение отрицательного малого и положительно большого чисел – -16 и 954684621654652187. Результат: 0.0994мс (рисунок 9)

Сложение отрицательного большого и положительно малого чисел – -16549684654168546516545 и 14. Результат: 0.1173мс (рисунок 10)

Сложение отрицательного большого и положительного большого чисел – -16549684654168546516545 и 144146546541685465488462. Результат: 0.0901мс (рисунок 11)



Рисунок 4

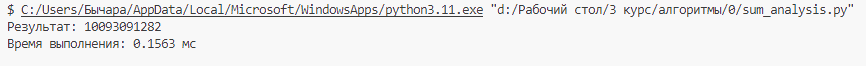


Рисунок 5



Рисунок 6



Рисунок 7



Рисунок 8



Рисунок 9



Рисунок 10



Рисунок 11

Результаты замера времени сложения чисел представлены в таблице 1

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Тип складываемых чисел | Значение чисел | | | | Количество времени выполнения сложения, мс |
| Малые положительные | 5 | 6 | | | 0,1409 |
| Большие положительные | 106578942 | 9986512340 | | | 0.1563 |
| Малые отрицательные | -12 | -9 | | | 0.1519 |
| Большие отрицательные | -125596421378 | -96661485123015 | | | 0.1703 |
| Малое отрицательное и малое положительное | -12 | | 9 | | 0.1178 |
| Малое отрицательное и большое положительное | -16 | | 954684621654652187 | | 0.0994 |
| Большое отрицательное и Малое положительное | -16549684654168546516545 | | | 14 | 0.1173 |
| Большое отрицательное и большое положительное | -16549684654168546516545 | | 144146546541685465488462 | | 0.0901 |

Таблица 1

Шаг 5

Для выполнения этого шага была переписана функция замера времени, а также была написана новая функция, считающая сумму элементов массива. Функция считает время 5 раз, затем выводит среднее арифметическое время выполнения (рисунок 12)

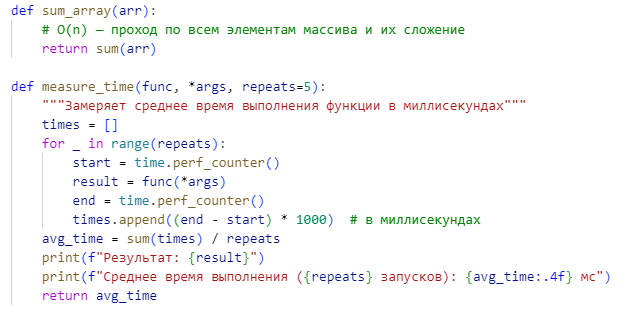


Рисунок 12. Программа для 5-ого шага

Для теста было создано три массива: 1 тыс. элементов, 100 тыс. элементов и 1м элементов (рисунок 13, 14)

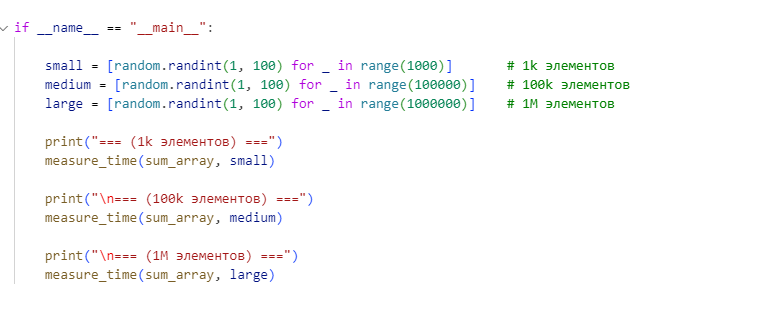


Рисунок 13. Входная точка программы

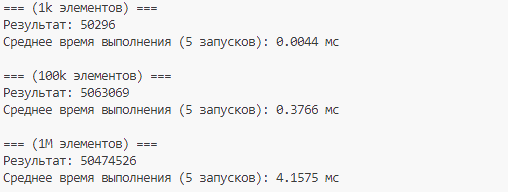


Рисунок 14. Результат запуска алгоритма

В результате работы алгоритма был построен график (рисунок 15, 16)



Рисунок 15. Код для постройки графика

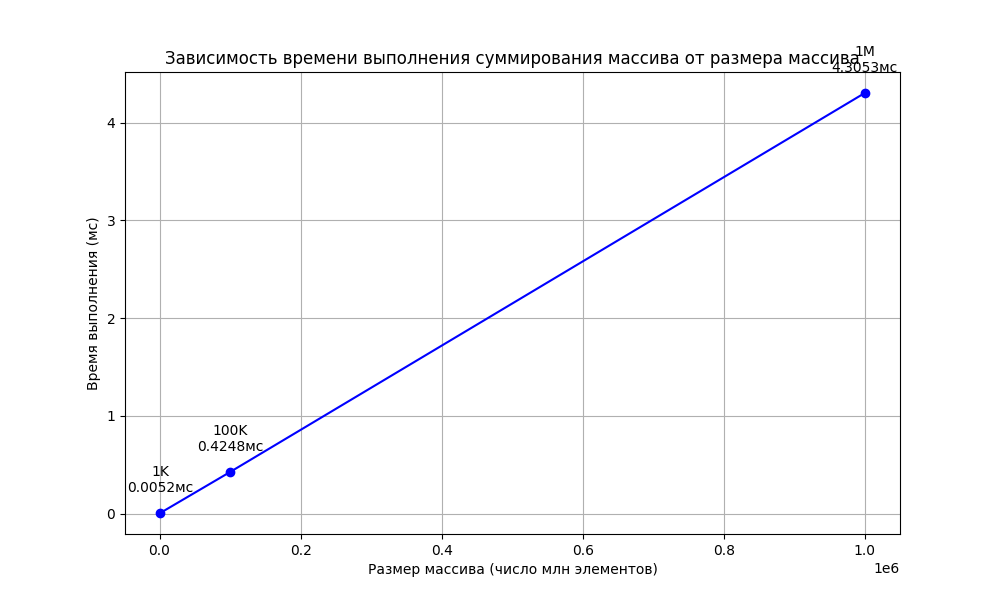


Рисунок 16. График измерения времени выполнения функции

Шаг 6

Рассмотрим подробнее временную сложность каждой операции внутри функции sum\_array, которая суммирует элементы массива.

Для вычисления суммы всех элементов используется встроенная функция sum. Эта функция проходит по каждому элементу массива ровно один раз, выполняя операцию сложения каждого элемента с накопленным результатом. Таким образом, для массива длиной n потребуется n шагов сложения.

Каждая операция сложения является элементарной операцией и выполняется за константное время (O(1)). Следовательно, выполнение всех n операций сложения требует O(n) времени.

Рассмотрев график на рисунке 16, мы видим линейную зависимость времени выполнения от размера массива, что подтверждает нашу теорию о сложности O(n). Чем больше размер массива, тем дольше длится процесс подсчета суммы.

При увеличении размера массива мы наблюдаем пропорциональное увеличение времени выполнения, что соответствует нашей оценке сложности. Если масштабировать массив вдвое, время выполнения также примерно удваивается, что характерно именно для функций с линейной сложностью.

Шаг 7