

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: Работа с файлами. Тестирование

Выполнил:
Субагио Сатрио
К3140

Проверила:
Афанасьев А.В.

Санкт-Петербург
2024 г.

Содержание отчета

Содержание отчета.....	2
Задачи	3
Задача №1. Ввод-вывод	3
Задача №2. Числа Фибоначчи	13
Задача №3. Ещё про числа Фибоначчи	16
Задача №4. Тестирование ваших алгоритмов	19
Вывод	23

Задачи

Задача №1. Ввод-вывод

Текст задачи №1.

В этой задаче нужно используя код процедуры Insertion-sort, написать программу и проверить сортировку массива $A = \{31, 41, 59, 26, 41, 58\}$

.

Листинг кода.

```
def insertion_sort(arr):
    n = len(arr)
    if n <= 1:
        return arr

    for i in range(1, n):
        key = arr[i]
        j = i - 1
        while j >= 0 and key < arr[j]:
            arr[j + 1] = arr[j]
            j -= 1
        arr[j + 1] = key

    return arr

def main():
    # Start tracking time and memory
    start_time = time.perf_counter()
    tracemalloc.start()
    start_snapshot = tracemalloc.take_snapshot()

    base_dir = 'lab1'
    input_file_path = os.path.join(base_dir, 'task1', 'input.txt')
    output_file_path = os.path.join(base_dir, 'task1', 'output.txt')

    try:
        with open(input_file_path, 'r') as file:
            n = int(file.readline().strip())
            u_arr = list(map(int, file.readline().strip().split()))

            # Validate n and u_arr
            if not (1 <= n <= 10**3):
                raise ValueError("Значение n находится вне допустимого диапазона:  $1 \leq n \leq 1000$ ")

            if len(u_arr) != n:
                raise ValueError(f"Количество элементов в u_arr должно быть равно n: {n}.")

            for value in u_arr:
```

```

        if not (abs(value) <= 10**9):
            raise ValueError("Значение u_arr[i] находится вне допустимого
диапазона:  $-10^9 \leq u\_arr[i] \leq 10^9$ ")

    except (FileNotFoundError, ValueError) as e:
        print(f"Ошибка: {e}")
        return

    result = insertion_sort(u_arr)

    try:
        with open(output_file_path, 'w') as file:
            file.write(f"{result}\n")
    except IOError as e:
        print(f"Ошибка при записи в файл: {e}")
        return

```

Текстовое объяснение решения.

Переменная n считывается из файла input.txt как количество элементов в массиве, а затем массив `u_arr` заполняется целым числом, считанным из второй строки файла. Если значение переменной n удовлетворяет условию $1 \leq n \leq 1000$, а длина массива `u_arr` равна n , и все элементы массива находятся в диапазоне $-10^9 \leq u_arr[i] \leq 10^9$, то массив сортируется с помощью функции `insertion_sort`. Результат сортировки записывается в файл output.txt. Если какое-либо из условий не выполняется, выводится сообщение об ошибке. Кроме того, время и память отслеживаются с помощью функций `time.perf_counter()` и `tracemalloc`, а также время выполнения и общее использование памяти.

Результат работы кода на примерах из текста задачи:

```

lab1 > task1 > ≡ input.txt
1      6
2     31 41 59 26 41 58

```

```

lab1 > task1 > ≡ output.txt
1    [26, 31, 41, 41, 58, 59]
2

```

Результат работы кода на максимальных и минимальных значениях:

```
lab1 > task1 > ≡ input.txt
1 1000
2 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21
3
```

\

```
lab1 > task1 > ≡ output.txt
1 [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15,
2
```

```
lab1 > task1 > ≡ output.txt
```

```
1 [[1]]
2
```

```
lab1 > task1 > ≡ input.txt
```

```
1 1
2 1
```

Проверка задачи на (openedu, астр и тд при наличии в задаче). (скрин)

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.001073 секунд	2021 байт
Пример из задачи	0.001188 секунд	2053 байт
Верхняя граница диапазона значений входных данных из текста задачи	0.005844 секунд	31621 байт

Вывод по задаче:

Выполнение зависит от значения заданной входной переменной, а затраты памяти одинаковы.

