Лабораторная работа № 1

Выполнила:

студентка 21ПИ2 Болотина С. Е.

Цель:
Вычислить среднее время работы трех алгоритмов поиска в отсортированной матрице, заполненной первым способом:
a[i][j] = (N / M * I + j) * 2, где target = 2N + 1

Сравнить полученные результаты времени работы трех алгоритмов, сделать выводы.

А также вычислить время работы экспоненциального поиска в отсортированной матрице, заполненной вторым способом:
a[i][j] = (N/M*(i + 1)(j + 1)) * 2, где target = 16N + 1

Сравнить полученные результаты времени работы экспоненциального поиска на двух видах данных, сделать выводы.

Ход работы:

В лабораторной использовались:
ЯП Python 3.9
IDE PyCharm Community Edition 2021.3.2

Сложность:

Binary search: O(M * Log N)

Ladder search: O(M + N)

Exponential search: O(log N) – проход по строке и таких строк M, следовательно, O(M * Log N)

Стоит заметить, что если первый элемент матрицы(он же минимальный) больше искомого, алгоритмы завершают свою работу – сложность O(1). В дальнейшем рассматриваем случай, когда искомый элемент предположительно может содержаться в матрице.

Можно было бы предположить, что бинарный и экспоненциальный поиск работают примерно одинаково.

1. Однако по результатам видно(диаграмма 1), что с увеличением количества строк в матрице хуже всего работает бинарный поиск. В случае если искомого элемента нет в матрице, алгоритм будет зависеть только от количества столбцов(N) и строк(M). Значит, при постоянном количестве столбцов(N), а следовательно, и постоянном времени работы бинарного поиска по строкам, равного logN, время работы алгоритма будет расти в той же степени, в какой увеличивается число строк(M). Как можно видеть из таблицы(приложение 1), время работы увеличивается в два раза при увеличении M в два раза.

В лучшем случае, искомый элемент будет найдет в первой строке, тогда сложность будет составлять O(logN).

2. Время работы экспоненциального поиска сравнимо со временем работы бинарного поиска на промежутке от M = 2 до M = 256, однако, когда количество столбцов матрицы все больше

приближается к количеству строк, время выполнения бинарного поиска начинает возрастать быстрее.
На основании имеющихся данных можно сказать, что экспоненциальный поиск превосходит бинарный, когда количество строк в матрице возрастает, в том случае, если при выполнении алгоритма экспоненциального поиска рассматривается не вся строка, а её часть. Это происходит, когда искомый элемент предположительно находится ближе к концу строки, а поскольку данные отсортированы по возрастанию, в дальнейшем поиске можно будет использовать только подходящий интервал строки, что уменьшает время работы бинарного поиска по строке до Log j, где j – размер интервала.

3. Время работы алгоритма «лесенка» колеблется на том же уровне в промежутке от M = 4 до M = 2048, при минимальном количестве строк время работы несколько ниже, начиная с M = 4096, время работы начинает возрастать. Сложность этого алгоритма O(M + N) превосходит сложность бинарного и экспоненциального O(M * log N). На имеющихся начальных данных, где M <= 10, хорошо видно, что алгоритм «лесенка» работает значительно дольше.

Однако, поскольку N не меняется, следовательно, log(N) тоже, тогда начиная с какого-то M произведение M * log N будет превосходить M + N.

4. Как можно видеть на диаграмме 2, экспоненциальный поиск работает эффективнее, когда данные в имеющейся матрице заполнены вторым способом(экспоненциально), поскольку тогда разница значений между соседними элементами большая и при экспоненциальном увеличении индекса элемента быстрее находится интервал, где может находиться искомый элемент. В том случае, когда значение количество столбцов матрицы приближается к значению количества строк, время работы экспоненциального поиска на двух разных типах данных стремится к одному и тому же значению, как можно видеть из диаграммы 3 время работы во втором случае чуть менее, чем в два раза, меньше. Т.е. на экспоненциальный поиск заметно замедляется при увеличении количества столбцов матрицы

Выводы:

Экспоненциальный поиск лучше использовать с большими массивами, когда бинарный поиск затратен, поскольку экспоненциальный поиск разделяет данные на более доступные для поиска части.

Если N не меняется, следовательно, log(N) тоже, тогда начиная с какого-то M произведение M * log N будет превосходить M + N.

