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

Отчет по лабораторной работе №2

по курсу «Алгоритмы и структуры данных»

Тема: Merge Sort

Выполнил:
Лазарев Марк Олегович
К3241

Санкт-Петербург
2025 г.

Задачи по варианту

Задача №1. Сортировка слиянием

Задача №2. Бинарный поиск

Задача №3. Метод Штрассена для умножения матриц

Задачи по варианту

1 задача. Сортировка слиянием

1. Используя *псевдокод* процедур `Merge` и `Merge-sort` из презентации к Лекции 2 (страницы 6-7), напишите программу сортировки слиянием на Python и проверьте сортировку, создав несколько рандомных массивов, подходящих под параметры:

   - **Формат входного файла (input.txt).** В первой строке входного файла содержится число n ($1 \leq n \leq 2 \cdot 10^4$) — число элементов в массиве. Во второй строке находятся n различных целых чисел, по модулю не превосходящих 10^9.
   - **Формат выходного файла (output.txt).** Одна строка выходного файла с отсортированным массивом. Между любыми двумя числами должен стоять ровно один пробел.
   - Ограничение по времени. 2сек.
   - Ограничение по памяти. 256 мб.

2. Для проверки можно выбрать наихудший случай, когда сортируется массив размера 1000, $10^4, 10^5$ чисел порядка 10^9, отсортированных в обратном порядке; наилучший, когда массив уже отсортирован, и средний. Сравните, например, с сортировкой вставкой на этих же данных.

3. Перепишите процедуру `Merge` так, чтобы в ней не использовались сигнальные значения. Сигналом к остановке должен служить тот факт, что все элементы массива L или R скопированы обратно в массив A, после чего в этот массив копируются элементы, оставшиеся в непустом массиве.

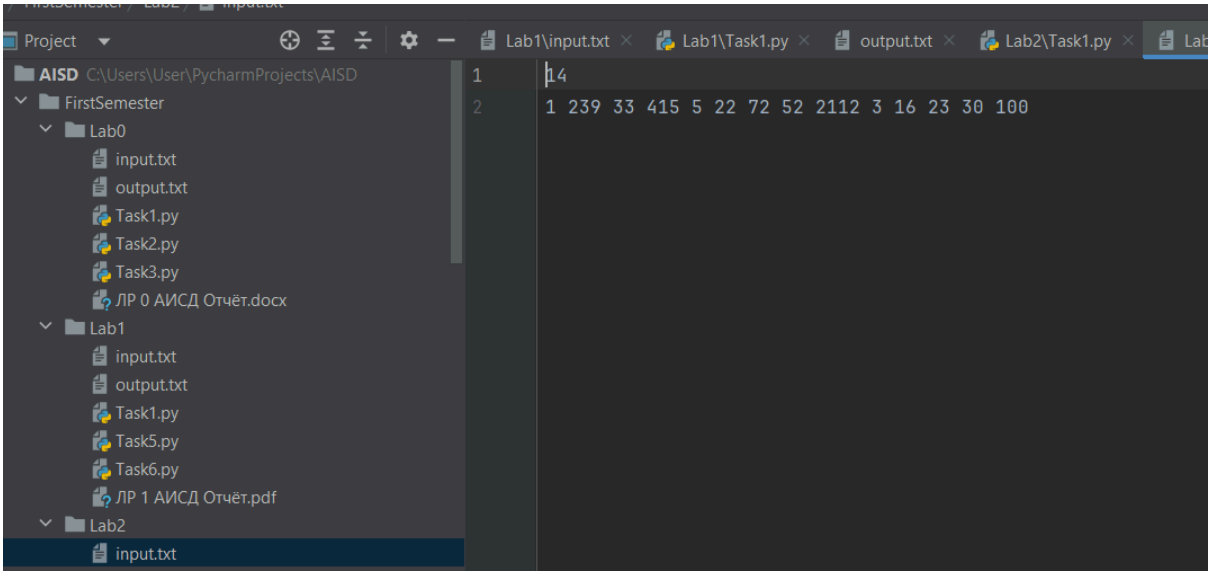
или перепишите процедуру `Merge` (и, соответственно, `Merge-sort`) так, чтобы в ней не использовались значения границ и середины - p, r и q.