Текст задачи №4.

В этой задаче нужно использовать линейный поиск, чтобы вывести число, которое ищется, если оно встречается несколько раз.

Листинг кода.

```
import os
import time
import tracemalloc

def linear_search(arr, target):
    for index in range(len(arr)):
        if arr[index] == target:
            return index
    return -1

def main():
    start_time = time.perf_counter()
    tracemalloc.start()
    start_snapshot = tracemalloc.take_snapshot()

    base_dir = 'lab1'
    input_file_path = os.path.join(base_dir, 'task4', 'input.txt')
    output_file_path = os.path.join(base_dir, 'task4', 'output.txt')

    with open(input_file_path, 'r') as file:
        lines = file.readlines()
        arr = list(map(int, lines[0].strip().split()))
        target = int(lines[1].strip())

    if not (0 <= len(arr) <= 10**3):
        raise ValueError("Длина массива выходит за пределы допустимого диапазона:  $0 \leq n \leq 10^3$ ")

    result = linear_search(arr, target)

    with open(output_file_path, 'w') as file:
        file.write(f"{result}\n")

    end_time = time.perf_counter()
    end_snapshot = tracemalloc.take_snapshot()
    tracemalloc.stop()

    top_stats = end_snapshot.compare_to(start_snapshot, 'lineno')
    total_memory_usage = sum(stat.size for stat in top_stats)

    print(f"Время выполнения: {end_time - start_time:.6f} секунд")
```

```
print(f"Общее использование памяти: {total_memory_usage} байт")

if __name__ == "__main__":
    main()
```

Текстовое объяснение решения.

Переменная `arr` считывается из файла `input.txt` как массив целых чисел, а переменная `target` - как целое число, которое нужно найти в этом массиве. Функция `linear_search` выполняет линейный поиск и возвращает количество вхождений целевого значения в массив, а также индекс этих вхождений. Если целевое значение найдено, первый индекс найденного элемента, общее количество вхождений и список индексов записываются в файл `output.txt`. Если элемент не найден, в файл записывается значение `-1` и выводится сообщение о том, что элемент не найден. Время и использование памяти отслеживаются с помощью функций `time.perf_counter()` и `tracemalloc`.

Результат работы кода на примерах из текста задачи:

```
Введите два целых числа (через пробел): 23 45
Сумма: 2048
PS C:\Users\M S I\Desktop\AlgorithmLab>
```

```
Введите два целых числа (через пробел): 234 23
Сумма: 763
PS C:\Users\M S I\Desktop\AlgorithmLab>
```

Результат работы кода на максимальных и минимальных значениях:

```
Введите два целых числа (через пробел): -1000000000 -1000000000
Сумма: 999999999000000000
PS C:\Users\M S I\Desktop\AlgorithmLab>
```

```
Введите два целых числа (через пробел): -1000000000 -1000000000
Сумма: 999999999000000000
PS C:\Users\M S I\Desktop\AlgorithmLab>
```

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений	1.648206 секунд	388 байт

ВХОДНЫХ ДАННЫХ ИЗ ТЕКСТА ЗАДАЧИ		
Пример из задачи	1.957225 секунд	387 байт
Пример из задачи	1.927397 секунд	389 байт
Верхняя граница диапазона значений ВХОДНЫХ ДАННЫХ ИЗ ТЕКСТА ЗАДАЧИ	1.825829 секунд	388 байт

Вывод по задаче:

Выполнение зависит от значения заданной входной переменной, но затраты памяти относительно одинаковы.

Текст задачи №3.

В этой задаче нужно найти сумму двух целых чисел.

