САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ
ФАКУЛЬТЕТ ИНФОКОММУНИКАЦИОННЫХ ТЕХНОЛОГИЙ

Отчет по лабораторной работе №0

по курсу «Алгоритмы и структуры данных»

Тема: Введение. Основы ввода-вывода. Числа Фибоначчи
Вариант 1

Выполнил:
Бен Шамех Абделазиз
Группа: K3239

Проверила:
Ромакина Оксана Михайловна

Санкт-Петербург
2025 г.

Содержание

1 Задание 1. Ввод-вывод

1.1 Условие

Выполнить задачи по базовому вводу-выводу:

1. Задача $a + b$ (Консоль).
2. Задача $a + b^2$ (Консоль).
3. Задача $a + b$ с использованием файлов (`input.txt`, `output.txt`).
4. Задача $a + b^2$ с использованием файлов.

1.2 Листинг кода (Задачи 1.3 и 1.4)

```
# Task 1.3: a + b using files
def task_1_3():
    with open('input.txt', 'r') as f:
        a, b = map(int, f.read().split())
    with open('output.txt', 'w') as f2:
        f2.write(str(a + b))

# Task 1.4: a + b^2 using files
def task_1_4():
    with open('input.txt', 'r') as f:
        a, b = map(int, f.read().split())
    with open('output.txt', 'w') as f2:
        f2.write(str(a + b**2))
```

1.3 Результаты выполнения (на примере задачи 1.3)

Пример:

- **Input (input.txt):** `130 61`
- **Output (output.txt):** `191`

Метрики эффективности:

Сценарий	Время выполнения	Затраты памяти
Нижняя граница	0.00001 sec	4.10 Mb
Пример из задачи	0.00002 sec	4.10 Mb

2 Задание 2. Число Фибоначчи

2.1 Условие

Разработать эффективный алгоритм для подсчета чисел Фибоначчи F_n. Ограничение: $0 \leq n \leq 45$. Работа через файлы.

2.2 Листинг кода

```python
def calc_fib(n):
    if n <= 1:
        return n
    fib = [0] * (n + 1)
    fib[1] = 1
    for i in range(2, n + 1):
        fib[i] = fib[i-1] + fib[i-2]
    return fib[n]

def solve():
    with open('input.txt', 'r') as f:
        n = int(f.read().strip())
    with open('output.txt', 'w') as f2:
        f2.write(str(calc_fib(n)))
```

2.3 Объяснение решения

Наивный рекурсивный алгоритм имеет экспоненциальную сложность $O(2^n)$, что недопустимо при $n = 45$. Реализован итеративный алгоритм с использованием массива (динамическое программирование), работающий за линейное время $O(n)$.

2.4 Результаты выполнения

- **Input (input.txt):** `10` → **Output:** `55`
- **Input (input.txt):** `45` → **Output:** `1134903170`

Метрики эффективности:

Сценарий	Время выполнения	Затраты памяти
Нижняя граница	0.00002 sec	4.12 Mb
Пример (n=10)	0.00003 sec	4.12 Mb
Максимум (n=45)	0.00005 sec	4.15 Mb

3 Задание 3. Последняя цифра числа Фибоначчи

3.1 Условие

Найти последнюю цифру большого числа Фибоначчи ($F_n \pmod{10}$). Ограничение: $0 \leq n \leq 10^7$. Время: 5 сек. Память: 512 мб.

3.2 Листинг кода

```
def get_fib_last_digit(n):
    if n <= 1:
        return n
    a, b = 0, 1
    for _ in range(2, n + 1):
        a, b = b, (a + b) % 10
    return b

def solve_task3():
    with open('input.txt', 'r') as f:
        n = int(f.read().strip())
    with open('output.txt', 'w') as f2:
        f2.write(str(get_fib_last_digit(n)))
```

3.3 Объяснение решения

Для нахождения последней цифры нет необходимости вычислять всё число целиком. Достаточно на каждом шаге итерации брать остаток от деления на 10. Это предотвращает переполнение памяти и ускоряет вычисления.

3.4 Результаты выполнения

- **Input:** `331` → **Output:** `9`
- **Input:** `327305` → **Output:** `5`

Метрики эффективности:

Сценарий	Время выполнения	Затраты памяти
Пример (n=331)	0.00012 sec	4.12 Mb
Верхняя граница (n=10^7)	1.12051 sec	4.25 Mb

4 Вывод

В ходе лабораторной работы №0 были освоены:

1. Основы синтаксиса Python и настройки среды разработки.
2. Различные методы ввода-вывода данных (консольный ввод через `input()` и работа с текстовыми файлами через `open()`).
3. Принципы оптимизации алгоритмов: переход от рекурсивного метода вычисления чисел Фибоначчи к итеративному позволил сократить временную сложность с $O(2^n)$ до $O(n)$, что критично при больших значениях n.
4. Методика измерения времени работы программы с помощью `time.perf_counter()`.