Тема 00: Решение алгоритмических задач. Введение в инструменты и критерии оценки.

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.

Теория (кратко):

- **Алгоритм** — это последовательность шагов для решения определенной задачи.
- **Структуры данных** — способы организации данных для эффективной работы с ними.
- **Оценка решения:** Правильность работы алгоритма и его **эффективность** (скорость работы и потребление памяти) — ключевые критерии качества.
- **Ввод и вывод данных:** Стандартные потоки ввода/вывода (`stdin`, `stdout`). Работа с консолью.
- **Сложность алгоритма:** Количество операций, выполняемых алгоритмом, как функция от объема входных данных (N). Описывается с помощью **О-нотации (О-большое)**.

Практика (подробно):

Задание: Написать программу, которая:

1. Считывает два целых числа, `a` и `b`, из стандартного потока ввода.
2. Вычисляет их сумму.
3. Выводит результат в стандартный поток вывода.

Шаги выполнения:

1. **Выбор среды разработки:** Установить и настроить IDE (например, PyCharm, VS Code) или использовать онлайн-компилятор.
2. **Создание проекта:** Создать новый проект и файл с именем `sum_analysis.py`.
3. **Написание и анализ кода:**
    - Реализовать чтение чисел из файла (с кратким выводом в консоль содержимого файла с поснением).
    - Реализовать вычисление суммы.
    - Реализовать вывод результата.
    - **После каждой строки кода в комментарии указать её асимптотическую сложность.**
    - **В конце функции в комментарии указать общую сложность алгоритма.**
4. **Тестирование:** Протестировать программу на различных входных данных (малые, большие, отрицательные числа).
5. **Эмпирический анализ производительности:**
    - Написать функцию-обертку для замера времени выполнения.
    - Провести замеры времени выполнения для усложненной версии задачи (суммирования всего массива чисел) на наборах данных разного размера.
    - Построить график зависимости времени выполнения/количества операций от размера входных данных. ВАЖНО все замеры проводить на одной и той же вычислительной машине.
6. **Анализ результатов:** Сравнить теоретическую оценку сложности с практическими результатами.

7. **Вывод результатов:** Оформить вывод всех результатов (теоритических и практических выводов, а также необходимой графической информации) в соотвествующий файл (поддерживающий открытие для чтения в режиме онлайн болшинством удаленных репозиториев)
8. **Требования к коду:** Код должен соответсвовать стилю написания кода на Python PEP8. Полное соблюдение требований в соотвесвиии с прилагаемым файлом 00_lab00-Требования к коду.pdf
9. **Контроль версий:** Стратегия ветвления рекомендуеться GitHub Flow.