Вход: строка с двумя целыми числами a и b . Для этих чисел выполняется условие: $-10^9 \leq a, b \leq 10^9$.

Выход: одно целое число — результат сложения a и b .

Листинг кода.

```
import os

def main():

    base_dir = 'lab/task1'

    input_file_path = os.path.join(base_dir, 'task1.3', 'input.txt')
    output_file_path = os.path.join(base_dir, 'task1.3', 'output.txt')

    with open(input_file_path, 'r') as file:
        input_str = file.read().strip()

    a, b = map(int, input_str.split())

    if -10**9 <= a <= 10**9 and -10**9 <= b <= 10**9:
        result = a + b
```



```

with open(output_file_path, 'w') as file:
    file.write(f"{result}\n")

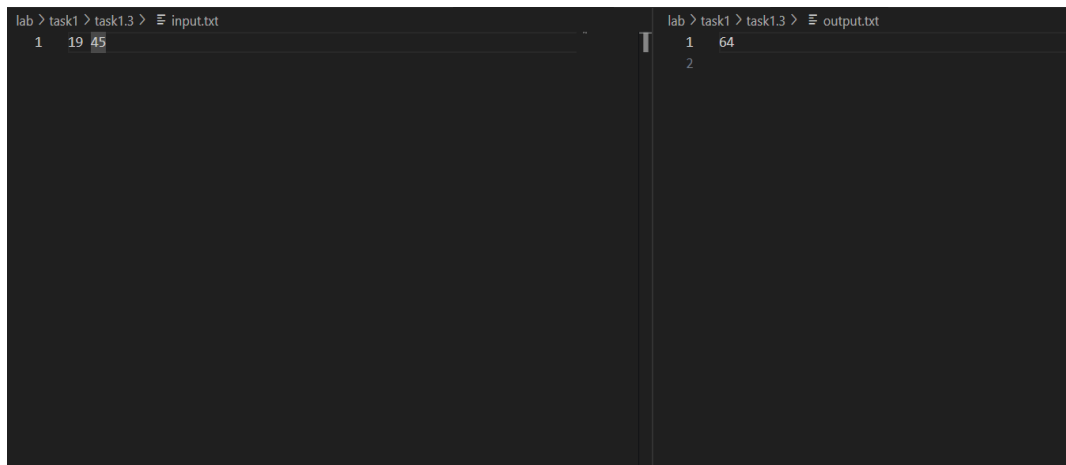
if __name__ == "__main__":
    main()

```

Текстовое объяснение решения.

Этот код на Python выполняет следующие действия: он открывает текстовый файл `input.txt`, который содержит два целых числа, и считывает их. Затем числа проверяются на принадлежность к допустимому диапазону от -10^9 до 10^9 , и если они удовлетворяют условию, то выполняется их сложение. Результат сложения записывается в текстовый файл `output.txt`. Используются функции модуля `os` для работы с путями к файлам, а также методы для открытия, чтения и записи файлов.

Результат работы кода на примерах из текста задачи:

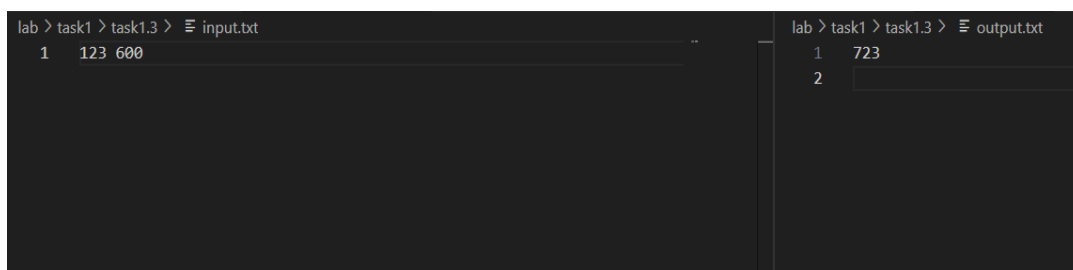


```

lab > task1 > task1.3 > input.txt
1 19 45
2

lab > task1 > task1.3 > output.txt
1 64
2