Экспоненциальный поиск работает эффективнее, когда данные заполнены экспоненциально, поскольку тогда разница значений между соседними элементами большая и при экспоненциальном увеличении индекса элемента быстрее находится интервал, где может находиться искомый элемент. Однако с увеличением M, которое таким образом приближается к N, время работы алгоритма сильно увеличивается.

Приложение

Диаграмма 1.

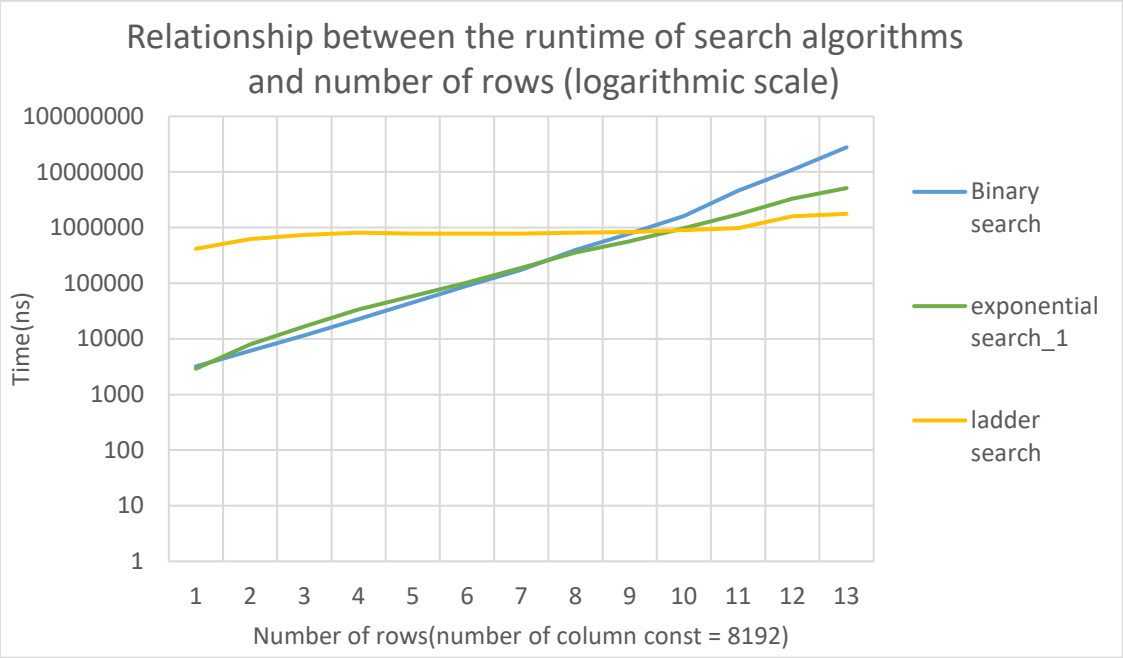


Диаграмма 2.