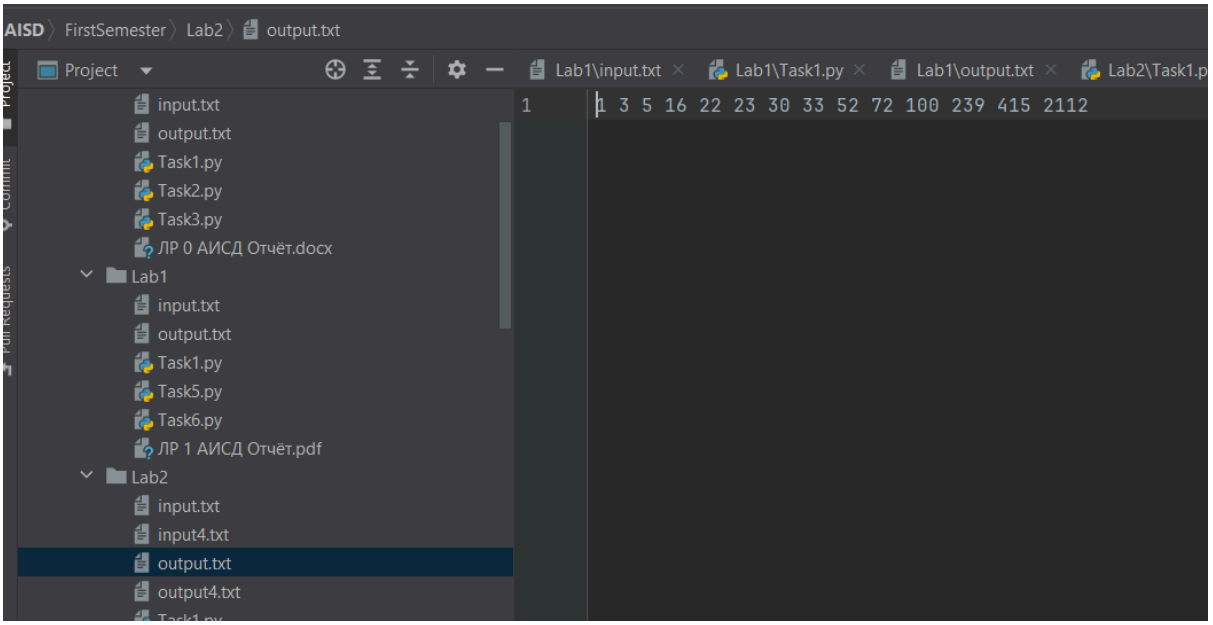
Код программы:

```
def merge_sort(arr):
    if len(arr) <= 1:
```

```
        return arr
    mid = len(arr) // 2
    left = merge_sort(arr[:mid])
    right = merge_sort(arr[mid:])
    return merge(left, right)

def merge(left, right):
    merged = []
    i = j = 0
    while i < len(left) and j < len(right):
        if left[i] <= right[j]:
            merged.append(left[i])
            i += 1
        else:
            merged.append(right[j])
            j += 1
    # Добавление оставшихся элементов
    merged.extend(left[i:])
    merged.extend(right[j:])
    return merged

with open('input.txt', 'r') as f:
    n = int(f.readline().strip())
    arr = list(map(int, f.readline().split()))

# Сортировка
sorted_arr = merge_sort(arr)

with open('output.txt', 'w') as f:
    f.write(' '.join(map(str, sorted_arr)))
```

Результат работы кода на примерах:


Project
Lab1\input.txt
Lab1\Task1.py
output.txt
Lab2\Task1.py
AISD C:\Users\User\PycharmProjects\AISD
FirstSemester
Lab0
input.txt
output.txt
Task1.py
Task2.py
Task3.py
ЛР 0 АИСД Отчёт.docx
Lab1
input.txt
output.txt
Task1.py
Task5.py
Task6.py
ЛР 1 АИСД Отчёт.pdf
Lab2
input.txt
14
1 239 33 415 5 22 72 52 2112 3 16 23 30 100


