

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

Отчет по лабораторной работе №1  
по курсу «Алгоритмы и структуры данных»  
Тема: Работа с файлами. Тестирование  
Вариант 16

Выполнил:  
Субагио Сатрио  
K3140

Проверила:  
Афанасьев А.В.

Санкт-Петербург  
2024 г.

## Содержание отчета

Содержание отчета.....	2
Задачи .....	3
<b>Задача №1. Сортировка вставкой .....</b>	<b>3</b>
<b>Задача №4. Линейный поиск .....</b>	<b>6</b>
<b>Задача №8. Секретарь Своп .....</b>	<b>8</b>
<b>Задача №2. Сортировка вставкой + .....</b>	<b>13</b>
<b>Задача №3. Сортировка вставкой по убыванию .....</b>	<b>15</b>
<b>Задача №9. Сложение двоичных чисел .....</b>	<b>19</b>
Вывод .....	22

## Задачи

### Задача №1. Сортировка вставкой

#### Текст задачи №1.

В этой задаче нужно используя код процедуры Insertion-sort, напишите программу и проверьте 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.

## Задача №4. Линейный поиск

### 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:

```

lab1 > task4 > ≡ input.txt
1  1 2 2 3 4 5 5 6 6 7 8 4
2  2

```

```

lab1 > task4 > ≡ output.txt
1  2
2  Количество вхождений: 2, Индексы: 2, 3
3

```

### Результат работы кода на максимальных и минимальных значениях:

```

lab1 > task4 > ≡ input.txt
1  1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000
2  1000
3

```

```
lab1 > task4 > ≡ output.txt
1 1
2 Количество вхождений: 1000, Индексы: 1, 2, 3, 4, 5, 6, 7, 8,
3
```

```
lab1 > task4 > ≡ input.txt
1
2 5
```

```
lab1 > task4 > ≡ output.txt
1 -1
2
```

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.001184 секунд	2091 байт
Пример из задачи	0.000742 секунд	2324 байт
Верхняя граница диапазона значений входных данных из текста задачи	0.011495 секунд	76630 байт

### Вывод по задаче:

Время выполнения зависит от значения заданной входной переменной, но затраты памяти сильно зависят от заданного входного значения, например, при максимальном значении, которое занимает в 3 раза больше памяти, чем остальные.

### Задача №8. Секретарь Своп

#### Текст задачи №8.

В задаче необходимо использовать алгоритм пузырьковой сортировки для сортировки массива и реверсирования отсортированного массива.



## Листинг кода.

```
import os
import time
import tracemalloc

def bubble_sort_with_swaps(arr):
    swaps = []
    n = len(arr)
    sorted = False

    while not sorted:
        sorted = True
        for i in range(n - 1):
            if arr[i] > arr[i + 1]:
                arr[i], arr[i + 1] = arr[i + 1], arr[i]
                swaps.append(f"Swap elements at indices {i + 1} and {i + 2}.")
                sorted = False

    return arr, swaps

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, 'task8', 'input.txt')
    output_file_path = os.path.join(base_dir, 'task8', 'output.txt')

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

    if not (3 <= n <= 5000):
        raise ValueError("Длина массива должна быть в пределах:  $3 \leq n \leq 5000$ ")

    result, swaps = bubble_sort_with_swaps(arr)

    with open(output_file_path, 'w') as file:
        for swap in swaps:
            file.write(swap + "\n")
        file.write("No more swaps needed.\n") # Final message

    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 `n` считывается из файла `input.txt` как целое число, обозначающее количество элементов в массиве, а переменная `arr` - как массив целых чисел. Функция `bubble_sort_with_swaps` сортирует массив с помощью метода `bubble` и возвращает отсортированный массив и список операций подкачки, выполненных в процессе сортировки. Каждая операция обмена записывается в формате «Поменять местами элементы по индексам `X` и `Y`». Главная функция считывает данные из файла, проверяет корректность значения `n` и вызывает функцию сортировки. Результат операции замены записывается в файл `output.txt`. Время и использование памяти отслеживаются с помощью функций `time.perf_counter()` и `tracemalloc`. В конце работы программы выводится время выполнения и общее количество использованной памяти.

### Результат работы кода на примерах из текста задачи:

```

lab1 > task8 > ≡ input.txt
1      7
2      3 1 4 2 2 8 9

lab1 > task8 > ≡ output.txt
1      Swap elements at indices 1 and 2.
2      Swap elements at indices 3 and 4.
3      Swap elements at indices 4 and 5.
4      Swap elements at indices 2 and 3.
5      Swap elements at indices 3 and 4.
6      No more swaps needed.
7

```

### Результат работы кода на максимальных и минимальных значениях:



```
lab1 > task8 > ≡ input.txt
```

```
1 3
2 5 3 2
```

```
lab1 > task8 > ≡ output.txt
```

```
1 Swap elements at indices 1 and 2.
2 Swap elements at indices 2 and 3.
3 Swap elements at indices 1 and 2.
4 No more swaps needed.
5
```

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.001409 секунд	2331 байт
Пример из задачи	0.001335 секунд	2527 байт
Верхняя граница диапазона значений входных данных из текста задачи	140.578845 секунд	1873 байт

### Вывод по задаче:

Время выполнения зависит от значения вводимой переменной и наибольшее максимальное значение составляет около 3 минут времени выполнения, а также стоимость памяти зависит от значения вводимой переменной, поэтому результаты получаются разными, в данном случае максимальное значение имеет наибольшую память.

## Задача №2. Сортировка вставкой +

### Текст задачи №2.

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
import time
import tracemalloc

def insertion_sort_with_indices(arr):
    n = len(arr)
    indices = list(range(n))
    sorted_arr = arr[:]

    for i in range(1, n):
        key = sorted_arr[i]
        key_index = indices[i]
        j = i - 1

        while j >= 0 and key < sorted_arr[j]:
            sorted_arr[j + 1] = sorted_arr[j]
            indices[j + 1] = indices[j]
            j -= 1

        sorted_arr[j + 1] = key
        indices[j + 1] = key_index

    return indices, sorted_arr

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, 'task2(prod)', 'input.txt')
    output_file_path = os.path.join(base_dir, 'task2(prod)', 'output.txt')

    file_exists = os.path.isfile(input_file_path)
    if not file_exists:
        print("Ошибка: Файл не найден.")
        return
```

```

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

if 1 <= n <= 10**3 and len(u_arr) == n:
    if all(abs(value) <= 10**9 for value in u_arr):
        result_indices, sorted_arr = insertion_sort_with_indices(u_arr)
        with open(output_file_path, 'w') as file:
            file.write(' '.join(map(str, [index + 1 for index in
result_indices]))) + '\n')
            file.write(' '.join(map(str, sorted_arr)) + '\n')
    else:
        print("Ошибка: Значения u_arr[i] находятся вне допустимого
диапазона:  $-10^9 \leq u\_arr[i] \leq 10^9$ .")
    else:
        print("Ошибка: Значение n должно быть в диапазоне  $1 \leq n \leq 1000$  и
количество элементов должно совпадать с 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.

Этот код представляет собой программу на Python, которая выполняет сортировку с использованием алгоритма **Insertion Sort**, отслеживая при этом новые индексы отсортированных элементов. Сначала программа импортирует необходимые модули для работы с файлами, измерения времени и отслеживания использования памяти. В функции `main` программа читает данные из файла `input.txt`, который находится в поддиректории `task2(prod)` внутри директории `lab1`. После считывания количества элементов и массива из файла код сразу же запускает функцию сортировки, не выполняя проверку корректности входных данных, предполагая, что предоставленные данные всегда верны. Результаты сортировки — новые индексы и отсортированный массив — затем записываются в файл `output.txt` в том же месте. Кроме того, программа измеряет время выполнения и использование памяти в процессе, которые затем выводятся в консоль. Упрощение кода за счет удаления проверки валидности входных

данных делает его более простым, но требует уверенности в том, что файл ввода всегда содержит корректные данные.

### Результат работы кода на примерах из текста задачи:

```
lab1 > task2(prod) > ≡ input.txt
1    10
2    1 8 4 2 3 7 5 6 9 0
```

```
lab1 > task2(prod) > ≡ output.txt
1    10 1 4 5 3 7 8 6 2 9
2    0 1 2 3 4 5 6 7 8 9
3
```

	Время выполнения	Затраты памяти
Пример из задачи	0.001713 секунд	2369 байт

### Вывод по задаче:

Сделайте вывод, что программа на Python эффективно применяет алгоритм Insertion Sort для сортировки элементов в массиве, отмечая при этом новые индексы элементов. Сохранив результаты сортировки и новые индексы в выходном файле и измерив время выполнения и использование памяти.

## Задача №3. Сортировка вставкой по убыванию

### Текст задачи.

Перепишите процедуру Insertion-sort для сортировки в невозрастающем порядке вместо неубывающего с использованием процедуры Swap.

Формат входного и выходного файла и ограничения - как в задаче 1.

*Подумайте, можно ли переписать алгоритм сортировки вставкой с использованием рекурсии?*

## Листинг кода.

```
import os
import time
import tracemalloc

def insertion_sort_recursive(arr, n):

    if n <= 1:
        return arr

    insertion_sort_recursive(arr, n - 1)

    key = arr[n - 1]
    j = n - 2

    while j >= 0 and arr[j] > key:
        arr[j + 1] = arr[j]
        j -= 1
    arr[j + 1] = key

    return arr

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, 'task3(prod)', 'input.txt')
    output_file_path = os.path.join(base_dir, 'task3(prod)', 'output.txt')

    if not os.path.isfile(input_file_path):
        print("Ошибка: Файл не найден.")
        return

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

    if not (1 <= n <= 1000):
        print("Ошибка: Значение n находится вне допустимого диапазона:  $1 \leq n \leq 1000$ ")
        return

    if len(arr) != n:
```



```

        print(f"Ошибка: Количество элементов в arr должно совпадать с n:
{n}.")
        return

    for i in range(len(arr)):
        if not (abs(arr[i]) <= 10**9):
            print("Ошибка: Значение arr[i] находится вне допустимого
диапазона:  $-10^9 \leq arr[i] \leq 10^9$ ")
            return

    result = insertion_sort_recursive(arr, n)

    with open(output_file_path, 'w') as file:
        file.write(" ".join(map(str, 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.

Этот код выполняет сортировку вставки с использованием рекурсии. Функция `insertion_sort_recursive` реализует логику сортировки, которая принимает массив `arr` и имеет длину `n`. Если `n` меньше или равно 1, массив считается отсортированным и возвращается. В противном случае функция рекурсивно вызывает саму себя для сортировки первых `n-1` элементов. После этого последний элемент помещается на нужное место в отсортированной части массива. Главная функция отвечает за управление вводом и выводом: она проверяет файл `input.txt`, считывает количество элементов и сам массив, после чего начинает сортировку. По окончании сортировки результат записывается в файл `output.txt`. В коде также используется модуль `tracemalloc` для отслеживания использования памяти и измерения времени выполнения алгоритма. При возникновении ошибок, таких как отсутствие файла или недопустимые данные, выводится соответствующее сообщение.

*Результат работы кода на примерах из текста задачи:*

```
lab1 > task3(prod) > ≡ input.txt
1      6
2      31 41 59 26 41 58
```

```
lab1 > task3(prod) > ≡ output.txt
1      59 58 41 41 31 26
2
```

*Результат работы кода на максимальных и минимальных значениях:*

```
lab1 > task3(prod) > ≡ input.txt
1      1000
2      475 289 146 63 264 290 138 340 11 358 254 103 317 93 430 455 363 416 87 166 230 284 346 492 240 449 170 320 236 235
3
```

```
lab1 > task3(prod) > ≡ output.txt
1      0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32
2
```

```
lab1 > task3(prod) > ≡ input.txt
1      1
2      1
```

```
lab1 > task3(prod) > ≡ output.txt
1      1
2
```

