МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ

РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное бюджетное образовательное

учреждение высшего образования

«Казанский национальный исследовательский технический

университет им. А.Н. Туполева-КАИ»

(КНИТУ-КАИ)

Колледж информационных технологий «КИТ»

Дисциплина:

«Теория алгоритмов»

ЛАБОРАТОРНАЯ РАБОТА № 2

на тему:

ВЫЧИСЛЕНИЕ СЛОЖНОСТИ РАБОТЫ АЛГОРИТМОВ

|  |  |  |
| --- | --- | --- |
| Выполнил: | Обучающийся группы 4335 |  |
|  | Желваков А. С. |  |
| Проверил: | Валова П.А. |  |

Казань 2024г.

Вариант 9.

1. Константная сложность.

Задание:

|  |
| --- |
| Обмен значениями переменных. Создайте функцию, которая принимает два числа и возвращает их в обратном порядке. |

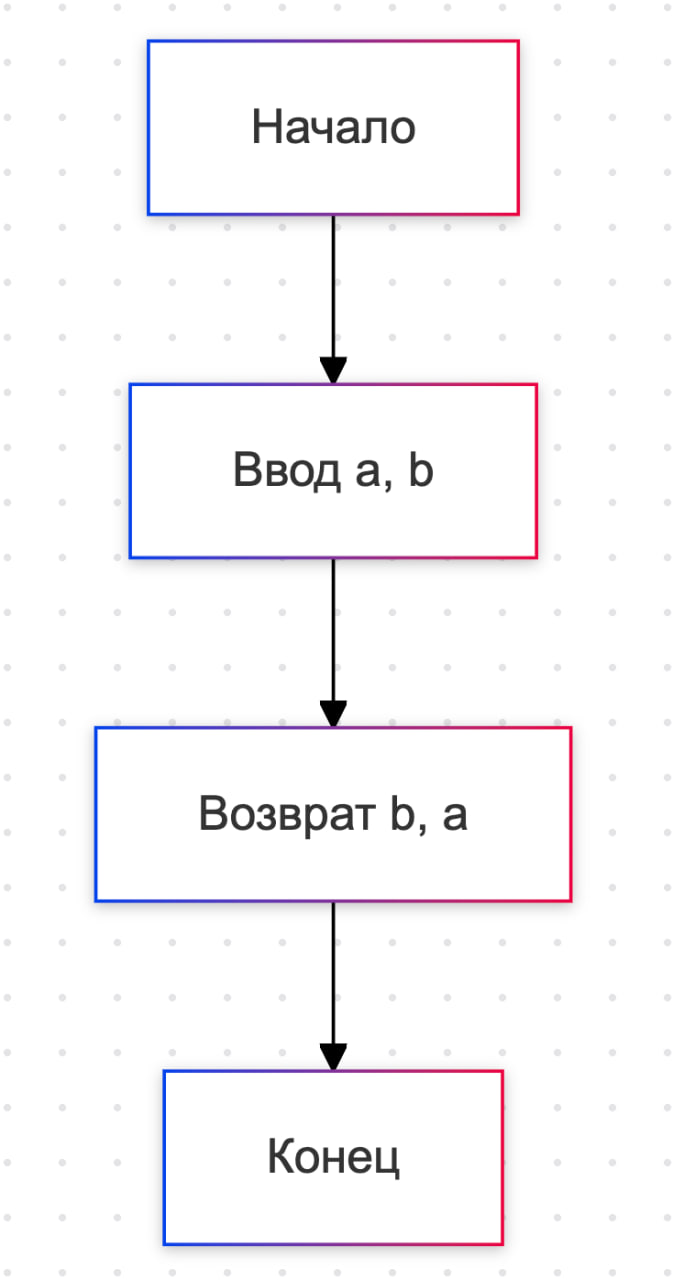


Рис. 1 - Блок-схема для константной сложности

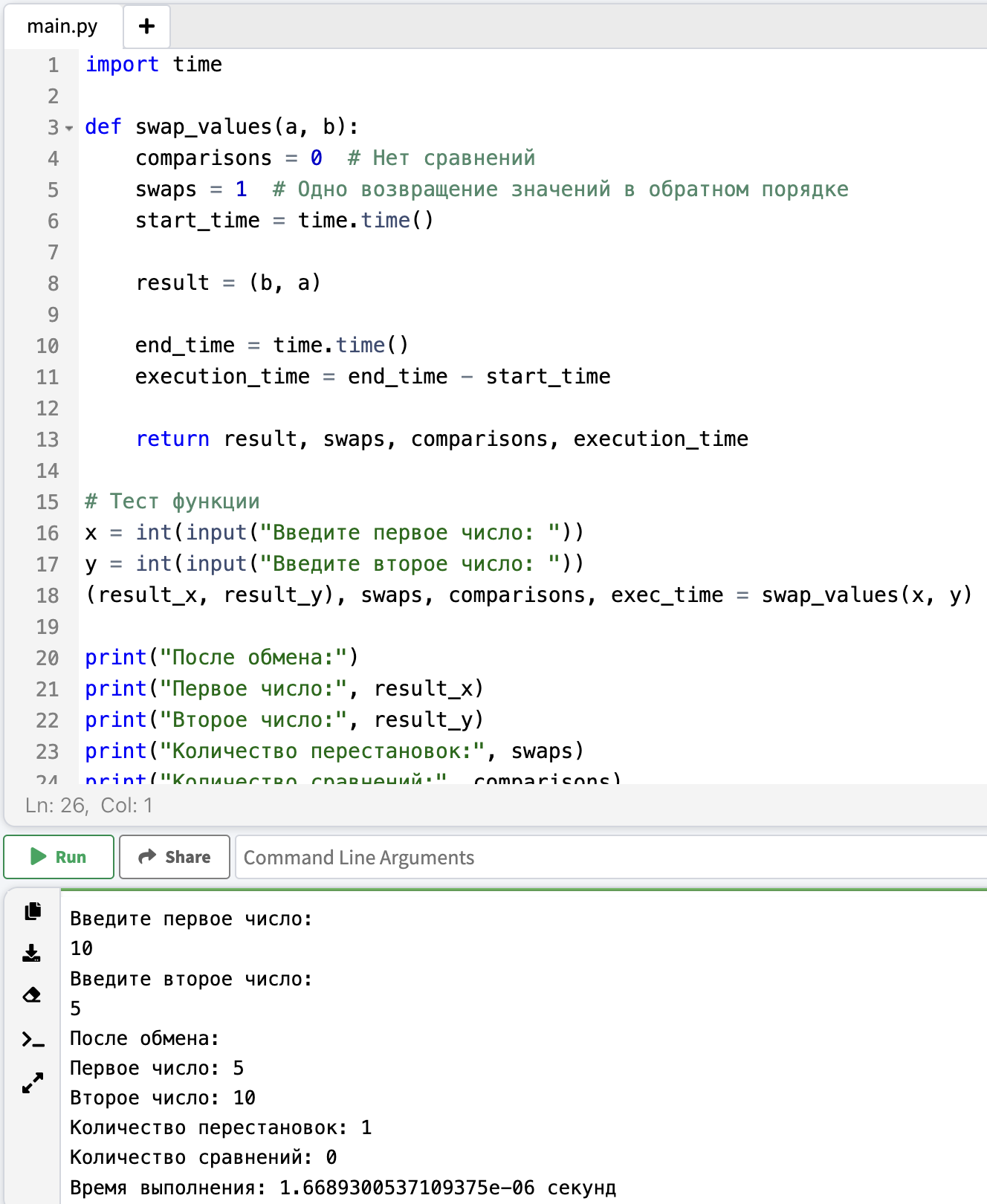


Рис. 2 - Работа программы для константной сложности

Объяснение

|  |
| --- |
| Объяснение работы и сложности  - Описание работы: Функция принимает два числа a и b и возвращает их в обратном порядке, то есть возвращает b и a вместо a и b. Это простой обмен значениями, без каких-либо дополнительных операций.    Сложность:  - Временная сложность — O(1), так как функция выполняет одну и ту же операцию (возвращение значений в обратном порядке) за постоянное время, независимо от значений a и b.  - Количество сравнений — 0, так как функция не выполняет сравнений.  - Количество перестановок — 1, так как значения a и b меняются местами один раз. |

1. Логарифмическая сложность

Задание:

|  |
| --- |
| Поиск в отсортированном массиве. Реализовать бинарный поиск для нахождения заданного элемента в отсортированном массиве. Если элемент не найден, вернуть -1. |