AISD
FirstSemester
Lab2
output.txt
Project
Lab1\input.txt
Lab1\Task1.py
Lab1\output.txt
Lab2\Task1.p
input.txt
output.txt
Task1.py
Task2.py
Task3.py
ЛР 0 АИСД Отчёт.docx
Lab1
input.txt
output.txt
Task1.py
Task5.py
Task6.py
ЛР 1 АИСД Отчёт.pdf
Lab2
input.txt
input4.txt
output.txt
output4.txt
1 3 5 16 22 23 30 33 52 72 100 239 415 2112

4 задача. Бинарный поиск

В этой задаче вы реализуете алгоритм бинарного поиска, который позволяет очень эффективно искать (даже в огромных) списках при условии, что список отсортирован. Цель - реализация алгоритма двоичного (бинарного) поиска.

- **Формат входного файла (input.txt).** В первой строке входного файла содержится число n ($1 \leq n \leq 10^5$) — число элементов в массиве, и последовательность $a_0 < a_1 < ... < a_{n-1}$ из n **различных** положительных целых чисел в порядке возрастания, $1 \leq a_i \leq 10^9$ для всех $0 \leq i < n$. Следующая строка содержит число k, $1 \leq k \leq 10^5$ и k положительных целых чисел $b_0, ...b_{k-1}$, $1 \leq b_j \leq 10^9$ для всех $0 \leq j < k$.
- **Формат выходного файла (output.txt).** Для всех i от 0 до $k-1$ вывести индекс $0 \leq j \leq n-1$, такой что $a_i = b_j$ или -1, если такого числа в массиве нет.
- Ограничение по времени. 2сек.
- Ограничение по памяти. 256 мб.
- Пример:

input.txt	output.txt
5 1 5 8 12 13 5 8 1 23 1 11	2 0 -1 0 -1

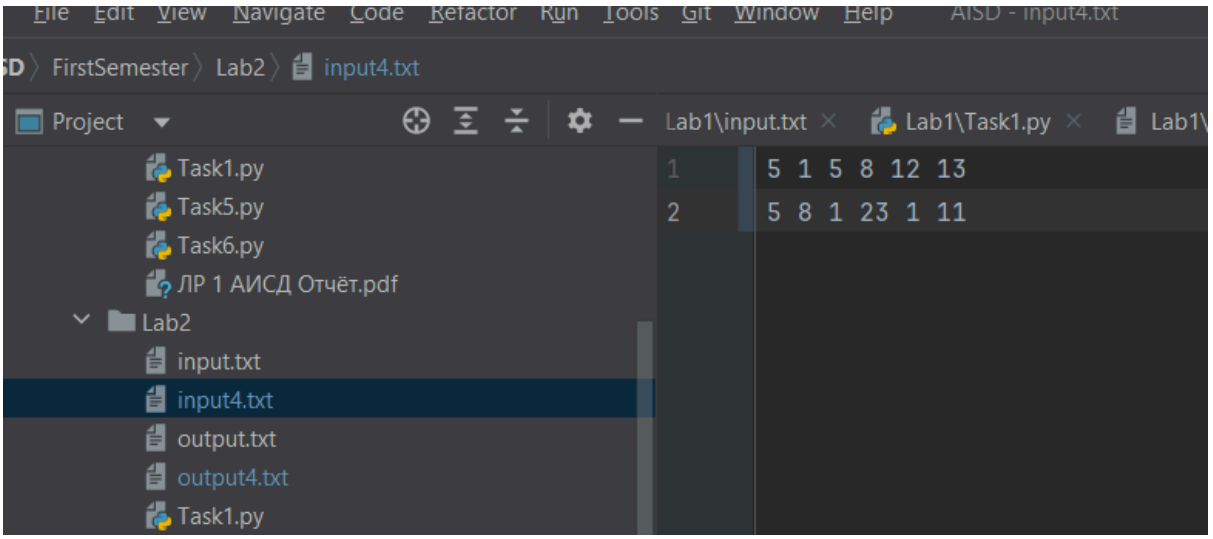
В этом примере есть возрастающая последовательность из $a_0 = 1, a_1 = 5, a_2 = 8, a_3 = 12$ и $a_4 = 13$ длиной в $n = 5$ и пять чисел для поиска: 8 1 23 1 11. Видно, что $a_2 = 8$ и $a_0 = 1$, но чисел 23 и 11 нет в последовательности a, поэтому они имеют индекс -1. В итоге ответ: 2 0 -1 0 -1.