Пример кода на Python с анализом сложности:

```
# sum_analysis.py
import timeit
import matplotlib.pyplot as plt
import random

# Исходная простая задача
def calculate_sum():
    """Считает сумму двух введенных чисел."""
    a = int(input())  # O(1) - чтение одной строки и преобразование
    b = int(input())  # O(1)
    result = a + b    # O(1) - арифметическая операция
    print(result)     # O(1) - вывод одной строки
    # Общая сложность функции: O(1)

# calculate_sum()  # Раскомментировать для проверки исходной задачи

# УСЛОЖНЕННАЯ ЗАДАЧА ДЛЯ АНАЛИЗА ПРОИЗВОДИТЕЛЬНОСТИ
# Суммирование N чисел для демонстрации линейной сложности O(N)
def sum_array(arr):
    """Возвращает сумму всех элементов массива.

    Сложность: O(N), где N - длина массива.
    """
    total = 0                 # O(1) - инициализация переменной
    for num in arr:           # O(N) - цикл по всем элементам массива
        total += num          # O(1) - операция сложения и присваивания
    return total              # O(1) - возврат результата
    # Общая сложность: O(1) + O(N) * O(1) + O(1) = O(N)

# Функция для замера времени выполнения
def measure_time(func, data):
    """Измеряет время выполнения функции в миллисекундах."""
    start_time = timeit.default_timer()
    func(data)
    end_time = timeit.default_timer()
    return (end_time - start_time) * 1000  # Конвертация в миллисекунды

# Характеристики ПК (заполнить своими данными)
pc_info = """
Характеристики ПК для тестирования:
- Процессор: Intel Core i7-10750H @ 2.60GHz
- Оперативная память: 16 GB DDR4
- ОС: Windows 11
```

```
- Python: 3.9.7
"""
print(pc_info)

# Проведение экспериментов
sizes = [1000, 5000, 10000, 50000, 100000, 500000]  # Размеры массивов
times = []  # Время выполнения для каждого размера

print("Замеры времени выполнения для алгоритма суммирования массива:")
print("{:>10} {:>12} {:>15}".format("Размер (N)", "Время (мс)", "Время/N (мкс)"))

for size in sizes:
    # Генерация случайного массива заданного размера
    data = [random.randint(1, 1000) for _ in range(size)]

    # Замер времени выполнения (усреднение на 10 запусках)
    execution_time = timeit.timeit(lambda: sum_array(data), number=10) * 1000 / 10

    times.append(execution_time)
    time_per_element = (execution_time * 1000) / size if size > 0 else 0  # мкс на
элемент

    print("{:>10} {:>12.4f} {:>15.4f}".format(size, execution_time,
time_per_element))

# Построение графика
plt.figure(figsize=(10, 6))
plt.plot(sizes, times, 'bo-', label='Измеренное время')
plt.xlabel('Размер массива (N)')
plt.ylabel('Время выполнения (мс)')
plt.title('Зависимость времени выполнения от размера массива\nСложность: O(N)')
plt.grid(True, which='both', linestyle='--', linewidth=0.5)
plt.legend()
plt.savefig('time_complexity_plot.png', dpi=300, bbox_inches='tight')
plt.show()

# Дополнительный анализ: сравнение с теоретической оценкой
print("\nАнализ результатов:")
print("1. Теоретическая сложность алгоритма: O(N)")
print("2. Практические замеры показывают линейную зависимость времени от N")
print("3. Время на один элемент примерно постоянно (~{:.4f}
мкс)".format(time_per_element))
```

Критерии оценки:

- **Оценка «3» (удовлетворительно):**
    - Программа написана и компилируется/запускается без ошибок.
    - Программа корректно работает на примере 2 и 3.
    - В коде присутствуют комментарии с оценкой сложности для основных операций.
- **Оценка «4» (хорошо):**

    - Выполнены все критерии на «3».
    - Код хорошо отформатирован, читаем и содержит комментарии.
    - Реализована функция замера времени выполнения.
    - Проведены замеры для 3-4 различных размеров входных данных.
    - Построен график зависимости времени выполнения от N.

- **Оценка «5» (отлично):**

    - Выполнены все критерии на «4».
    - Приведены характеристики ПК для тестирования.
    - Замеры времени проведены с усреднением результатов (многократный запуск).
    - На графике присутствуют подписи осей, заголовок, сетка.
    - В выводе присутствует анализ результатов, сравнение с теоретической оценкой сложности.
    - Программа корректно обрабатывает большие объемы данных (демонстрирует понимание практических ограничений).

Рекомендованная литература

1. **Юрий Петров: "Программирование на Python"** — онлайн-курс и учебные материалы.
    - *Ссылка для изучения:* https://www.yuripetrov.ru/edu/python/index.html
2. **Кормен, Т., Лейзерсон, Ч., Ривест, Р., Штайн, К.** Алгоритмы: построение и анализ, 3-е издание. — М.: Вильямс, 2022. — 1328 с.
    - *(Оригинальное название: Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C. Introduction to Algorithms, 3rd Edition)*
3. **Скиена, С.** Алгоритмы. Руководство по разработке, 3-е издание. — СПб.: БХВ-Петербург, 2022. — 720 с.
    - *(Оригинальное название: Skiena, Steven S. The Algorithm Design Manual, 3rd ed.)*