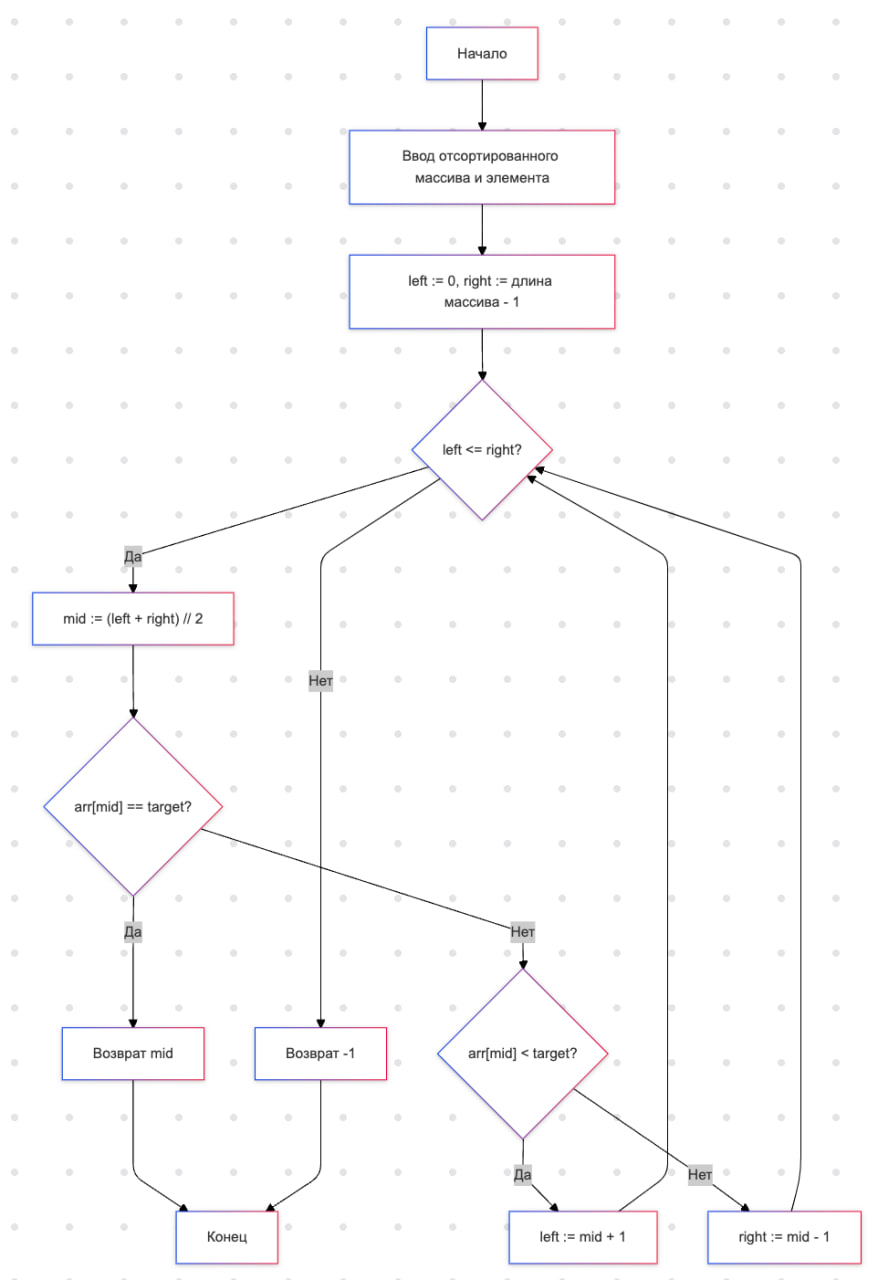


Рис. 3 - Блок-схема для логарифмической сложности

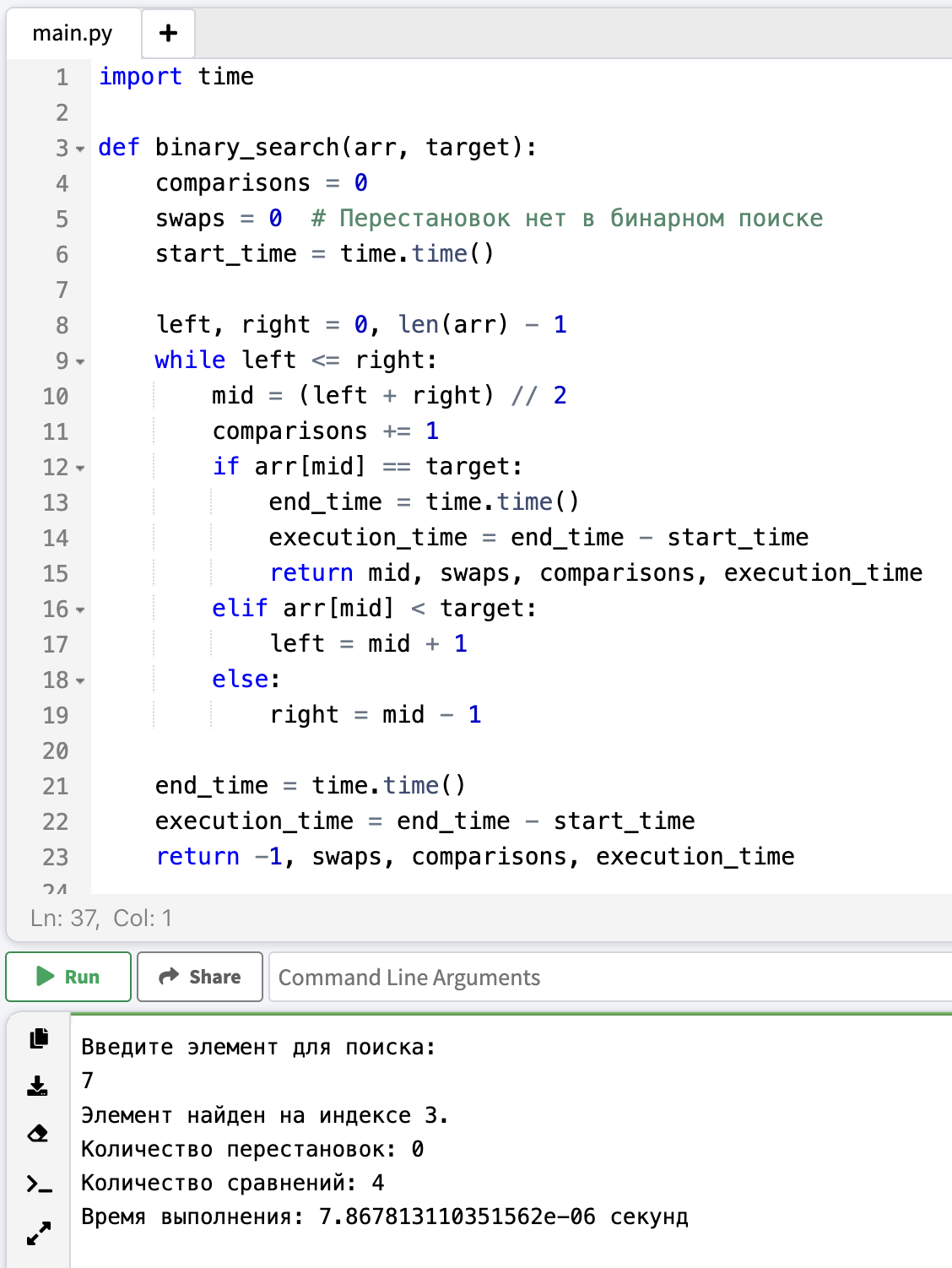


Рис. 4 - Работа программы для логарифмической сложности

Объяснение

|  |
| --- |
| Объяснение работы и сложности  - Описание работы: Бинарный поиск выполняется на отсортированном массиве. На каждой итерации массив делится пополам, и проверяется, находится ли искомый элемент в левой или правой половине. Процесс повторяется, пока не будет найден элемент или пока диапазон поиска не станет пустым.    Сложность:  - Временная сложность — O(logn), так как на каждой итерации размер диапазона поиска уменьшается вдвое.  - Количество сравнений — зависит от логарифмической глубины поиска, поскольку на каждой итерации выполняется одно сравнение с центральным элементом.  - Количество перестановок — 0, так как алгоритм бинарного поиска не изменяет порядок элементов в массиве. |

1. Линейная сложность

Задание:

|  |
| --- |
| Найти наибольший общий делитель всех элементов массива. |

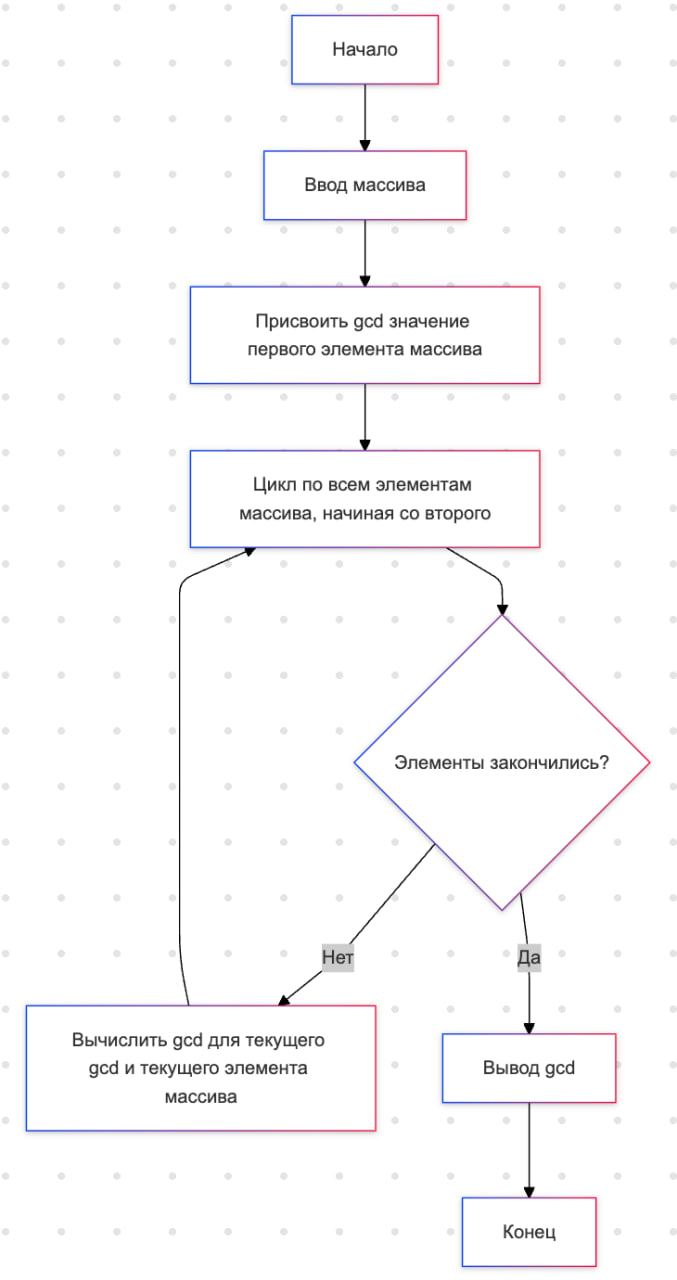


Рис. 5 - Блок-схема для линейной сложности

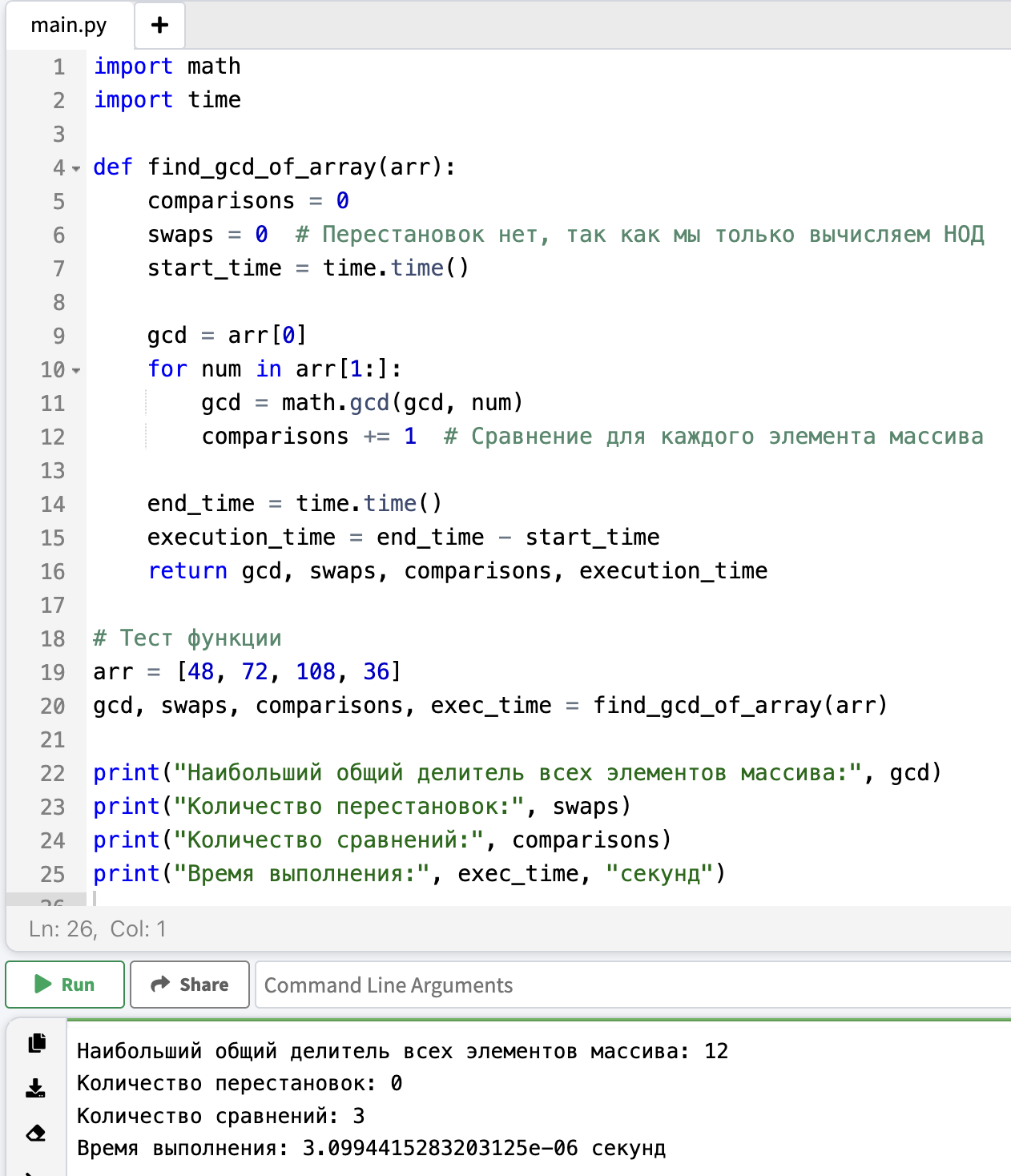


Рис. 6 - Работа программы для линейной сложности

Объяснение

|  |
| --- |
| Объяснение работы и сложности  - Описание работы: Алгоритм начинает с вычисления НОД для первого элемента массива и последовательно проходит по всем остальным элементам, вычисляя НОД текущего результата и следующего элемента. Этот процесс продолжается, пока не будет обработан весь массив.    Сложность:  - Временная сложность — O(n), поскольку мы проходим по каждому элементу массива один раз и выполняем вычисление НОД.  - Количество сравнений — n−1, так как мы вычисляем НОД для каждого следующего элемента массива, начиная со второго.  - Количество перестановок — 0, поскольку алгоритм не изменяет порядок элементов в массиве. |

1. Линейно-логарифмическая сложность

Задание:

|  |
| --- |
| Реализуйте сортировку слиянием. |

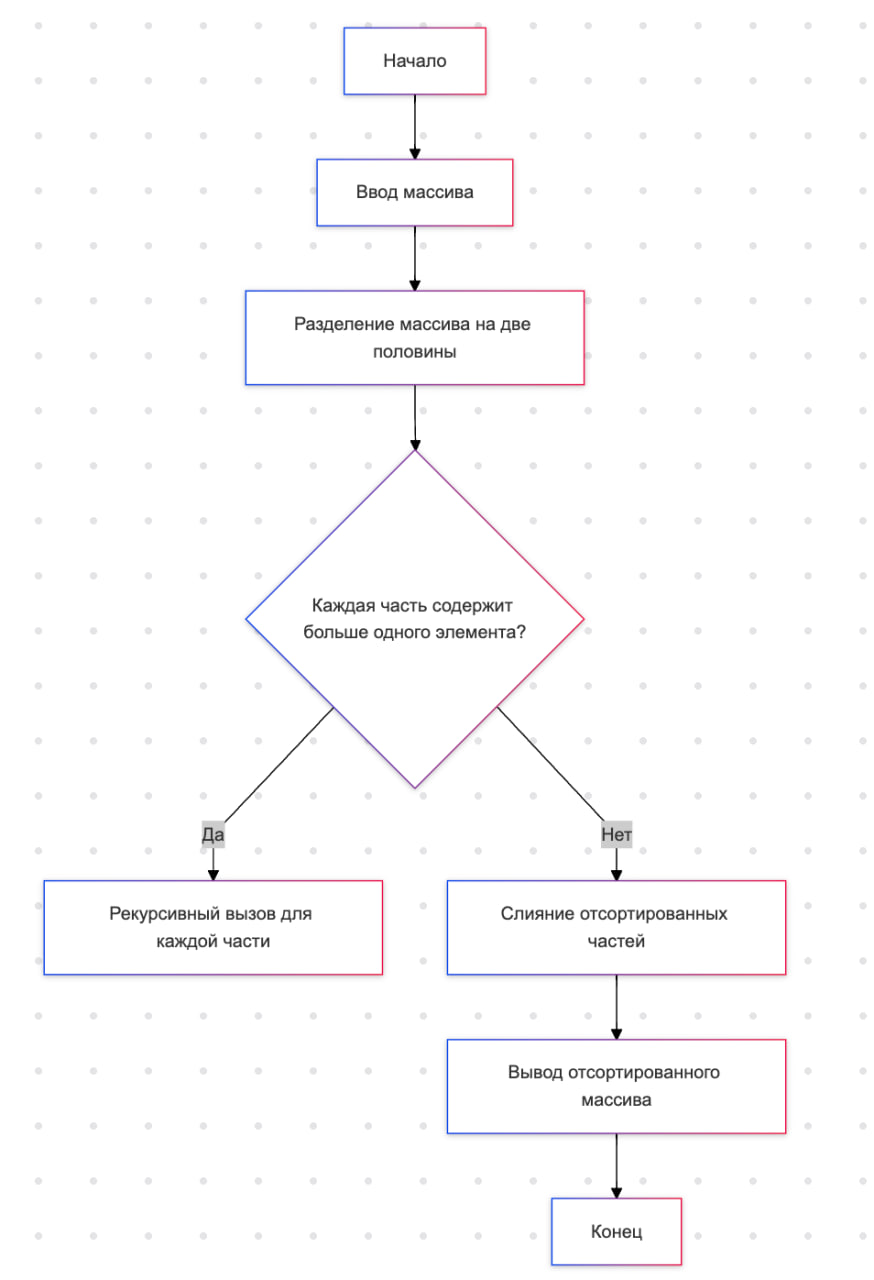


Рис. 7 - Блок-схема для линейно-логарифмической сложности

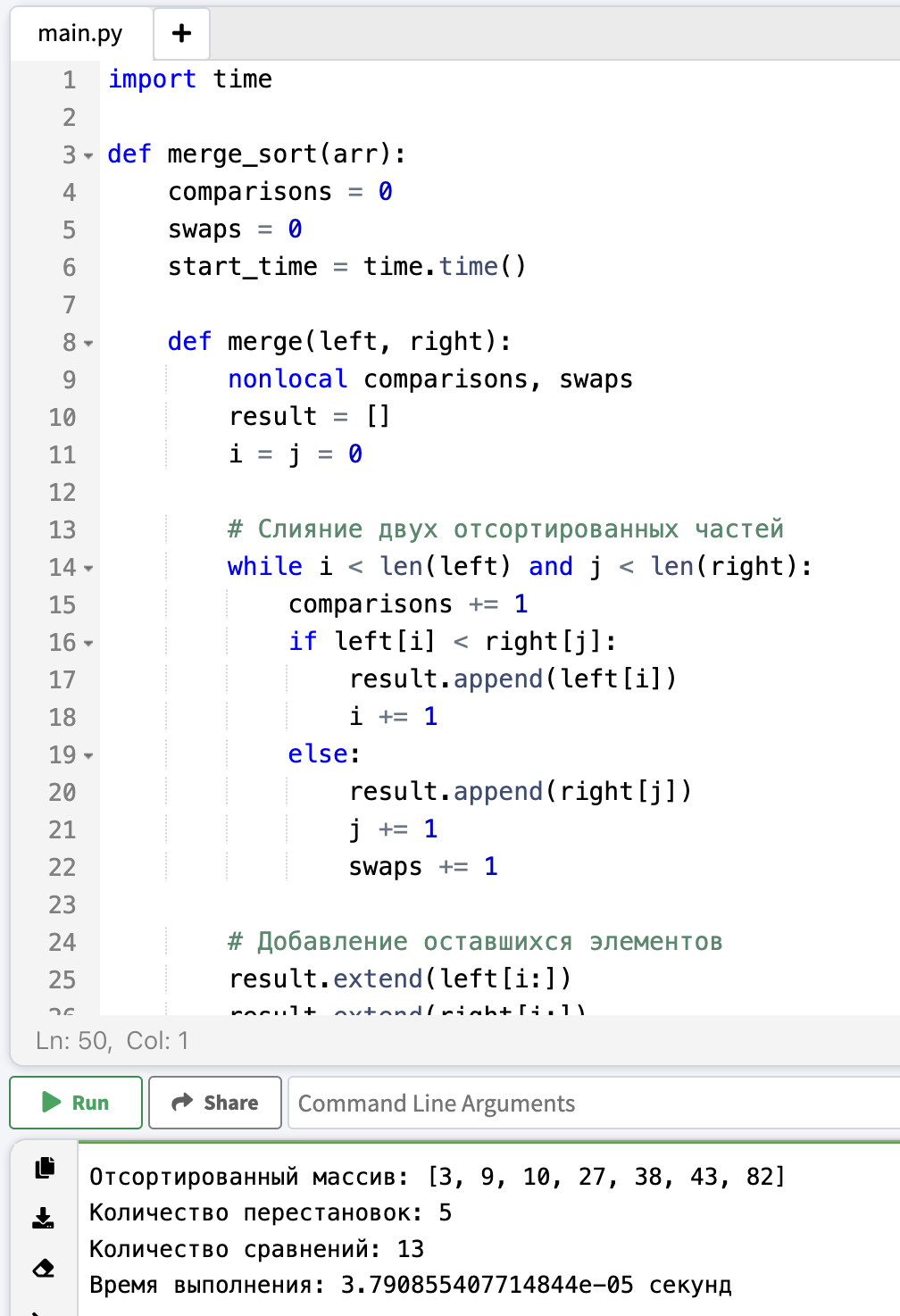


Рис. 8 - Работа программы для линейно-логарифмической сложности

Объяснение

|  |
| --- |
| Объяснение работы и сложности  - Описание работы: Алгоритм сортировки слиянием рекурсивно делит массив на две части, пока каждая часть не станет длиной в один элемент. Затем он выполняет слияние отсортированных частей, сравнивая элементы и добавляя их в новый отсортированный массив.    Сложность:  - Временная сложность — O(nlogn), поскольку на каждом уровне рекурсии массив делится пополам (logn уровней), а слияние на каждом уровне выполняется за линейное время O(n).  - Количество сравнений — O(nlogn), так как элементы сравниваются при каждом слиянии.  - Количество перестановок — отслеживается в процессе слияния, когда элементы из одной половины добавляются к результату. |

1. Квадратичная сложность

Задание:

|  |
| --- |
| Реализовать алгоритм, который найдет наибольший элемент в каждой строке квадратной матрицы. |