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.001744 секунд	2057 байт

Пример из задачи	0.001879 секунд	2089 байт
Верхняя граница диапазона значений входных данных из текста задачи	0.072866 секунд	3049 байт

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

## Задача №9. Сложение двоичных чисел

### Текст задачи.

Рассмотрим задачу сложения двух  $n$ -битовых двоичных целых чисел, хранящихся в  $n$ -элементных массивах  $A$  и  $B$ . Сумму этих двух чисел необходимо занести в двоичной форме в  $(n + 1)$ -элементный массив  $C$ . Напишите скрипт для сложения этих двух чисел.

- **Формат входного файла (input.txt).** В одной строке содержится два  $n$ -битовых двоичных числа, записанные через пробел ( $1 \leq n \leq 10^3$ )
- **Формат выходного файла (output.txt).** Одна строка - двоичное число, которое является суммой двух чисел из входного файла.
- Оцените асимптотическое время выполнения вашего алгоритма.

### Листинг кода.

```
import os
import time
import tracemalloc

def binary_addition(A, B):
    n = len(A)
    C = [0] * (n + 1)
    carry = 0

    for i in range(n - 1, -1, -1):
        total = A[i] + B[i] + carry
        C[i + 1] = total % 2
        carry = total // 2

    C[0] = carry
    return C
```

```

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, 'task9(prod)', 'input.txt')
    output_file_path = os.path.join(base_dir, 'task9(prod)', 'output.txt')

    with open(input_file_path, 'r') as file:
        line = file.readline().strip()
        A_str, B_str = line.split()

    n = len(A_str)

    if n < 1 or n > 1000:
        raise ValueError("Длина двоичного числа должна быть в диапазоне:  $1 \leq n \leq 1000$ ")

    A = list(map(int, A_str))
    B = list(map(int, B_str))

    C = binary_addition(A, B)

    with open(output_file_path, 'w') as file:
        file.write(''.join(map(str, C)) + '\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"Использование памяти: {total_memory_usage} байт")

if __name__ == "__main__":
    main()

```

### Текстовое объяснение решения.

Этот код реализует двоичное сложение двух двоичных чисел, представленных в виде строк. Функция `binary_addition` складывает два массива `A` и `B`, которые представляют двоичные числа. Функция создает массив `C` для хранения результата и переменную `carry` для учета переноса. Сложение выполняется от конца массива к началу, при этом каждый раз вычисляется соответствующее количество битов и возможный перенос. Результат

добавляется в массив C, который затем возвращается. Главная функция считывает данные из файла input.txt, содержащего два двоичных числа. Код проверяет длину чисел, чтобы убедиться, что они находятся в диапазоне от 1 до 1000. После выполнения сложения результат записывается в файл output.txt. Кроме того, программа измеряет время выполнения и использование памяти с помощью модуля tracemalloc и выводит эти данные по окончании выполнения.

**Результат работы кода на примерах из текста задачи:**

```
lab1 > task9(prod) > ≡ input.txt
1    1011 1101
```

```
lab1 > task9(prod) > ≡ output.txt
1    11000
2
```

	Время выполнения	Затраты памяти
Пример из задачи	0.001027 секунд	2413 байт

**Вывод по задаче:** Она заключается в том, что программа эффективно выполняет сложение двух двоичных чисел, заданных в виде строки. Функция `binary_addition` реализует алгоритм двоичного сложения, который учитывает биты обоих чисел, включая обработку возможного переноса. Программа также включает проверку того, что длина двоичных чисел находится в допустимом диапазоне, а также протоколирует время выполнения и использование памяти.

## **Вывод**

Лабораторная работа №. 1 позволяет вспомнить алгоритмы сортировки insertion sort, bubble sort, а также алгоритмы swap и линейного поиска и непосредственно применить их, не забывая о времени выполнения и используемой памяти.