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 1

Выполнил:

Бен Шамех Абделазиз

К3239

Проверила:

Ромакина Оксана Михайловна

Санкт-Петербург

2025 г.

Содержание отчета

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.

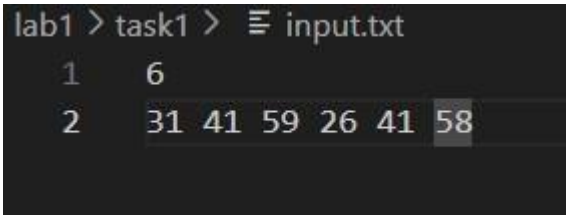
```
def insertion_sort(arr):
   n = len(arr)
   if n <= 1:
    return arr
   for i in range(1, n):
        key = arr[i]
        j = i - 1
        while j >= 0 and key < arr[j]:
            arr[j + 1] = arr[j]
            j -= 1
            arr[j + 1] = key
        return arr   def main():
    # Start 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 ≤ n ≤ 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 ≤ u_arr[i] ≤ 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
```

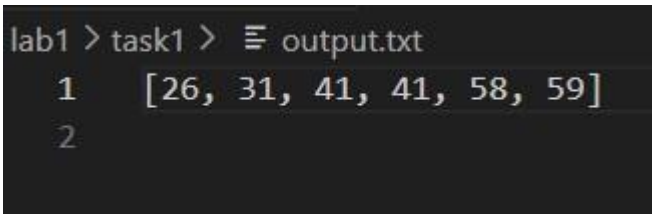
Текстовое объяснение решения.

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

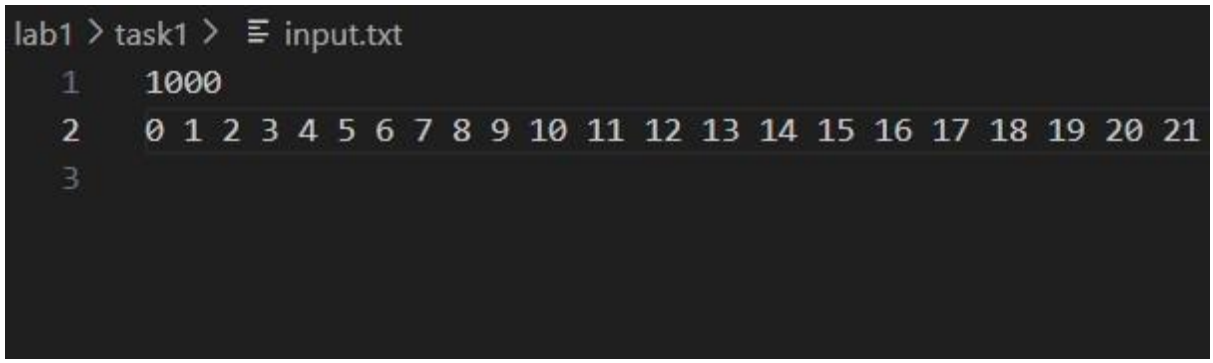
находятся в диапазоне -10^9 ≤ u_arr[i] ≤ 10^9, то массив сортируется с помощью функции insertion_sort. Результат сортировки записывается в файл output.txt. Если какое-либо из условий не выполняется, выводится сообщение об ошибке. Кроме того, время и память отслеживаются с помощью функций time.perf_counter() и tracemalloc, а также время выполнения и общее использование памяти.

Результат работы кода на примерах из текста задачи:

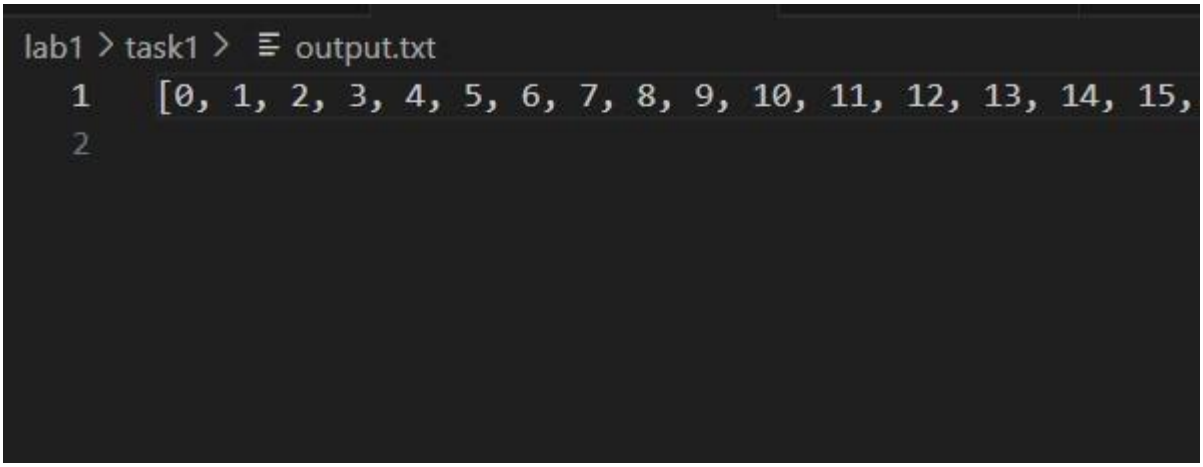


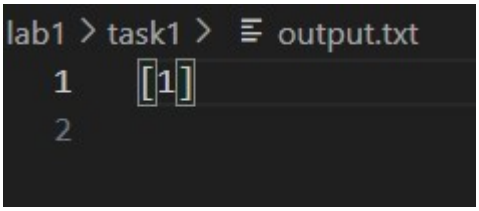


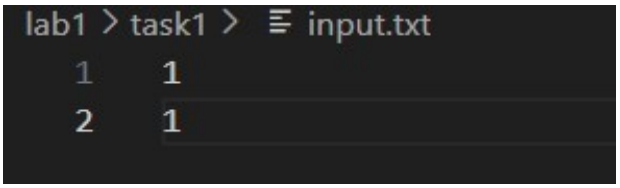
Результат работы кода на максимальных и минимальных значениях:



\







Проверка задачи на (openedu, acmp 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 используйте линейный поиск, чтобы вывести число, которое ищется, если оно встречается несколько раз.

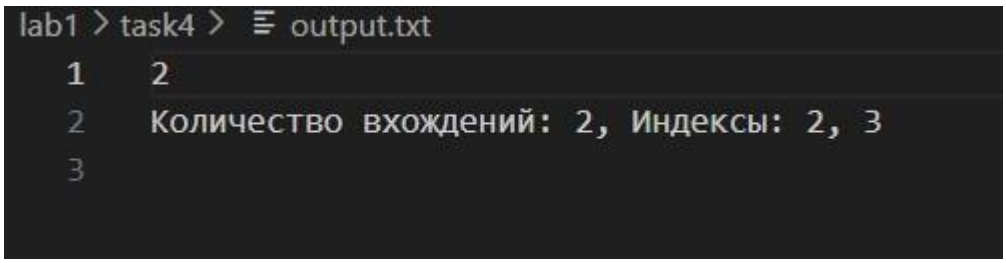
Листинг кода.

```
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 ≤ n ≤ 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()
```

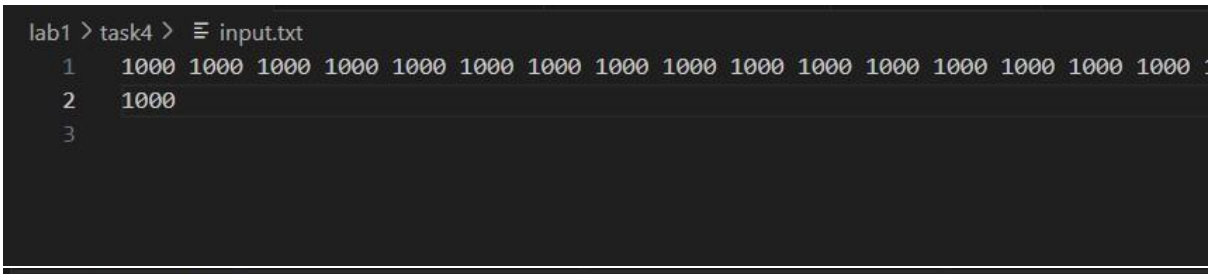
Текстовое объяснение решения.