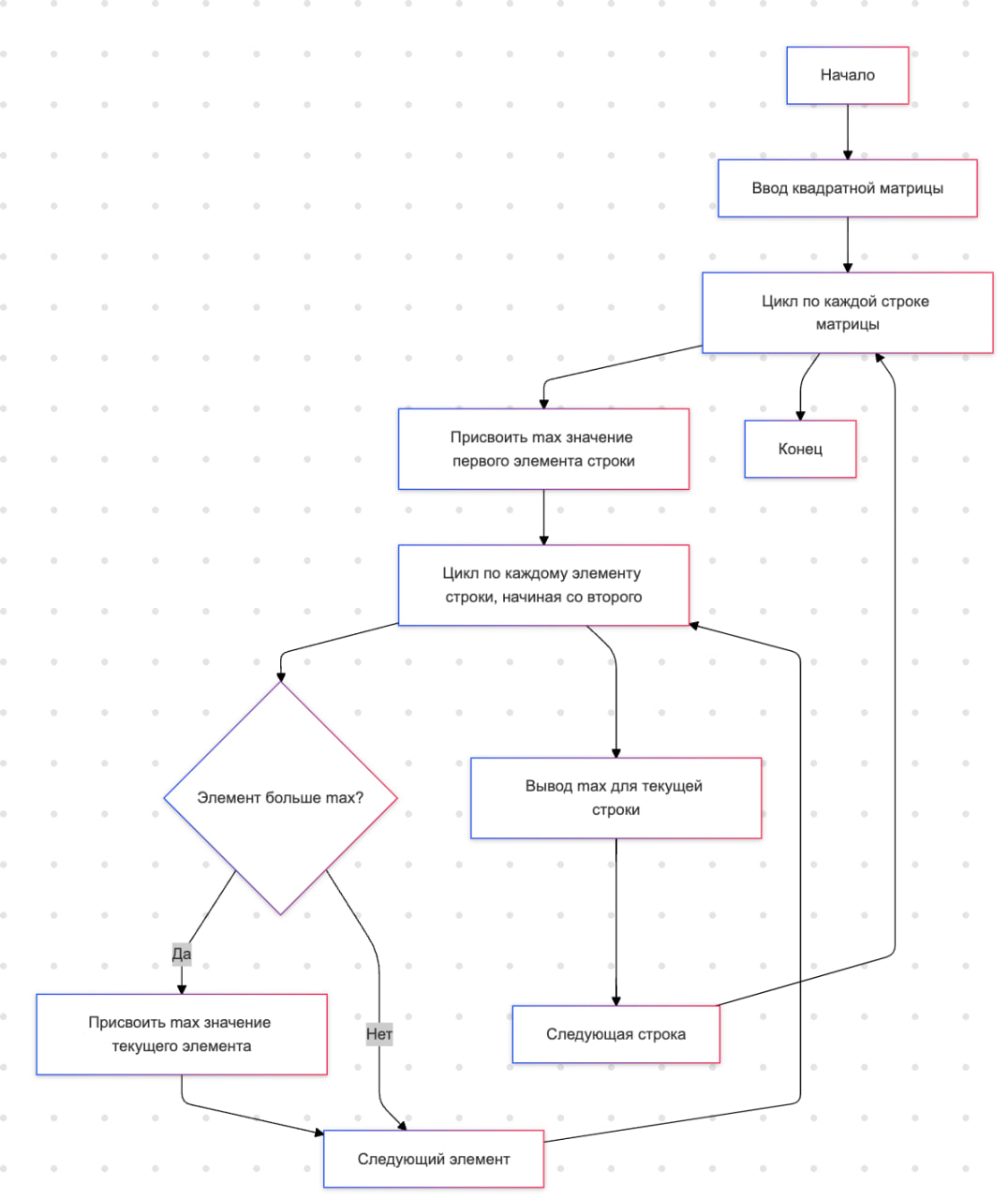
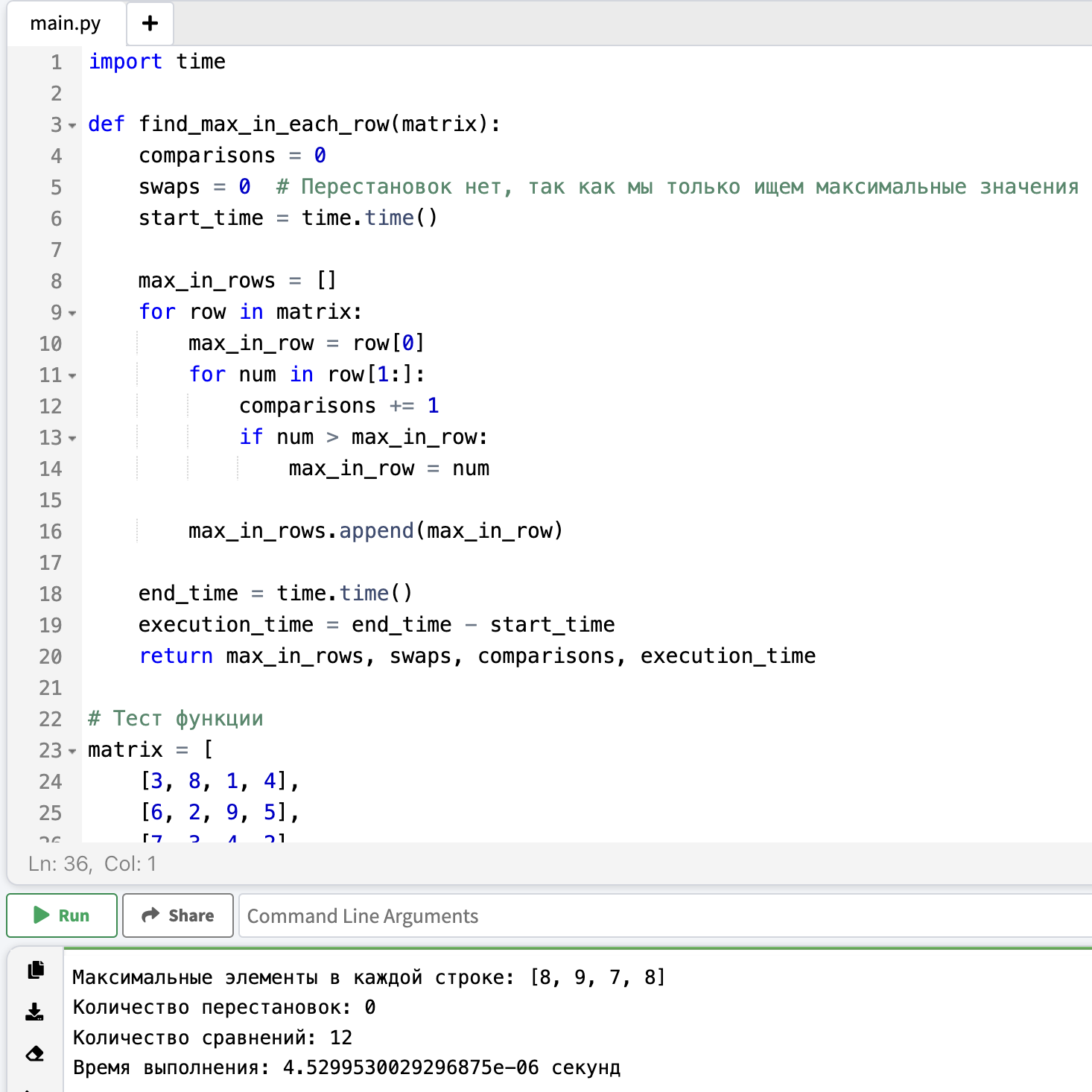


Рис. 9 - Блох-схема для квадратичной сложности

Рис. 10 - Работа программы для квадратичной сложности

Объяснение

|  |
| --- |
| Объяснение работы и сложности  - Описание работы: Алгоритм проходит по каждой строке квадратной матрицы. Для каждой строки он инициализирует переменную max\_in\_row значением первого элемента строки. Затем проходит по остальным элементам строки, сравнивая каждый из них с max\_in\_row. Если текущий элемент больше max\_in\_row, то max\_in\_row обновляется этим значением. После завершения прохода по строке результат сохраняется.    Сложность:  - Временная сложность — O(n\*\*2), так как для каждой из n строк выполняется n операций для нахождения максимума.  - Количество сравнений — n×(n−1), так как мы выполняем n−1 сравнений для каждой строки.  - Количество перестановок — 0, поскольку алгоритм не меняет порядок элементов. |

