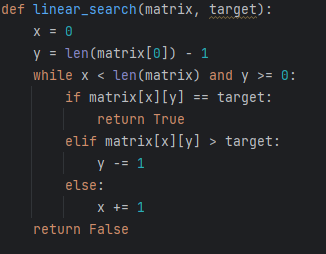
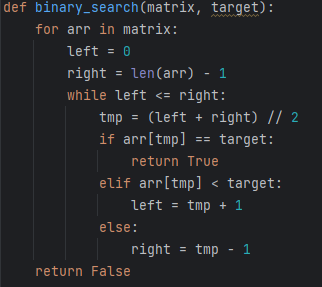
**Лабораторная работа №1**

1. Используемые алгоритмы
   1. Поиск лесенкой (О(n + m))



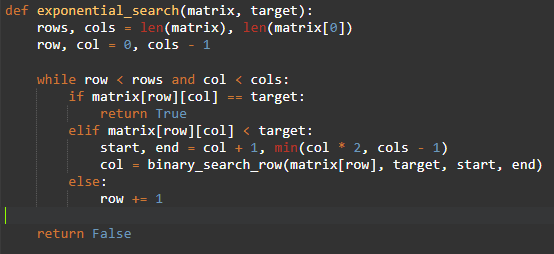
* 1. Бинарный поиск (О(m\*log(n)))

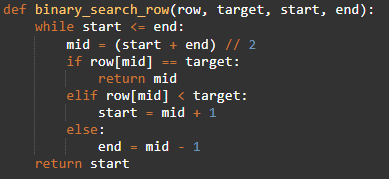
Перебираем каждую строку матрицы и запускаем в ней бинарный поиск.



* 1. Экспоненциальный поиск (О(m · (log n – log m + 1)))

Перебираем каждую строку матрицы, экспоненциально ищем отрезок, в котором может присутствовать наш элемент, запускаем на этом отрезке бинарный поиск.

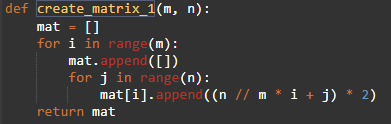




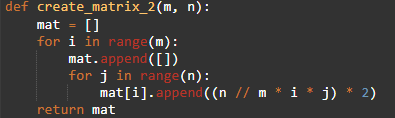
1. Условия для запуска
   1. Установить библиотеку numpy
   2. Установить библиотеку time
2. Генерация данных

Генерация данных возможна двумя способами

1. Генерация линейной матрицы

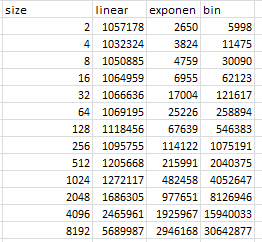


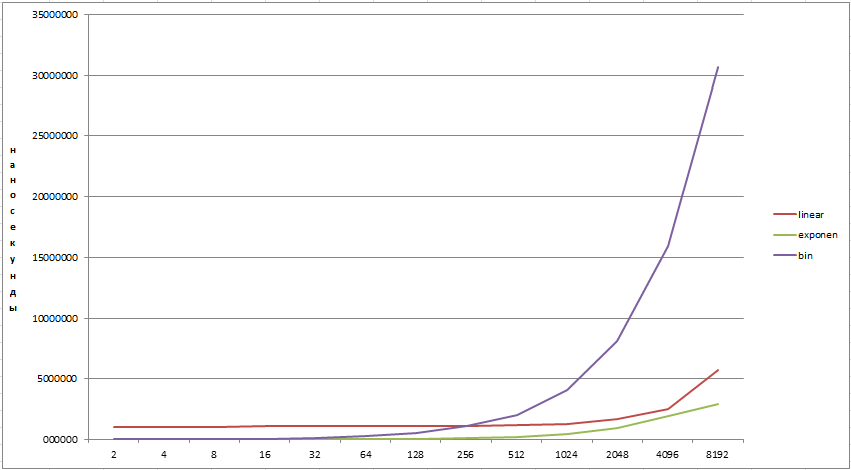
1. Генерация экспоненциальной матрицы



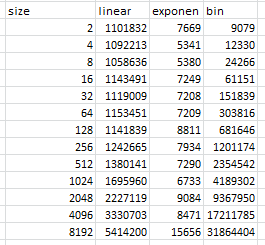
1. Результаты работы алгоритмов

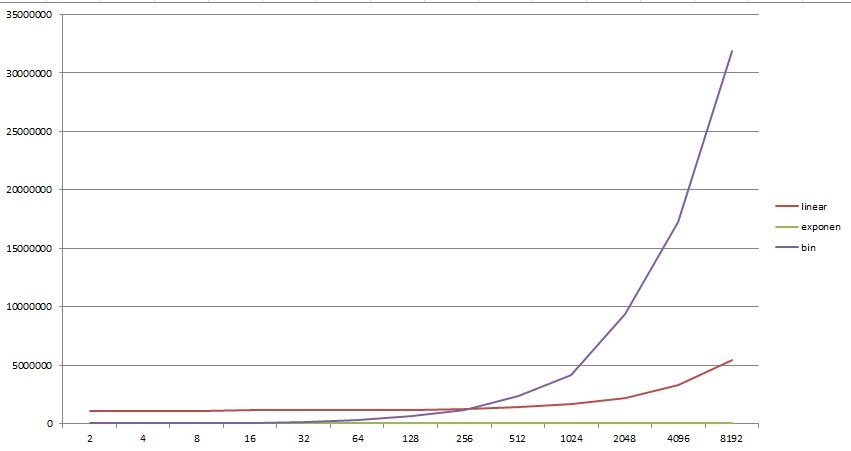
A[i][j] = (m / n \* i + j) + 1





A[i][j] = (m / n \* i \* j) + 1





Линейный поиск в данном контексте может быть хорошим выбором при работе с отсортированными матрицами, где распределение элементов позволяет использовать стратегию движения в матрице для уменьшения области поиска. Однако на больших матрицах или в случае неравномерного распределения элементов другие алгоритмы, такие как бинарный поиск или экспоненциальный поиск, могут быть более эффективными.

В данном случае, бинарный поиск является самым неоптимальным способ поиска элемента в матрице. Исходя из графика, можно заметить, что после рубежа в 256 \* 2\*\*13 элементов, время поиска резко возрастает, что связано с тем, что матрица становится более квадратной, и кол-во итераций кратно возрастает.

Экспоненциальный поиск может оказаться более эффективным на матрице, созданной с помощью create\_matrix\_2, из-за особенностей распределения элементов в этой матрице.

В create\_matrix\_2 элементы генерируются с использованием формулы: (n // m \* i \* j) \* 2. Эта формула создает матрицу с элементами, распределенными более равномерно по сравнению с формулой в create\_matrix\_1. В случае create\_matrix\_2 более равномерное распределение элементов делает процесс экспоненциального поиска более эффективным, поскольку шаг сужения чаще приводит к приближению к искомому элементу. Из-за этого на матрице заполненной create\_matrix\_2 экспоненциальный поиск работает быстрее.

Выводы:

Линейный поиск прост и эффективен для небольших матриц, где его линейная зависимость от размеров не сказывается на производительности.

Бинарный поиск эффективен для упорядоченных матриц, но может потерять эффективность при неоднородном распределении элементов.

Экспоненциальный поиск обеспечивает быстрое сужение области поиска и хорошо работает с большими матрицами, особенно при равномерном распределении элементов. Однако, также требует предварительной сортировки.