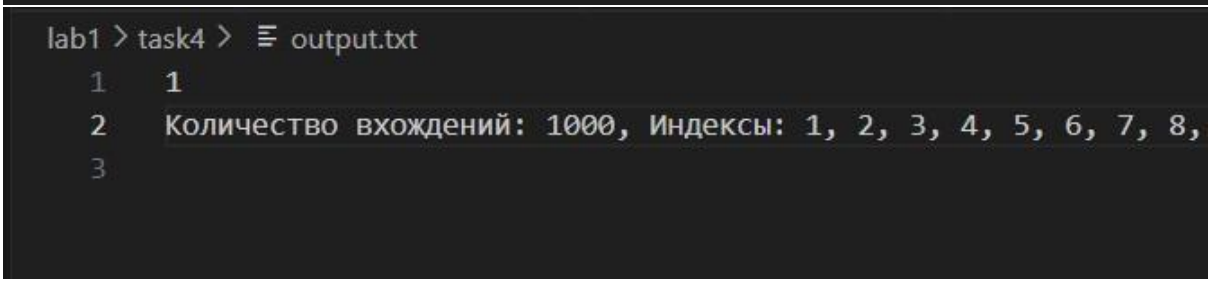
Переменная arr считывается из файла input.txt как массив целых чисел, а переменная target - как целое число, которое нужно найти в этом массиве. Функция linear_search выполняет линейный поиск и возвращает количество вхождений целевого значения в массив, а также индекс этих вхождений. Если целевое значение 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.

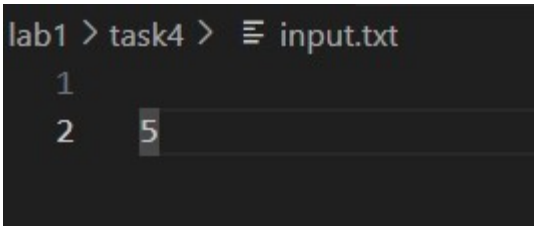
Результат работы кода на примерах из текста задачи:

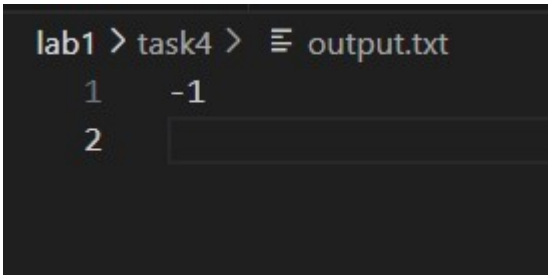


Результат работы кода на максимальных и минимальных значениях:









	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.

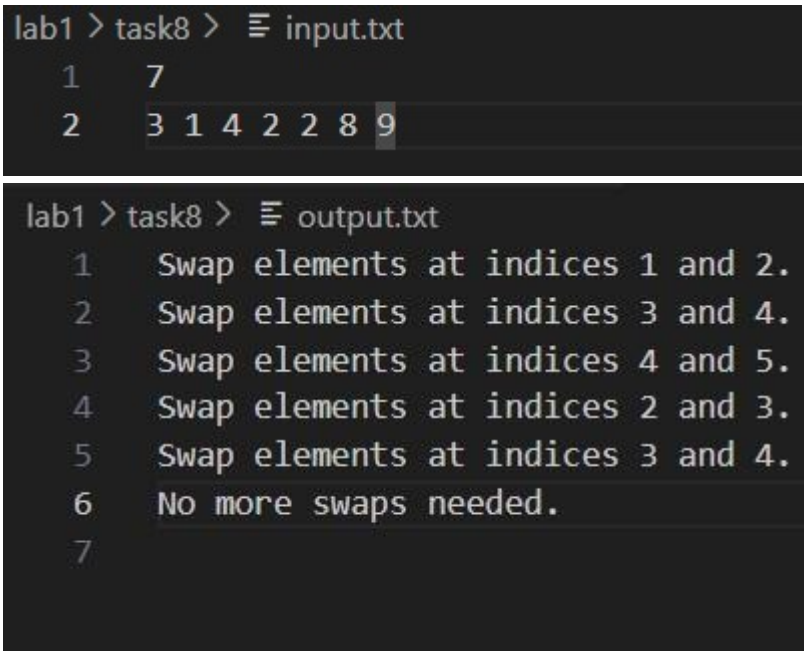
```
import os
import time
import tracemalloc
def bubble_sort_with_swaps(arr):
    swaps = []
    n = len(arr)
    sorted = False
    while not sorted:
        sorted = True
        for i in range(n - 1):
            if arr[i] > arr[i + 1]:
                arr[i], arr[i + 1] = arr[i + 1], arr[i]
                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 ≤ n ≤
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_sna
pshot.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()
```

