Отчет по лабораторной работе 0

Решение алгоритмических задач. Введение в инструменты и критерии оценки.

Дата: 2025-09-19
Семестр: 3 курс 1 полугодие - 5 семестр
Группа: ПИЖ-б-о-23-2
Дисциплина: Анализ сложности алгоритмов
Студент: Мальцев Виталий Игоревич

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

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.

```
# 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):
```

```python
    """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 i5-12500H @ 2.50GHz
- Оперативная память: 32 GB DDR4
- ОС: Windows 11
- Python: 3.12
"""
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

    # Замер времени выполнения (усреднение на 10 запусках)
    data = [random.randint(1, 1000) for _ in range(size)]

    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('./ПР_0/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))
```

<image src="time_complexity_plot.png">
<div style="display:flex; justify-content:center;">
<image src="out.png">
</div>

Контрольные вопросы

1. Что такое асимптотическая сложность алгоритма и зачем она нужна?

Асимптотическая сложность алгоритма — это оценка роста времени выполнения или объёма памяти, требуемого алгоритмом, в зависимости от размера входных данных при стремлении этого размера к бесконечности. Обычно выражается с помощью нотации O («большое O»), Ω («омега») или Θ («тета»).

Зачем нужна:

- Позволяет сравнивать эффективность алгоритмов независимо от аппаратного обеспечения и языка программирования.
- Помогает понять, как алгоритм масштабируется с ростом входных данных.
- Упрощает анализ, игнорируя константы и младшие члены, что делает оценку универсальной для больших n.

---

2. Объясните разницу между O(1), O(n) и O(log n). Приведите примеры алгоритмов с такой сложностью.

- **O(1)** — константная сложность. Время выполнения не зависит от размера входных данных.

    - *Пример:* доступ к элементу массива по индексу, операция push/pop в стеке (при реализации на массиве).

- **O(n)** — линейная сложность. Время выполнения пропорционально размеру входных данных.
    - *Пример:* линейный поиск в неупорядоченном массиве, обход всех элементов списка.

- **O(log n)** — логарифмическая сложность. Время выполнения растёт пропорционально логарифму размера данных (обычно $\log_2 n$).
    - *Пример:* бинарный поиск в отсортированном массиве, поиск в сбалансированном бинарном дереве.

---

3. В чем основное отличие линейного поиска от бинарного? Какие предварительные условия необходимы для выполнения бинарного поиска?

Основное отличие:

- Линейный поиск последовательно проверяет каждый элемент до тех пор, пока не найдёт нужный или не дойдёт до конца. Сложность — O(n).
- Бинарный поиск работает только с отсортированными данными и на каждом шаге сокращает область поиска вдвое, используя сравнение со средним элементом. Сложность — O(log n).

Предварительные условия для бинарного поиска:

- Массив (или другая структура данных) должен быть **отсортирован** (по возрастанию или убыванию).
- Должна быть возможность **быстрого доступа к произвольному элементу** (например, массив или вектор, а не связный список без индексации).

---

4. Почему на практике время выполнения алгоритма может отличаться от теоретической оценки O-большое?

Теоретическая оценка O-большое:

- Игнорирует константы и младшие члены.
- Предполагает идеальные условия и асимптотическое поведение при $n \to \infty$.

Практические причины расхождения:

- **Константы:** Алгоритм O(n) с большой константой может работать медленнее, чем $O(n^2)$ с маленькой константой при малых n.
- **Ограничения памяти и кэширования:** Алгоритмы с хорошей локальностью данных могут работать быстрее из-за кэша CPU.
- **Накладные расходы:** Вызовы функций, выделение памяти, управление

потоками могут замедлять работу.
- **Размер данных:** При малых n асимптотика не имеет значения — важны реальные коэффициенты.
- **Влияние ОС и других процессов:** Конкурентное использование ресурсов может влиять на время выполнения.

---

5. Как экспериментально подтвердить, что сложность алгоритма равна O(n) или O(log n)? Опишите план эксперимента.

План эксперимента:

Шаг 1: Подготовка

- Реализуйте алгоритм.
- Подготовьте набор тестовых данных разного размера: $n_1, n_2, \ldots, n_k$ (например, 100, 1000, 10000, 100000).
- Для каждого размера n сгенерируйте несколько наборов данных (например, 10–100) и усредните время выполнения.

Шаг 2: Измерение времени

- Запустите алгоритм на каждом наборе данных.
- Измерьте время выполнения (в миллисекундах или наносекундах) с помощью таймера (например, `time` в Python, `System.nanoTime()` в Java).
- Запишите результаты в таблицу: `n | t(n)`.

Шаг 3: Анализ

- Постройте график: ось X — размер данных n, ось Y — время выполнения `t(n)`.
- Для O(n): график должен быть примерно линейным (прямая линия).
- Для O(log n): график должен расти очень медленно, почти горизонтально; можно построить график `t(n)` vs `log(n)` — он должен быть линейным.

Шаг 4: Проверка через отношение

- Рассчитайте отношение времени к теоретической сложности:
  - Для O(n): `t(n) / n` должно быть примерно постоянным.
  - Для O(log n): `t(n) / log(n)` должно быть примерно постоянным.
- Если значения стабильны — гипотеза подтверждается.

Шаг 5: Визуализация и вывод

- Постройте графики и выведите средние отношения.
- Сделайте вывод: если зависимость соответствует ожидаемой, то сложность подтверждена экспериментально.