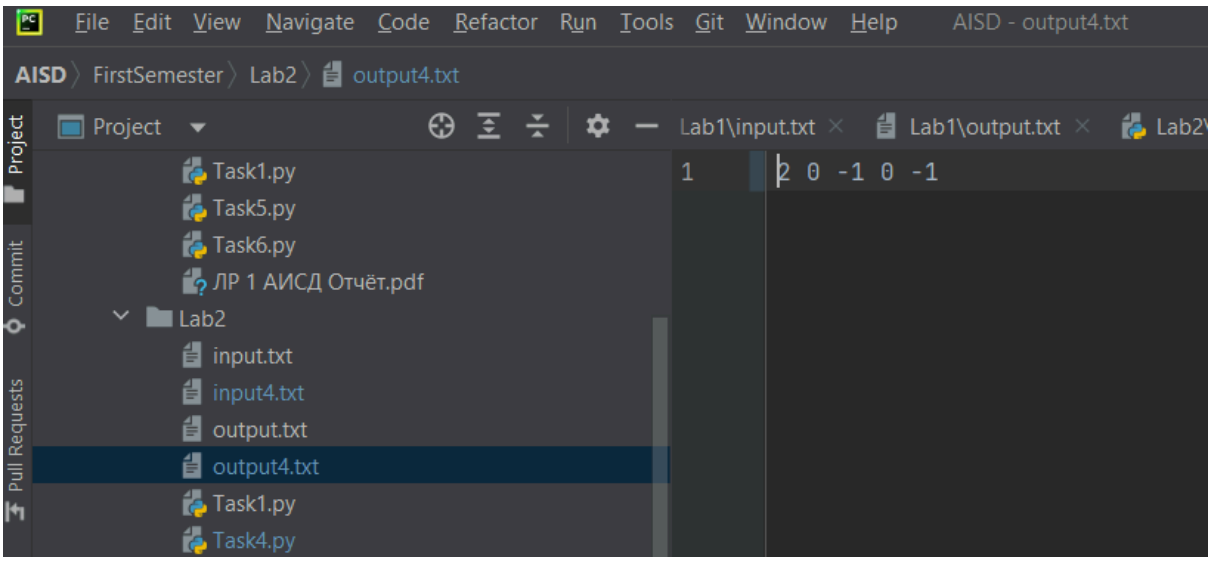
Код программы:

```
def binary_search(array, target):
    left, right = 0, len(array) - 1
    while left <= right:
        mid = (left + right) // 2
        if array[mid] == target:
            return mid
        elif array[mid] < target:
            left = mid + 1
        else:
            right = mid - 1
```

```
    return -1

with open("input4.txt", "r") as infile:
    a_line = infile.readline().split()
    n = int(a_line[0])
    a = list(map(int, a_line[1:n+1]))

    b_line = infile.readline().split()
    k = int(b_line[0])
    b = list(map(int, b_line[1:k+1]))

# Поиск каждого элемента из b в a
results = []
for number in b:
    index = binary_search(a, number)
    results.append(str(index))

with open("output4.txt", "w") as outfile:
    outfile.write(" ".join(results))
```

Результат работы кода на примерах:




File Edit View Navigate Code Refactor Run Tools Git Window Help
AISD - output4.txt
AISD
FirstSemester
Lab2
output4.txt
Project
Commit
Pull Requests
Lab1\input.txt
Lab1\output.txt
Task1.py
Task5.py
Task6.py
ЛР 1 АИСД Отчёт.pdf
Lab2
input.txt
input4.txt
output.txt
output4.txt
Task1.py
Task4.py
1
2 0 -1 0 -1

Задачу умножения матриц достаточно легко разбить на подзадачи, поскольку произведение можно составлять из *блоков*. Разобьём каждую из матриц X и Y на четыре блока размера $\frac{n}{2} \times \frac{n}{2}$:

$$X = \begin{bmatrix} A & B \\ C & D \end{bmatrix}, \; Y = \begin{bmatrix} E & F \\ G & H \end{bmatrix},$$