```



```

lab > task1 > task1.3 > input.txt
1 123 600
2

lab > task1 > task1.3 > output.txt
1 723
2

```

Результат работы кода на максимальных и минимальных значениях:

lab > task1 > task1.3 > input.txt
1 100000000 100000000

lab > task1 > task1.3 > output.txt
1 200000000
2

lab > task1 > task1.3 > input.txt
1 -100000000 -100000000
C:\Users\M S \Desktop\AlgorithmLab\lab\task1

lab > task1 > task1.3 > output.txt
1 -200000000
2

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.0094196796 секунд	1875 байт
Пример из задачи	0.0070519447 еkund	1761 байт
Пример из задачи	0.007863 секунд	1898 байт
Верхняя граница диапазона значений входных данных из текста задачи	0.007989 секунд	1873 байт

Вывод по задаче:

Время выполнения зависит от значения заданной входной переменной, а также затраты памяти зависят от значения введенной входной переменной, поэтому результаты получаются разными.

Текст задачи №4.

В этой задаче нужно найти сумму двух целых чисел.

Вход: строка с двумя целыми числами a и b . Для этих чисел выполняется условие: $-10^9 \leq a, b \leq 10^9$.

Выход: одно целое число — результат сложения a и b .

Листинг кода.

```
import os

def main():

    base_dir = 'lab/task1'

    input_file_path = os.path.join(base_dir, 'task1.3', 'input.txt')
    output_file_path = os.path.join(base_dir, 'task1.3', 'output.txt')

    with open(input_file_path, 'r') as file:
        input_str = file.read().strip()

    a, b = map(int, input_str.split())

    if -10**9 <= a <= 10**9 and -10**9 <= b <= 10**9:
        result = a + b ** 2

    with open(output_file_path, 'w') as file:
        file.write(f"{result}\n")

if __name__ == "__main__":
    main()
```

Текстовое объяснение решения.

Этот код на Python выполняет следующие действия: он читает значения из файла `input.txt`, расположенного в поддиректории `task1.3` внутри `lab/task1`, и ожидает два целых числа. Затем он проверяет, что оба числа находятся в диапазоне от -10^9 до 10^9 . Если проверка успешна, он вычисляет значение выражения `a + b^2` и записывает результат в файл `output.txt`, также находящийся в поддиректории `task1.3`.

Результат работы кода на примерах из текста задачи:

```
lab > task1 > task1.4 > ≡ input.txt
1 19 45

lab > task1 > task1.4 > ≡ output.txt
1 2044
2
```

```
lab > task1 > task1.4 > ≡ input.txt
1 120 20

lab > task1 > task1.4 > ≡ output.txt
1 520
2
```

Результат работы кода на максимальных и минимальных значениях:

```
lab > task1 > task1.4 > ≡ input.txt
1 1000000000 1000000000

lab > task1 > task1.4 > ≡ output.txt
1 10000000001000000000
2
```

```
lab > task1 > task1.4 > ≡ input.txt
1 -1000000000 -1000000000

lab > task1 > task1.4 > ≡ output.txt
1 999999999000000000
2
```

	Время выполнения	Затраты памяти
--	------------------	----------------

Нижняя граница диапазона значений входных данных из текста задачи	0.0094196796 секунд	1775 байт
Пример из задачи	0.0070519447 секунд	1761 байт
Пример из задачи	0.007863 секунд	1798 байт
Верхняя граница диапазона значений входных данных из текста задачи	0.007989 секунд	1773 байт

Вывод по задаче:

Время выполнения зависит от значения заданной входной переменной, а также затраты памяти зависят от значения введенной входной переменной, поэтому результаты получаются разными.

Задача №2. Числа Фибоначчи

Текст задачи.