1. Экспоненциальная сложность

Задание:

|  |
| --- |
| Алгоритм, который генерирует все подмножества множества. |

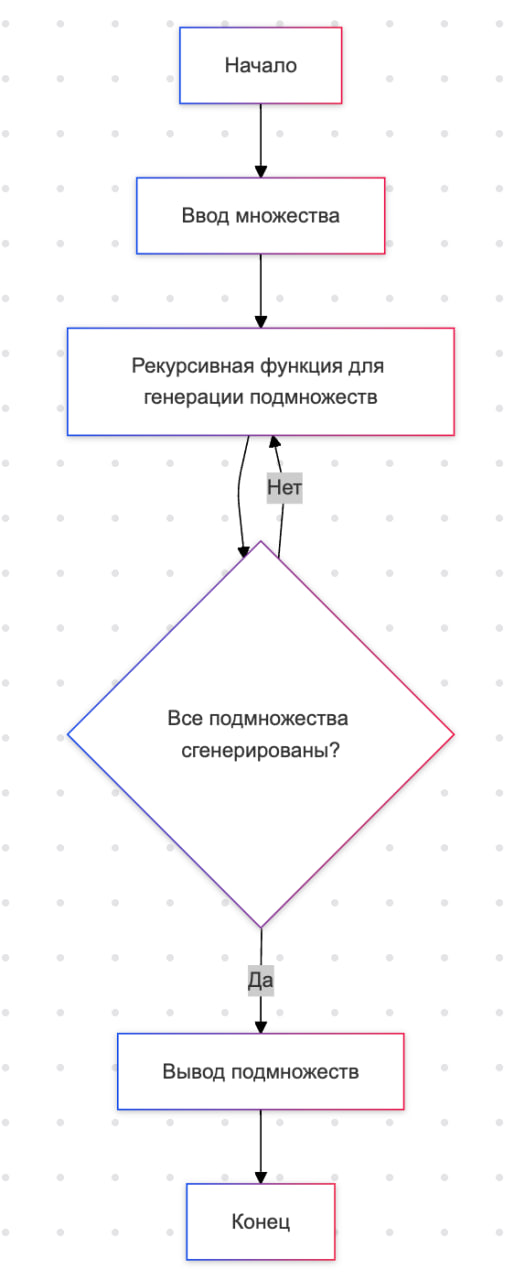
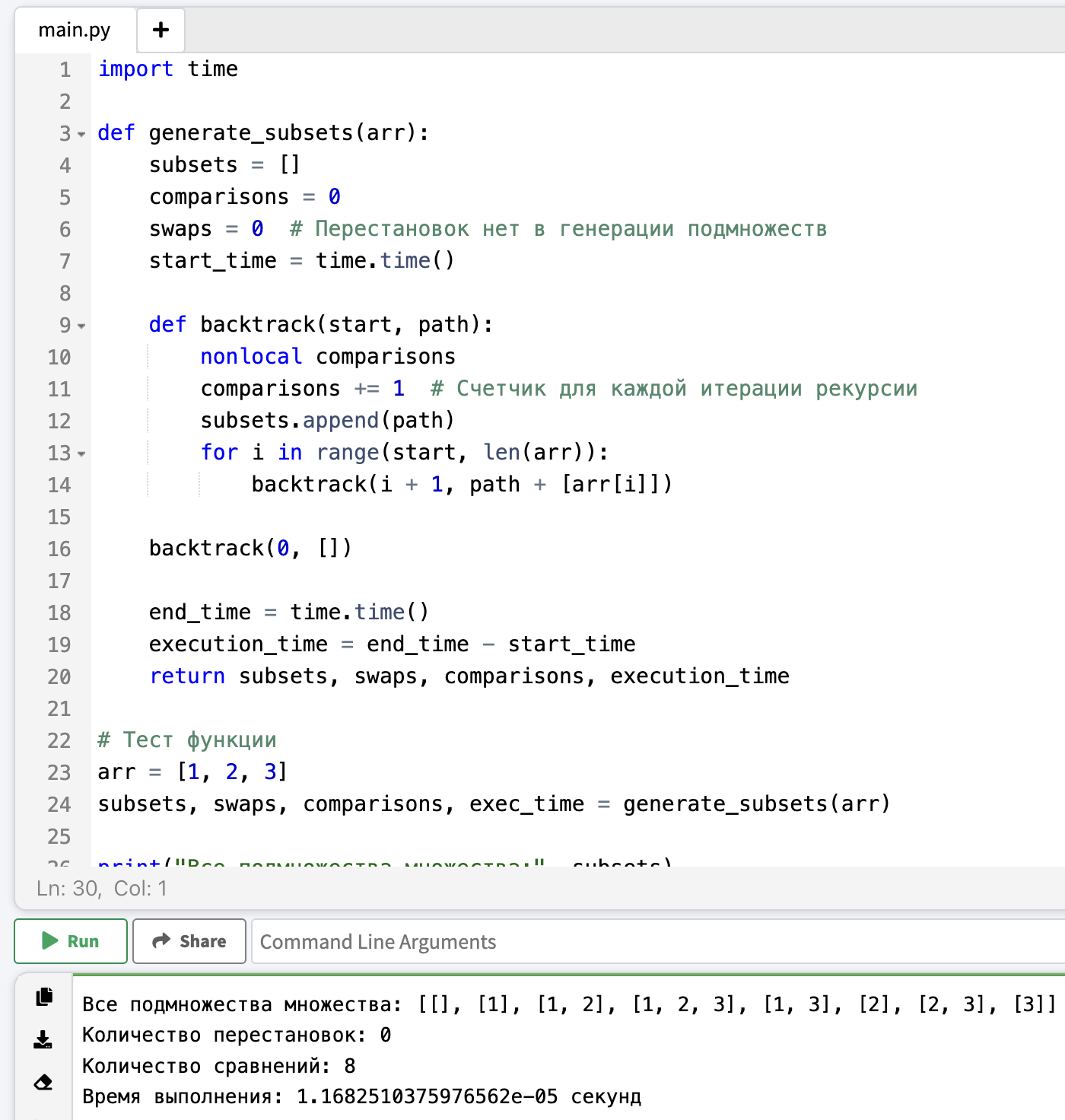


Рис. 11 - Блок-схема для экспоненциальной сложности

Рис. 12 - Работа программы для экспоненциальной сложности

Объяснение

|  |
| --- |
| Объяснение работы и сложности  - Описание работы: Алгоритм использует рекурсию для генерации всех подмножеств заданного множества. На каждом уровне рекурсии он добавляет текущий элемент в подмножество или пропускает его, чтобы создать все возможные комбинации.    Сложность:    - Временная сложность — O(n\*\*2), так как для множества из n элементов существует 2\*\*n возможных подмножеств.  - Количество сравнений — увеличивается экспоненциально с размером входного множества, так как для каждого элемента выполняются две рекурсивные ветки.  - Количество перестановок — отсутствует, поскольку алгоритм только генерирует и добавляет подмножества в результирующий список. |

1. Факториальная сложность

Задание:

|  |
| --- |
| Алгоритм, который генерирует все перестановки элементов в массиве. |