Тогда их произведение выражается в терминах этих блоков по обычной формуле умножения матриц :

$$XY = \begin{bmatrix} A & B \\ C & D \end{bmatrix} \cdot \begin{bmatrix} E & F \\ G & H \end{bmatrix} = \begin{bmatrix} AE + BG & AF + BH \\ CE + DG & CF + DH \end{bmatrix}$$

Вычислив рекурсивно восемь произведений $AE, BG, AF, BH, CE, DG, CF, DH$ и просуммировав их за время $O(n^2)$, мы вычислим необходимое нам произведение матриц. Соответствующее рекуррентное соотношение на время работы алгоритма

$$T(n) = 8T(\frac{n}{2}) + O(n^2).$$

Какое получилось время у предыдущего рекурсивного алгоритма? Да, ничуть не лучше наивного. Однако его можно ускорить с помощью алгебраического трюка: для вычисления произведения XY достаточно перемножить ***семь*** пар матриц размера $\frac{n}{2} \times \frac{n}{2}$, после чего хитрым образом (*и как только Штрассен догадался?*) получить ответ:

$$XY = \begin{bmatrix} P_5 + P_4 - P_2 + P_6 & P_1 + P_2 \\ P_3 + P_4 & P_1 + P_5 - P_3 - P_7 \end{bmatrix}$$

где

$$\begin{aligned} P_1 &= A(F - H), & P_5 &= (A + D)(E + H), \\ P_2 &= (A + B)H, & P_6 &= (B - D)(G + H), \\ P_3 &= (C + D)E, & P_7 &= (A - C)(E + F), \\ & & P_4 &= D(G - E). \end{aligned}$$

- **Цель**. Применить метод Штрассена для умножения матриц и сравнить его с простым методом. *Найти размер матриц n, при котором метод Штрассена работает существенно быстрее простого метода.*
- **Формат входа**. Стандартный ввод или input.txt. Первая строка - размер **квадратных** матриц n для умножения. Следующие строки соответсвенно сами значения матриц A и B.
- **Формат выхода**. Стандартный вывод или output.txt. Матрица $C = A \cdot B$.