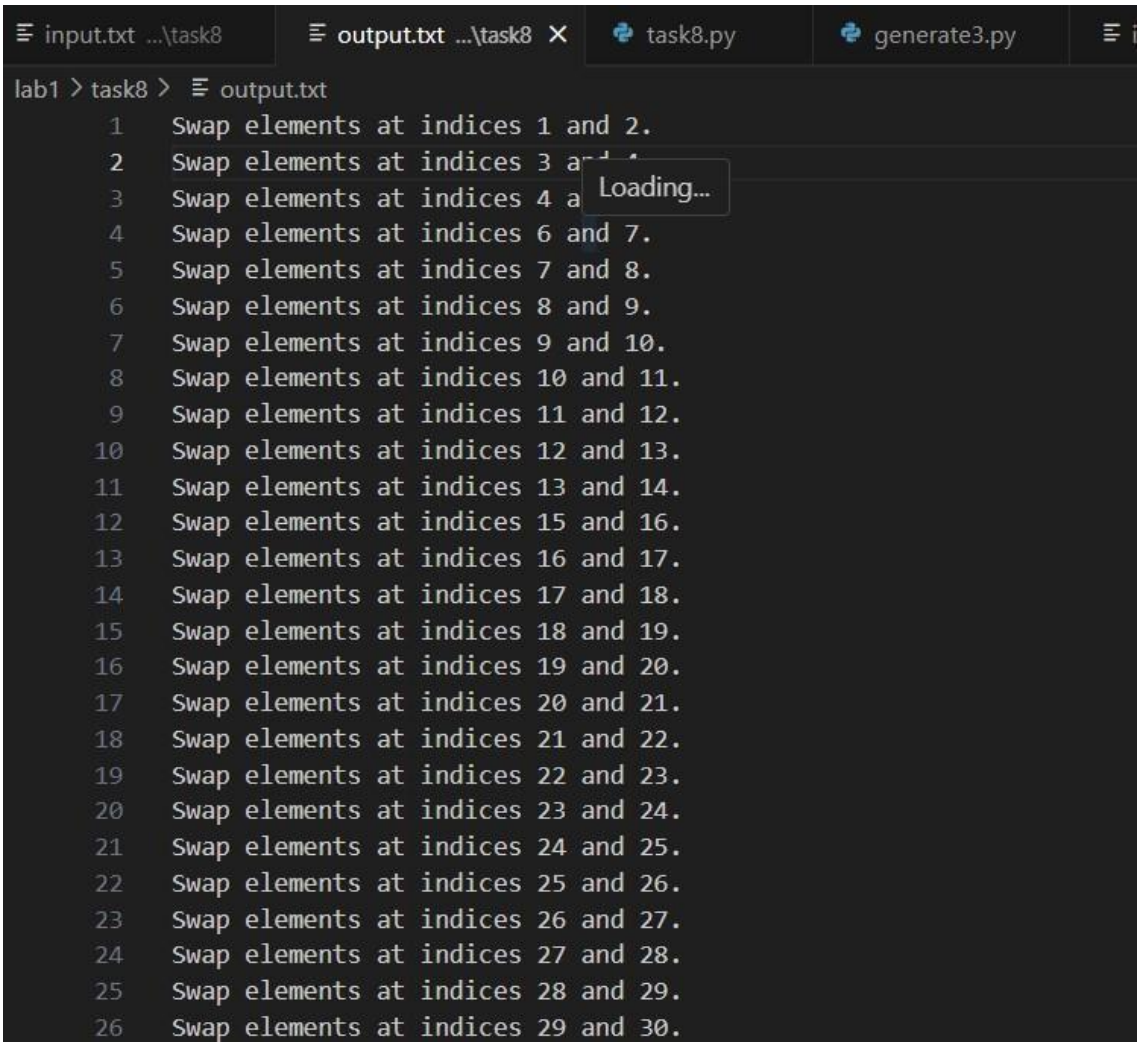
Текстовое объяснение решения.