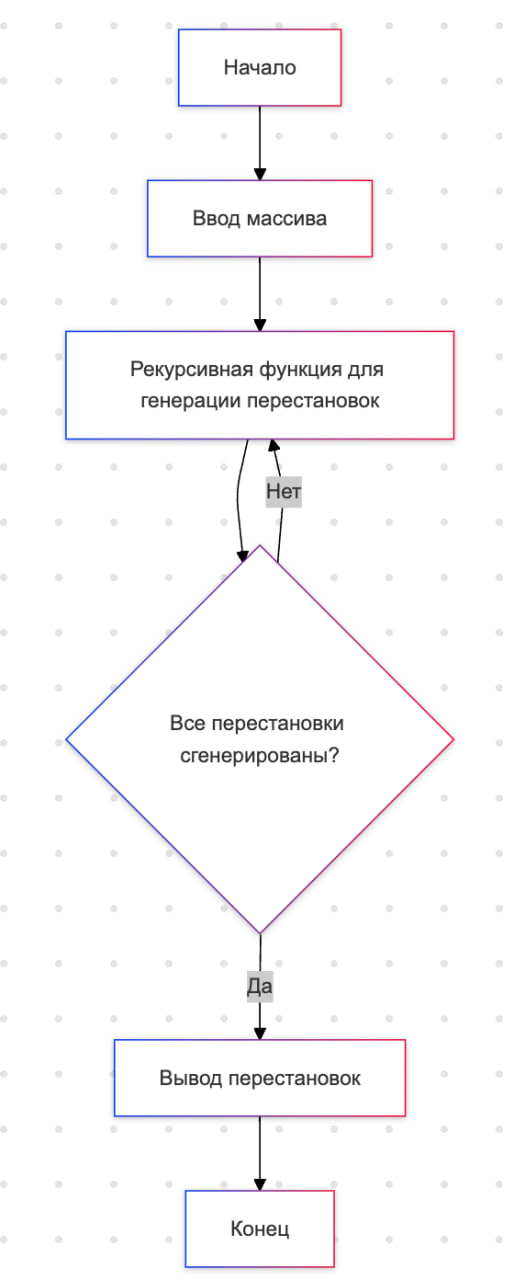
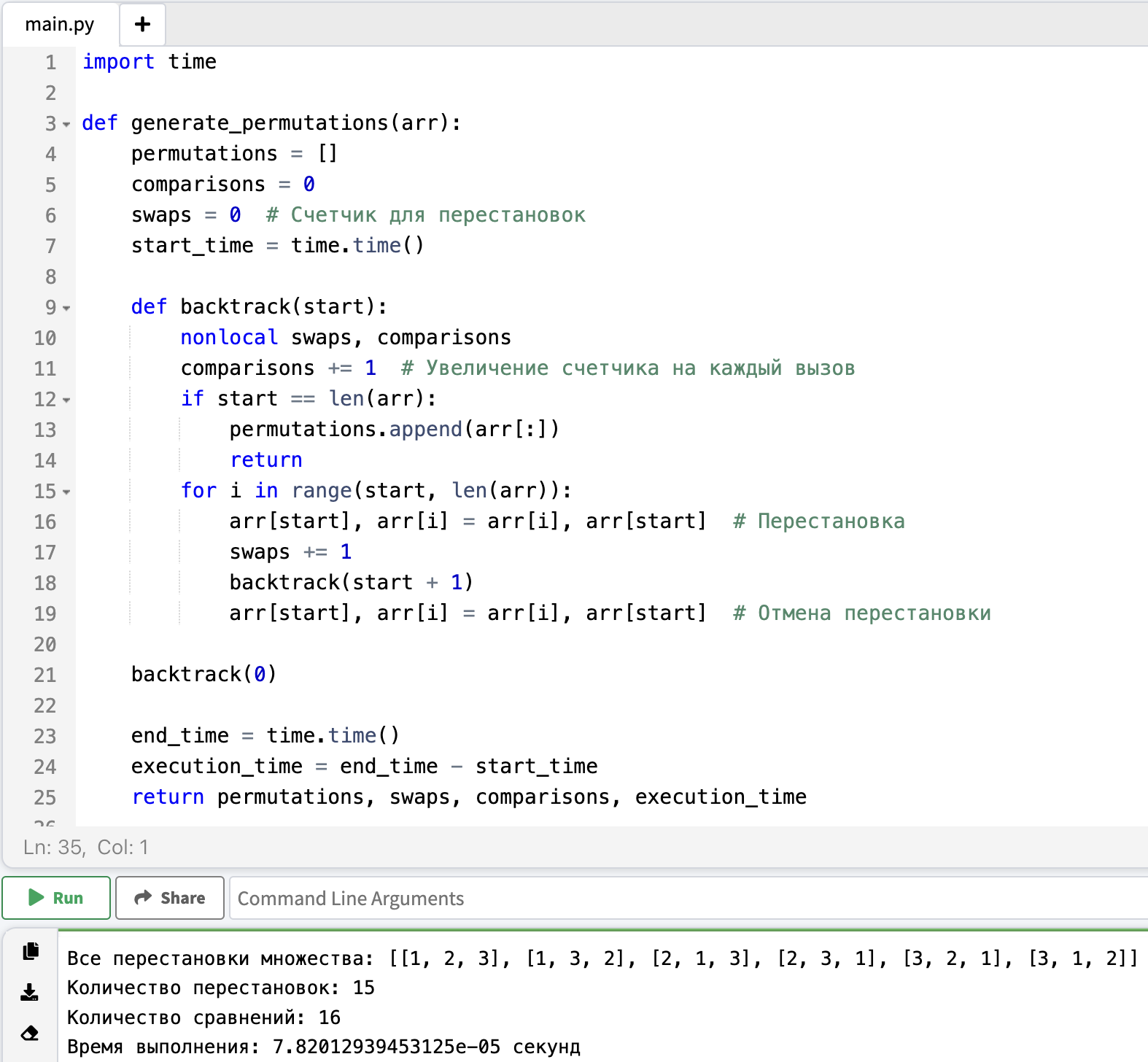


Рис. 13 - Блок-схема для факториальной сложности

Рис. 14 - Работа программы для факториальной сложности

Объяснение

|  |
| --- |
| Объяснение работы и сложности  - Описание работы: Алгоритм использует рекурсивный подход для генерации всех перестановок массива. На каждом уровне рекурсии элементы массива переставляются, затем рекурсия продолжается для следующих позиций. После каждой перестановки она отменяется, чтобы рассмотреть следующий вариант.    Сложность  - Временная сложность — O(n!), так как для массива из n элементов существует n! возможных перестановок.  - Количество сравнений — увеличивается факториально, так как каждый уровень рекурсии проверяет все возможные перестановки оставшихся элементов.  - Количество перестановок — также факториальное, поскольку алгоритм выполняет перестановку на каждом уровне рекурсии для каждого элемента. |

Листинг

<https://github.com/ArseniyZh/CIT/tree/main/3rd_year/%D0%A2%D0%90/%D0%BB%D0%B0%D0%B1%D1%8B/2>