Код программы

```
import numpy as np
```

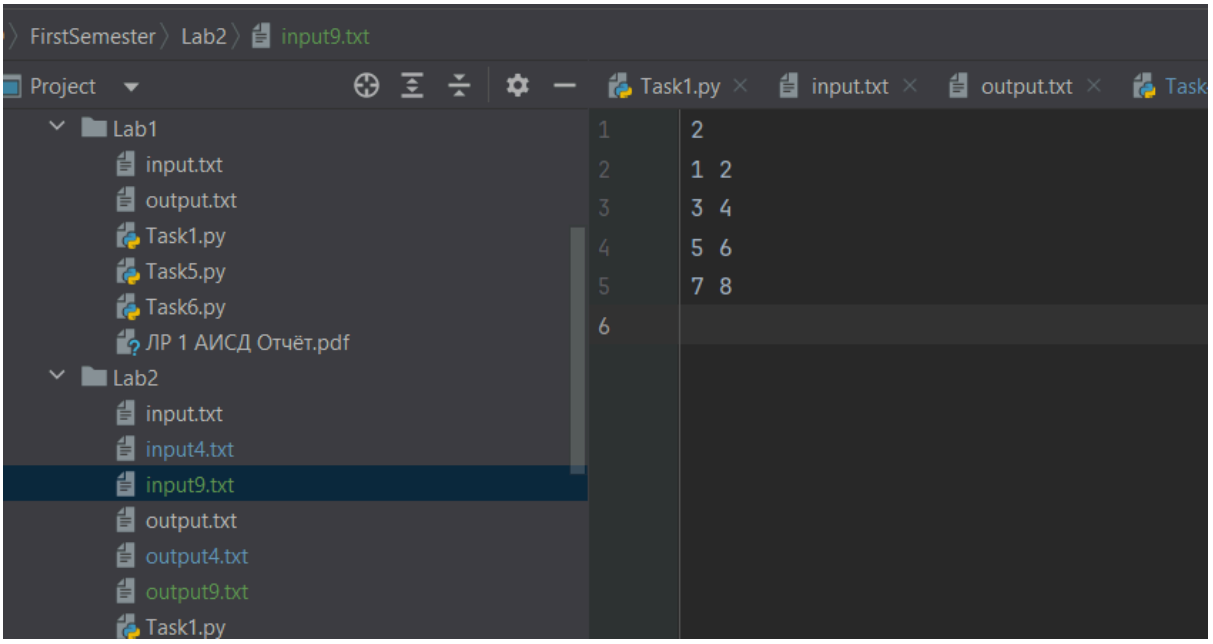
```
def strassen_multiply(A, B):
    """
    Реализация умножения матриц методом Штрассена.
    """
    # Получаем размеры матриц
    n, m = A.shape if isinstance(A, np.ndarray) else (1, len(A))
    p, q = B.shape if isinstance(B, np.ndarray) else (len(B), 1)

    # Проверяем базовый случай для корректного завершения рекурсии
    if n <= 1 or m <= 1 or p <= 1 or q <= 1:
        # Используем np.dot() вместо оператора * для матричного умножения
        return np.dot(A, B)

    # Для поддержки алгоритма Штрассена матрицы должны быть квадратными и размера 2^n
    # Находим ближайшую степень двойки
    max_dim = max(n, m, p, q)
    next_pow2 = 2 ** int(np.ceil(np.log2(max_dim)))

    # Создаем и заполняем расширенные матрицы
    A_padded = np.zeros((next_pow2, next_pow2))
    A_padded[:n, :m] = A

    B_padded = np.zeros((next_pow2, next_pow2))
    B_padded[:p, :q] = B

    # Рекурсивное вычисление произведения
    C_padded = _strassen_recursive(A_padded, B_padded)

    # Извлекаем нужную часть результата
    C = C_padded[:n, :q]
    return C
```