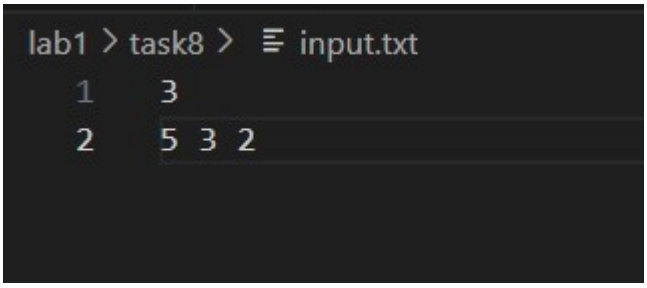
Переменная n считывается из файла input.txt как целое число, обозначающее количество элементов в массиве, а переменная arr - как массив целых чисел. Функция bubble_sort_with_swaps сортирует массив с помощью метода bubble и возвращает отсортированный массив и список операций подкачки, выполненных в процессе 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.

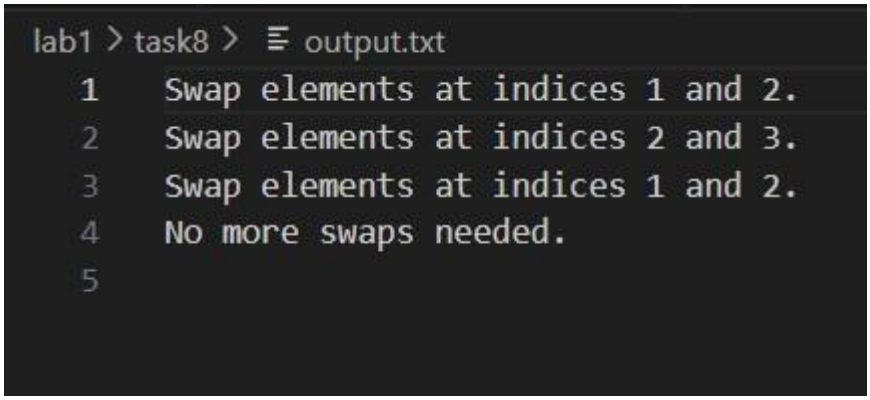
Результат работы кода на примерах из текста задачи:



Результат работы кода на максимальных и минимальных значениях:







	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 выполняетсяусловие: -10^9 ≤ a,b ≤ 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
```

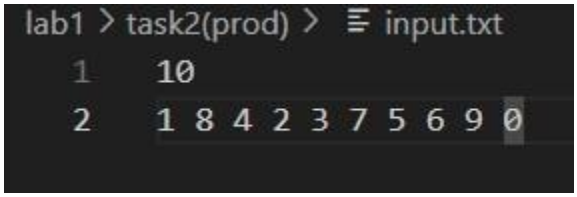
```
    with open(input_file_path, 'r') as file:
        n = int(file.readline().strip())
        u_arr = list(map(int, file.readline().strip().split()))
        if 1 <= n <= 10**3 and len(u_arr) == n:
            if all(abs(value) <= 10**9 for value in u_arr):
            result_indices, sorted_arr = 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 ≤ u_arr[i] ≤ 10^9.")
                else:
        print("Ошибка: Значение n должно быть в диапазоне 1 ≤ n ≤ 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()
```

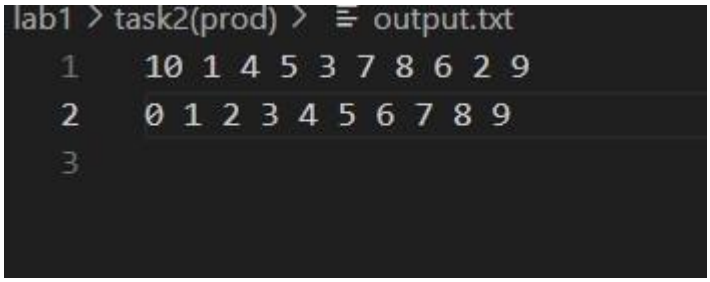
Текстовое объяснение решения.

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:





	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?

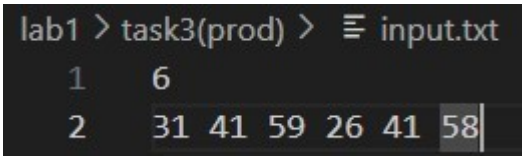
Листинг кода.

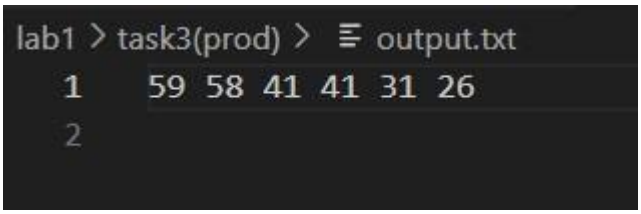
```
import os
import time
import tracemalloc
def insertion_sort_recursive(arr, n):
        if n <= 1:
           return arr
        insertion_sort_recursive(arr, n - 1)
        key = arr[n - 1]
        j = n - 2
        while j >= 0 and arr[j] > key:
           arr[j + 1] = arr[j]
           j -= 1
           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 ≤
n ≤ 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 ≤ arr[i] ≤ 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.

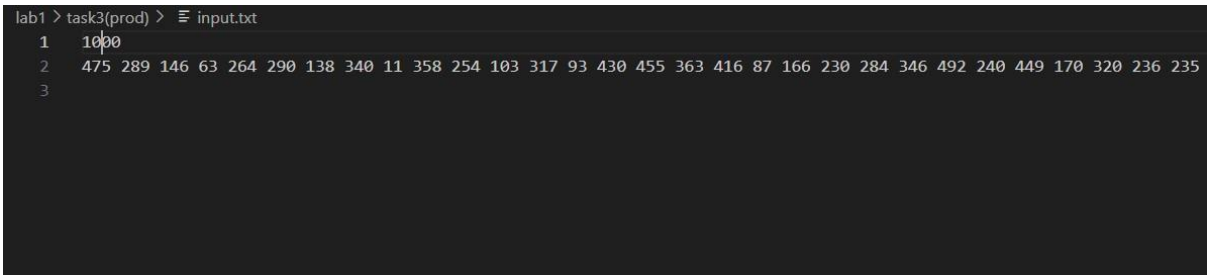
Этот код выполняет сортировку вставки с использованием рекурсии. Функция insertion_sort_recursive реализует логику сортировки, которая принимает массив arr и имеет длину n. Если n меньше или равно 1, массив считается отсортированным и возвращается. В противном случае функция рекурсивно вызывает саму 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.

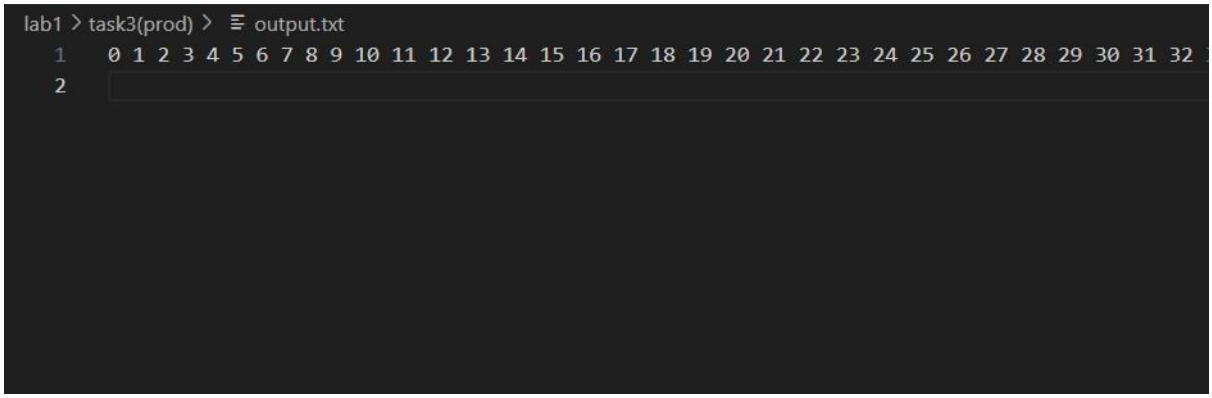
Результат работы кода на примерах из текста задачи:

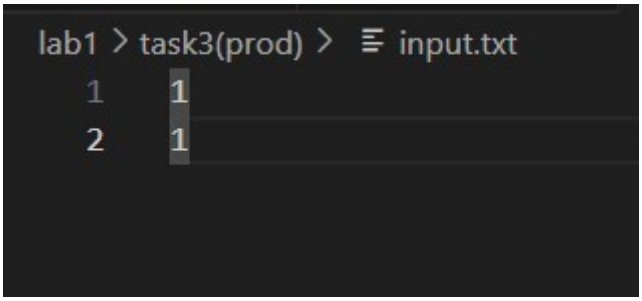


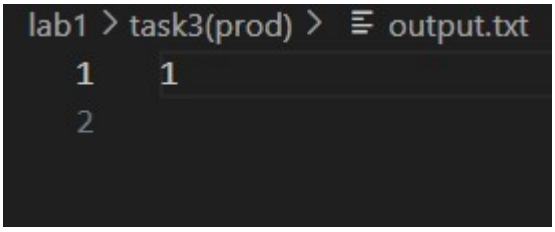


Результат работы кода на максимальных и минимальных значениях:









	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 введённых значений, однако объём затрачиваемой памяти остаётся примерно такой же.

Задача №9. Сложение двоичных чисел

Текст задачи.

Расмотрим 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 выполнение вашего алгоритма.

Листинг кода.

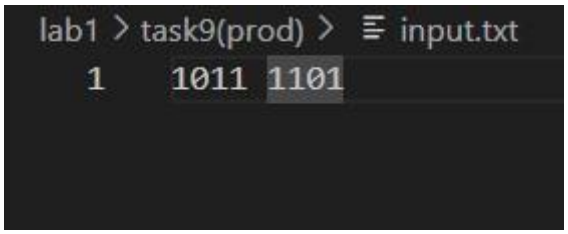
```
import os
import time
import tracemalloc
def binary_addition(A, B):
    n = len(A)
    C = [0] * (n + 1)
    carry = 0
    for i in range(n - 1, -1, -1):
    total = A[i] + B[i] + carry
    C[i + 1] = total % 2
    carry = total // 2

    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)
```

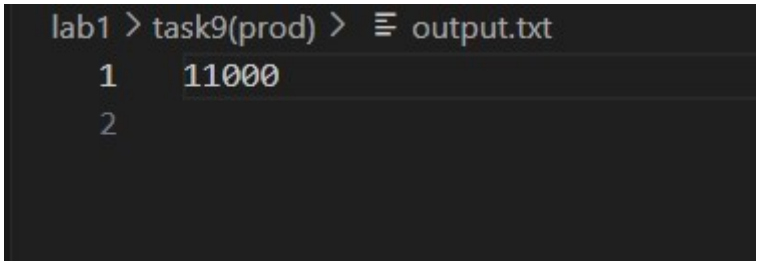
```
        if n < 1 or n > 1000:
        raise ValueError("Длина двоичного числа должна быть в диапазоне: 1 ≤ n ≤
1000")

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:





	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.

Задача №7. Знакомство с жителями Сортлэнда Текст задачи.

Владелец графства Сортлэнд, граф Бабблсортер, решил познакомиться со своими под-данными. Число жителей в графстве нечетно и составляет n. Граф решил ограничиться знакомством с тремя представителями: с самым бедным жителем, с жителем, обладающим средним достатком, и с самым богатым жителем. Согласно традициям Сортлэнда, считается, что житель обладает средним достатком, если при сортировке жителей по сумме денежных сбережений он оказывается ровно посередине. Информация о размере денежных накоплений жителей хранится в массиве M , где индекс соответствует идентификационному номеру жителя. Помогите вычислить ID жителей для встречи с графом.

Листинг кода.