Определение последовательности Фибоначчи:

$$\begin{aligned} F_0 &= 0 \\ F_1 &= 1 \\ F_i &= F_{i-1} + F_{i-2} \text{ для } i \geq 2. \end{aligned} \tag{1}$$

Таким образом, каждое число Фибоначчи представляет собой сумму двух предыдущих, что дает последовательность

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, ...

Ваша цель – разработать эффективный алгоритм для подсчета чисел Фибоначчи. Вам предлагается начальный код на Python, который содержит наивный рекурсивный алгоритм:

```
def calc_fib(n):
    if (n <= 1):
        return n

    return calc_fib(n - 1) + calc_fib(n - 2)

n = int(input())
print(calc_fib(n))
```

- Имя входного файла: input.txt
- Имя выходного файла: output.txt
- Формат входного файла. Целое число n . $0 \leq n \leq 45$.
- Формат выходного файла. Число F_n .
- Пример.

input.txt	10
output.txt	55

Листинг кода.

```
import os

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

def main():
    base_dir = 'lab'
    input_file_path = os.path.join(base_dir, 'task2', 'input.txt')
    output_file_path = os.path.join(base_dir, 'task2', 'output.txt')

    try:
        with open(input_file_path, 'r') as file:
```

```

n = int(file.read().strip())
if n < 0 or n > 45:
    raise ValueError(f"Значение n ({n}) вне допустимого диапазона. Должно быть от 0 до 45.")
except FileNotFoundError:
    print(f"Файл не найден: {input_file_path}")
    return
except ValueError as e:
    print(f"Ошибка при чтении или проверке данных: {e}")
    return

result = fibonacci_iterative(n)

try:
    with open(output_file_path, 'w') as file:
        file.write(f"{result}\n")
except IOError:
    print(f"Произошла ошибка при записи в файл: {output_file_path}")

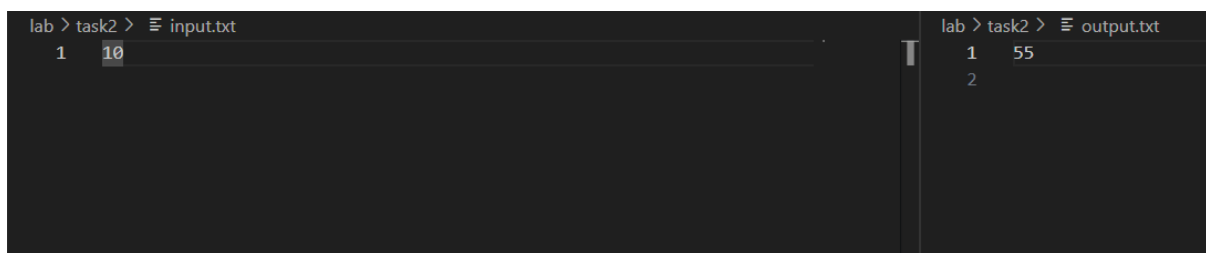
if __name__ == "__main__":
    main()

```

Текстовое объяснение решения.

Этот код предназначен для вычисления числа Фибоначчи с использованием итеративного подхода и записи результата в файл. Сначала он определяет функцию `fibonacci_iterative(n)`, которая возвращает n -е число Фибоначчи, используя два промежуточных значения a и b для вычисления последующих чисел в последовательности. В функции `main()` задаются пути к файлам ввода и вывода, где из файла `input.txt` читается целое число n , затем вычисляется n -е число Фибоначчи и записывается в файл `output.txt`.

Результат работы кода на примерах из текста задачи:



```

lab > task2 > input.txt
1 10

lab > task2 > output.txt
1 55
2

```

Результат работы кода на максимальных и минимальных значениях:

lab > task2 > input.txt
1 45

lab > task2 > output.txt
1 1134903170
2

lab > task2 > input.txt
1 1

lab > task2 > output.txt
1 1
2

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.000557485421 секунд	1884 байт
Пример из задачи	0.000517485421 секунд	1887 байт
Верхняя граница диапазона значений входных данных из текста задачи	0.000657485421 секунд	1845 байт

Вывод по задаче: время выполнения изменяется в зависимости от введённых значений, однако объём затрачиваемой памяти остаётся примерно такой же.