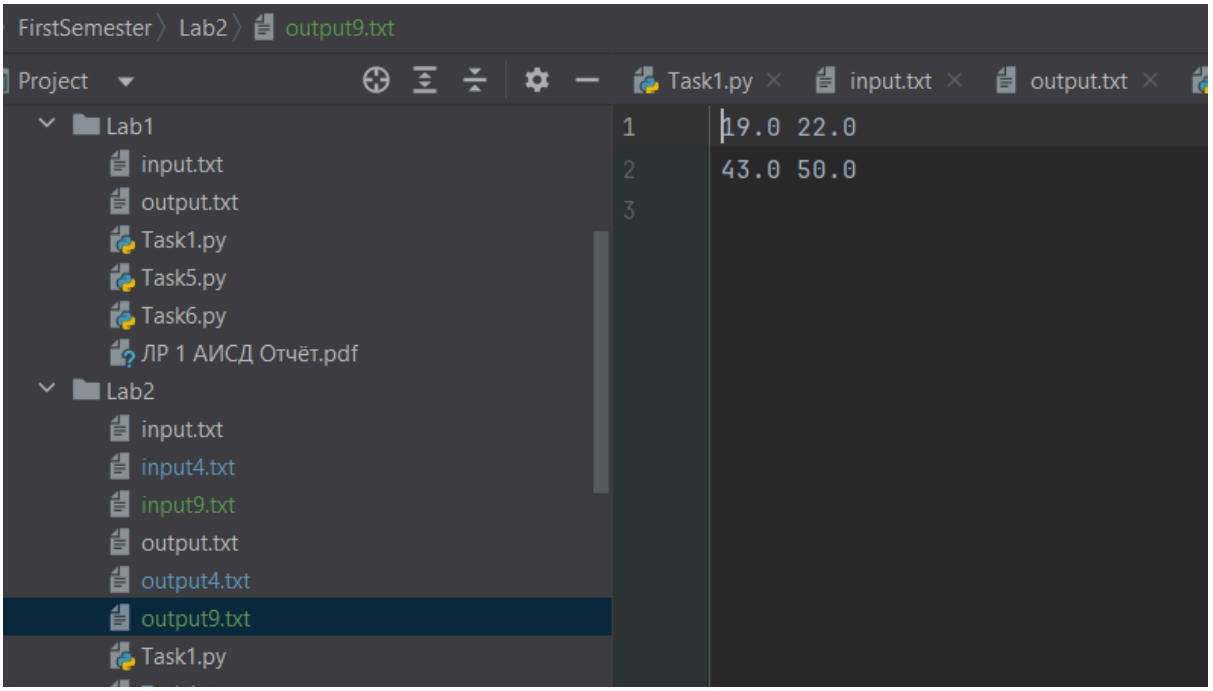
```
def _strassen_recursive(A, B):
    """Рекурсивная часть алгоритма Штрассена."""
    n = A.shape[0]

    # Базовый случай
    if n == 1:
        return np.array([[A[0, 0] * B[0, 0]]])

    # Разделение матриц на блоки
    mid = n // 2

     # Используем обозначения из задания: X = [A B; C
D], Y = [E F; G H]
    a = A[:mid, :mid]
    b = A[:mid, mid:]
    c = A[mid:, :mid]
    d = A[mid:, mid:]

    e = B[:mid, :mid]
    f = B[:mid, mid:]
    g = B[mid:, :mid]
    h = B[mid:, mid:]

    # Вычисляем 7 вспомогательных матриц P₁-P₇
    p1 = _strassen_recursive(a, f - h)
    p2 = _strassen_recursive(a + b, h)
    p3 = _strassen_recursive(c + d, e)
    p4 = _strassen_recursive(d, g - e)
    p5 = _strassen_recursive(a + d, e + h)
    p6 = _strassen_recursive(b - d, g + h)
    p7 = _strassen_recursive(a - c, e + f)

    # Вычисляем блоки результирующей матрицы
    # XY = [P₅ + P₄ - P₂ + P₆     P₁ + P₂
    #       P₃ + P₄                P₁ + P₅ - P₃ - P₇]
    c11 = p5 + p4 - p2 + p6
    c12 = p1 + p2
```

```
    c21 = p3 + p4
    c22 = p1 + p5 - p3 - p7

    # Собираем результат
    C = np.zeros((n, n))
    C[:mid, :mid] = c11
    C[:mid, mid:] = c12
    C[mid:, :mid] = c21
    C[mid:, mid:] = c22

    return C


# Чтение матриц из файла
def read_matrices(filename):
    """Чтение матриц из входного файла."""
    try:
        with open(filename, 'r') as f:
            n = int(f.readline().strip())

            # Чтение матрицы A
            A = np.zeros((n, n))
            for i in range(n):
                line = f.readline().strip()
                if line:
                                A[i] = list(map(float,
line.split()))

            # Чтение матрицы B
            B = np.zeros((n, n))
            for i in range(n):
                line = f.readline().strip()
                if line:
                                B[i] = list(map(float,
line.split()))

        return A, B
    except Exception as e:
```

```
        print(f"Ошибка при чтении файла: {e}")
        return None, None

def write_matrix(C, filename):
    """Запись результата в выходной файл."""
    with open(filename, 'w') as f:
        for row in C:
            f.write(' '.join(map(str, row)) + '\n')


def main():
    try:
        A, B = read_matrices('input9.txt')
        if A is None or B is None:
            return

        C = strassen_multiply(A, B)
        write_matrix(C, 'output9.txt')
    except Exception as e:
        print(f"Произошла ошибка: {e}")


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

Результат работы кода на примерах:


FirstSemester
Lab2
input9.txt
Project
Task1.py
input.txt
output.txt
Lab1
input.txt
output.txt
Task1.py
Task5.py
Task6.py
ЛР 1 АИСД Отчёт.pdf
Lab2
input.txt
input4.txt
input9.txt
output.txt
output4.txt
output9.txt
Task1.py
2
1 2
3 4
5 6
7 8


FirstSemester
Lab2
output9.txt
Project
Task1.py
input.txt
output.txt
Lab1
input.txt
output.txt
Task1.py
Task5.py
Task6.py
ЛР 1 АИСД Отчёт.pdf
Lab2
input.txt
input4.txt
input9.txt
output.txt
output4.txt
output9.txt
Task1.py
19.0 22.0
43.0 50.0