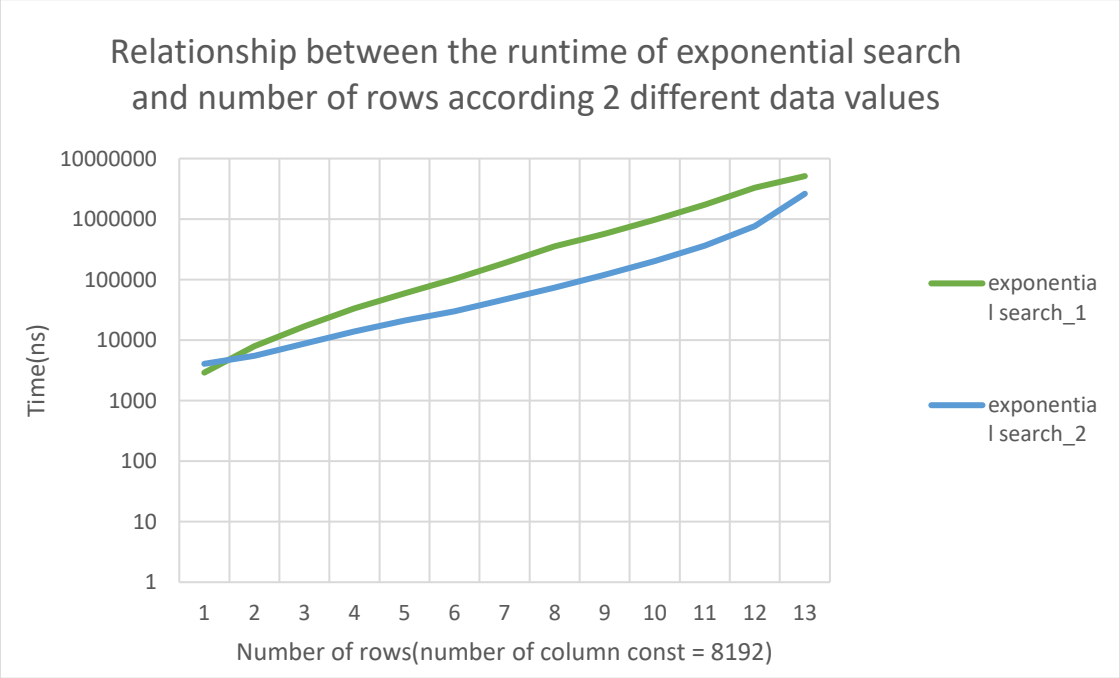
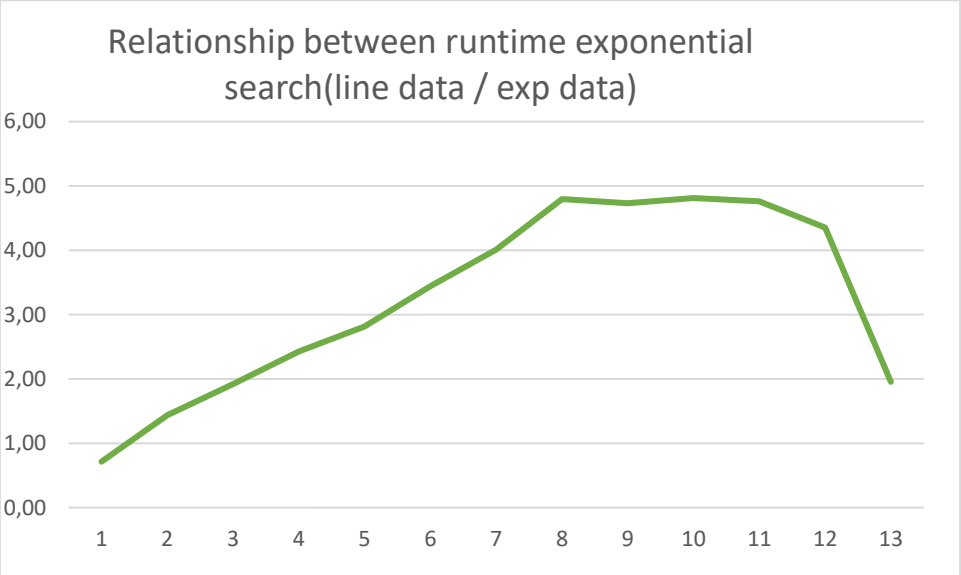


Диаграмма 3.



Приложение 1.

Binary search	
m	time(nanoseconds)
2	3196
4	6102
8	11558
16	22789
32	45039
64	89698
128	174111
256	395342
512	775402
1024	1611259
2048	4617881
4096	10911268
8192	27761831

Приложение 2.

exponential search_1	
m	time(nanoseconds)
2	2906
4	7910
8	16804
16	33586
32	58973
64	102608
128	188790
256	354518
512	569832
1024	980536
2048	1729678
4096	3314736
8192	5135877

Приложение 3.

ladder search	
m	time(nanoseconds)
2	416847
4	622337
8	743022
16	806746
32	779101
64	782278
128	781529
256	811688
512	829098
1024	903500
2048	970644
4096	1593568
8192	1769547

Приложение 4.

exponential search_2	
m	time(nanoseconds)
2	4064
4	5497
8	8746
16	13828
32	20928
64	29799
128	47078
256	73938
512	120486
1024	203753
2048	363125
4096	761361
8192	2626017

Приложение 5.

exp1 / exp2	
2	0,72
4	1,44
8	1,92
16	2,43
32	2,82
64	3,44
128	4,01
256	4,79
512	4,73
1024	4,81
2048	4,76
4096	4,35
8192	1,96