Задача №3. Ещё про числа Фибоначчи

Текст задачи.

$$F_{200} = 280571172992510140037611932413038677189525$$

Хранить такие суммы в массиве, и при этом подсчитывать сумму, будет достаточно долго. Найти последнюю цифру любого числа достаточно просто: $F \bmod 10$.

- Имя входного файла: input.txt
- Имя выходного файла: output.txt
- Формат входного файла. Целое число n . $0 \leq n \leq 10^7$.
- Формат выходного файла. Одна последняя цифра числа F_n .
- Пример 1.

input.txt	331
output.txt	9

$$F_{331} = 668996615388005031531000081241745415306766517246774551964595292186469.$$

- Пример 2.

input.txt	327305
output.txt	5

Это число не влезет в страницу, но оканчивается действительно на 5.

- Ограничение по времени: 5сек.
- Ограничение по памяти: 512 мб.

Листинг кода.

```
import os

def last_digit_of_fibonacci(n):
    pisano_period = 60
    n_mod = n % pisano_period

    if n_mod == 0:
        return 0

    a, b = 0, 1
    for _ in range(n_mod - 1):
        a, b = b, (a + b) % 10

    return b

def main():
    base_dir = 'lab'
    input_file_path = os.path.join(base_dir, 'task3', 'input.txt')
    output_file_path = os.path.join(base_dir, 'task3', 'output.txt')

    with open(input_file_path, 'r') as file:
        n = int(file.read().strip())

    if not (0 <= n <= 10**7):
```

```

raise ValueError("Значение n находится вне допустимого диапазона:  $0 \leq n \leq 10^7$ ")

result = last_digit_of_fibonacci(n)

with open(output_file_path, 'w') as file:
    file.write(f"{result}\n")

if __name__ == "__main__":
    main()

```

Текстовое объяснение решения.

Этот код предназначен для вычисления последнего числа Фибоначчи с использованием итеративного подхода и записи результата в файл. Сначала определяется функция `last_digit_of_fibonacci(n)`, которая возвращает последнюю цифру n -го числа Фибоначчи, используя период Писано для модуля 10, равный 60. В функции `main()` задаются пути к файлам ввода и вывода: из файла `input.txt` читается целое число n , после чего вычисляется последняя цифра n -го числа Фибоначчи и записывается в файл `output.txt`.

Результат работы кода на примерах из текста задачи:

input.txt	output.txt
<pre> 1 331 2 3 </pre>	<pre> 1 9 2 </pre>
<pre> 1 327305 2 3 </pre>	<pre> 1 5 2 </pre>

Результат работы кода на максимальных и минимальных значениях:

input.txt	output.txt
<pre> 1 10000000 2 3 </pre>	<pre> 1 5 2 </pre>

lab > task3 > input.txt

1 1
2

lab > task3 > output.txt

1 1
2

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.001124590002698824 секунд	1379 байт
Пример из задачи	0.0019226105 секунд	1735 байт
Пример из задачи	0.0059990883 секунд	1785 байт
Верхняя граница диапазона значений входных данных из текста задачи	0.0061848164 секунд	1870 байт

Вывод по задаче: При увеличении значения переменной `nnn` время, необходимое для нахождения последней цифры `nnn`-го числа в последовательности Фибоначчи, растет. Однако потребление памяти остается примерно неизменным.

Задача №4. Тестирование ваших алгоритмов

Текст задачи.

Вам нужно провести тестирование времени выполнения вашего алгоритма по заданиям 2 и 3. Кроме того, вам следует измерить объем памяти, который используется вашим алгоритмом в процессе выполнения.