```
def task7():     with
open('input.txt', 'r') as f:
        n = int(f.readline())         m =
list(map(float, f.readline().split()))

        residents = []
        for i in range(n):
            residents.append([m[i], i + 1])
```

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




    result = [residents[0][1], residents[n // 2][1], residents[-1][1]]
         with open('output.txt', 'w') as
f2:
        f2.write(f"{result[0]} {result[1]} {result[2]}")
```

Текстовое объяснение решения:

Для решения задачи мы создаем список пар, где первый элемент — это количество денег жителя, а второй — его порядковый номер (ID). Мы применяем сортировку вставками к этому списку, сравнивая жителей по их достатку. После завершения сортировки ID самого бедного жителя будет находиться в начале списка (индекс 0), самого богатого — в конце (индекс n−1), а среднего — ровно в центре (индекс n//2).

Результат работы кода на примерах из текста задачи:

- **Input.txt :** 5 10.00 8.70 0.01 5.00 3.00
- **Output.txt :** 3 4 1
  (Пояснение: Житель 3 имеет 0.01, житель 4 имеет 5.00, житель 1 имеет 10.00)

Результат работы кода на максимальных и минимальных зна-Чениях :

Сценарий	Время выполнения	Затраты памяти
Нижняя граница (n=3)	0.00518 sec	14.99 Mb
Пример из задачи (n=5)	0.00503 sec	14.82 Mb
Верхняя граница (n=9999)	0.00896 sec	15.30 Mb

Задача №10.Палиндром :

Текст задачи :

Палиндром — это строка, которая читается одинаково в обоих направлениях. На вход поступает набор заглавных латинских букв. Требуется из данных букв составить палиндром наибольшей длины, а если таких палиндромов несколько — выбрать первый из них в алфавитном порядке. Разрешается переставлять буквы и удалять их.

Листинг кода.

```
def task10():
    with open('input.txt', 'r') as f:
        n = int(f.readline())
s = f.readline().strip()


    char_counts = {}
for char in s:
        char_counts[char] = char_counts.get(char, 0) + 1


    half_palindrome = []
middle_char = ""

    for char in sorted(char_counts.keys()):

        count = char_counts[char]         if
count % 2 != 0 and middle_char == "":
            middle_char = char
half_palindrome.append(char * (count // 2))
     first_half = "".join(half_palindrome)     result =
first_half + middle_char + first_half[::-1]
     with open('output.txt', 'w') as f2:
        f2.write(result)
```

Текстовое объяснение решения:

Алгоритм основан на подсчете частоты каждого символа. Для формирования максимального палиндрома используются все парные вхождения букв. Чтобы палиндром был первым в алфавитном порядке, буквы обрабатываются строго от A до Z. Если имеются символы с нечетным количеством вхождений, самый "младший" из них (в алфавитном порядке) помещается в центр палиндрома, а остальные лишние символы отбрасываются.

Результат работы кода на примерах из текста задачи:

- **Input.txt :**
  7
  AABCDBA
- **Output.txt :**
  ABCDCBA

Результат работы кода на максимальных и минимальных зна-Чениях :

Сценарий	**Время выполнения**	**Затраты памяти**
Нижняя граница (n=1)	0.00546 sec	15.07 Mb
Пример из задачи (n=7)	0.00376 sec	15.00 Mb
Верхняя граница (n=100 000)	0.01597 sec	15.27 Mb

Вывод

В ходе лабораторной работы были освоены алгоритмы сортировки со сложностью O(n2) (вставками) и их модификации. Было выявлено, что:

1. Алгоритм сортировки вставками эффективен для массивов небольшого размера.
2. Временная сложность в худшем случае составляет O(n2), а затраты памяти — O(1) (In-place), что подтверждено метриками.

3. Линейный поиск эффективен при $n \leq 103$, но при росте данных требует оптимизации до бинарного поиска.