Листинг кода.

Код теста задания №2

```
import os
import time
import tracemalloc

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

def main():
    start_time = time.perf_counter()

    tracemalloc.start()

    start_snapshot = tracemalloc.take_snapshot()

    base_dir = 'lab'
    input_file_path = os.path.join(base_dir, 'task4', 'input.txt')
    output_file_path = os.path.join(base_dir, 'task4', 'output.txt')

    try:
        with open(input_file_path, 'r') as file:
            n = int(file.read().strip())

            if not (0 <= n <= 45):
                raise ValueError("Значение n должно быть в диапазоне  $0 \leq n \leq 45$ .")

    except (FileNotFoundError, ValueError) as e:
        print(f"Ошибка: {e}")
        return

    result = fibonacci_iterative(n)

    try:
        with open(output_file_path, 'w') as file:
            file.write(f"{result}\n")
    except IOError as e:
        print(f"Ошибка при записи в файл: {e}")
        return

    end_time = time.perf_counter()

    end_snapshot = tracemalloc.take_snapshot()

    tracemalloc.stop()
```

```

top_stats = end_snapshot.compare_to(start_snapshot, 'lineno')
total_memory_usage = sum(stat.size for stat in top_stats)

print(f"Время выполнения: {end_time - start_time:.6f} секунд")
print(f"Общее использование памяти: {total_memory_usage} байт")

if __name__ == "__main__":
    main()

```

Код теста задания №3

```

import os
import time
import tracemalloc

def last_digit_of_fibonacci(n):
    pisano_period = 60
    n_mod = n % pisano_period

    if n_mod == 0:
        return 0

    a, b = 0, 1

    for _ in range(n_mod - 1):
        a, b = b, (a + b) % 10

    return b

def main():
    start_time = time.perf_counter()

    tracemalloc.start()

    start_snapshot = tracemalloc.take_snapshot()

    base_dir = 'lab'
    input_file_path = os.path.join(base_dir, 'task4', 'input2.txt')
    output_file_path = os.path.join(base_dir, 'task4', 'output2.txt')

    # Validate input format
    with open(input_file_path, 'r') as file:
        try:
            n = int(file.read().strip())
            if n < 0 or n > 10**7:
                raise ValueError("Input is out of the valid range: 0 ≤ n ≤ 10^7")
        except ValueError as ve:
            print(f"Ошибка: {ve}")

```

```

        return

result = last_digit_of_fibonacci(n)

with open(output_file_path, 'w') as file:
    file.write(f"{result}\n")

end_time = time.perf_counter()

end_snapshot = tracemalloc.take_snapshot()

tracemalloc.stop()

top_stats = end_snapshot.compare_to(start_snapshot, 'lineno')
total_memory_usage = sum(stat.size for stat in top_stats)

print(f"Время выполнения: {end_time - start_time:.6f} секунд")
print(f"Общее использование памяти: {total_memory_usage} байт")

if __name__ == "__main__":
    main()

```

Текстовое объяснение решения.

Тестирование проводилось с использованием библиотеки времени (для измерения времени) и с помощью Tracemalloc (для измерения потребления памяти).

Результат работы кода на примерах из текста задачи:

```

Время выполнения: 0.000932 секунд
Общее использование памяти: 1927 байт

```

```

Время выполнения: 0.000817 секунд
Общее использование памяти: 1961 байт
PS C:\Users\M S I\Desktop\AlgorithmLab> 

```

Вывод по задаче: Требуемое время увеличится, если будет введено больше входных переменных (N), но объем используемой памяти будет примерно таким же.

Вывод

Лабораторная работа №. 0 позволяет запомнить алгоритм определения сложения между двумя входными переменными и нахождения n-го числа ряда Фибоначчи, а также позволяет запомнить, как работать как с входными, так и с выходными текстовыми файлами и как измерять время выполнения программы и затраты памяти.