Lab 1. Алгоритмы поиска в матрице.

1. Язык

Работа была выполнена на языке программирования Java с использованием framework для работы с Excel.

2. Немного кода

Код алгоритмов представлен в Algorithms.txt файле.
2.1. Для создания матриц и тестирования алгоритмов был использован класс Matrix:

```
public class Matrix {

    14 usages
    private final int[][] matrix;
    11 usages
    private final int rows;
    8 usages
    private final int columns;
    11 usages
    private final int key;

    2 usages
    Matrix(int rows, int columns, int key) {
        this.rows = rows;
        this.columns = columns;
        this.key = key;
        matrix = new int[rows][columns];
    }
```

2.2. Генерация матриц:

```
    public void generateInTheFirstWay(){
        for(int i = 0; i < rows; ++i){
            for(int j = 0; j < columns; ++j){
                matrix[i][j] = (columns / rows* i + j) * 2;
            }
        }
    }

    1 usage
    public void generateInTheSecondWay(){
        for(int i = 0; i < rows; ++i){
            for(int j = 0; j < columns; ++j){
                matrix[i][j] = (columns / rows* (i+1) * (j+1)) * 2;
            }
        }
    }
```

2.3. Бинарный поиск:

```java
public boolean binarySearch(){
    for(int i = 0; i < rows; ++i){
        int result = binSearch(i,  right: columns-1);
        if(result != -1){
            return true;
        }
    }
    return false;
}

1 usage
private int binSearch(int row, int right){
    int left = 0;
    while (left <= right){
        int middle = (left+right)/2;
        if(matrix[row][middle] == key){
            return middle;
        }
        if(matrix[row][middle] > key){
            right = middle - 1;
        } else{
            left = middle + 1;
        }
    }
    return -1;
}
```

2.4. Поиск “Лесенкой”:

```java
public boolean ladderSearch() {
    int row = 0, column = columns - 1;
    while (row < rows && column > -1){
        if(matrix[row][column] == key){
            return true;
        }
        else if(matrix[row][column] < key){
            row++;
        } else{
            column--;
        }
    }
    return false;
}
```

2.5. Поиск “Лесенкой” с использованием экспоненциального поиска по столбцу:

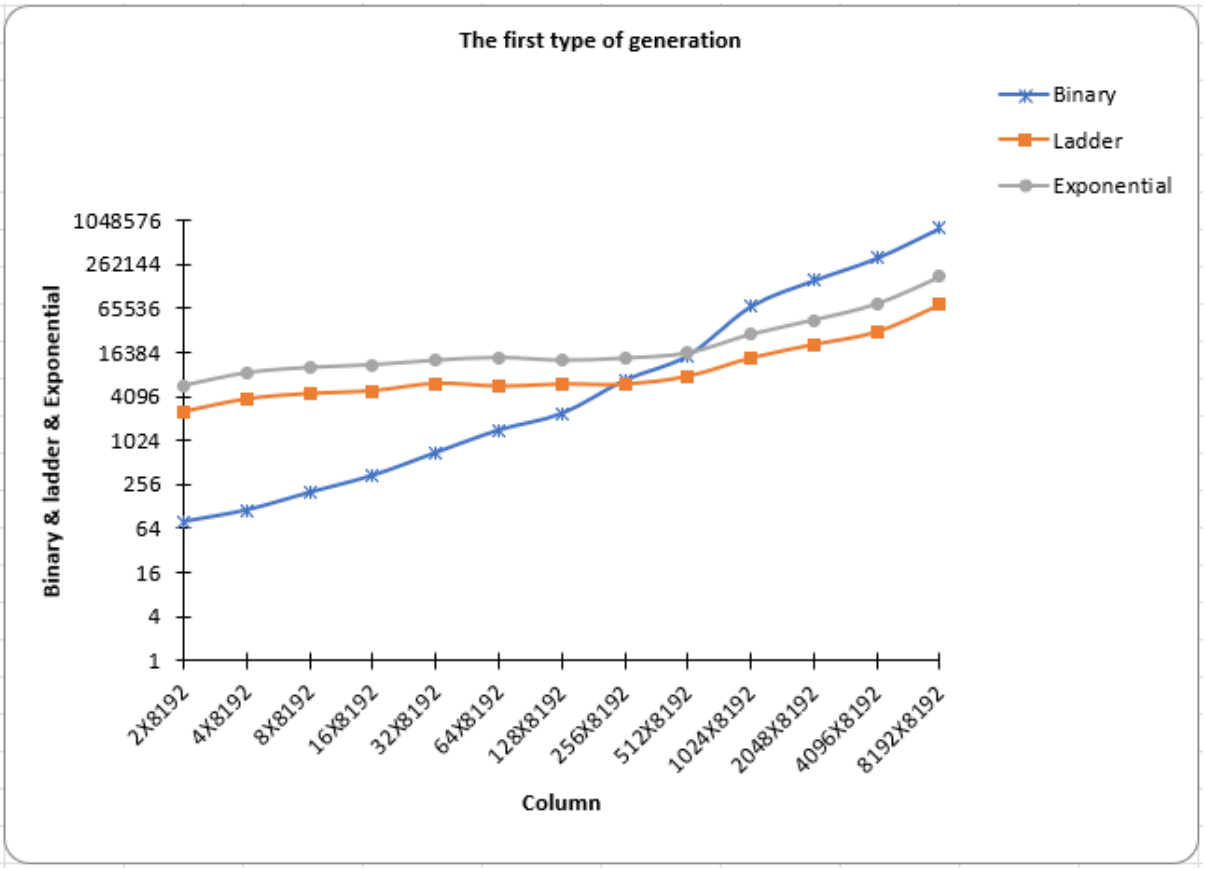
```java
public boolean ladderExpSearch(){
    int line = 0, column = columns - 1;
    while (column > -1){
        if(matrix[line][column] == key){
            return true;
        }
        else if(matrix[line][column] > key){
                column--;
        } else{
            int finish = 1;
            while ((line + finish < rows) && (matrix[line + finish][column] < key)){
                finish = finish * 2;
            }
            int start = line + finish/2;
            if(line + finish >= rows){
                finish = rows - 1;
            } else {
                finish += line;
            }
            if(start == finish){
                return false;
            }
            while (start < finish){
                int middle = (start+finish)/2;
                if(matrix[middle][column] == key){
                    return true;
                }
                if(matrix[middle][column] > key){
                    finish = middle - 1;
                } else{
                    start = middle + 1;
                }
            }
            line = start;
        }
    }
    return false;
}
```

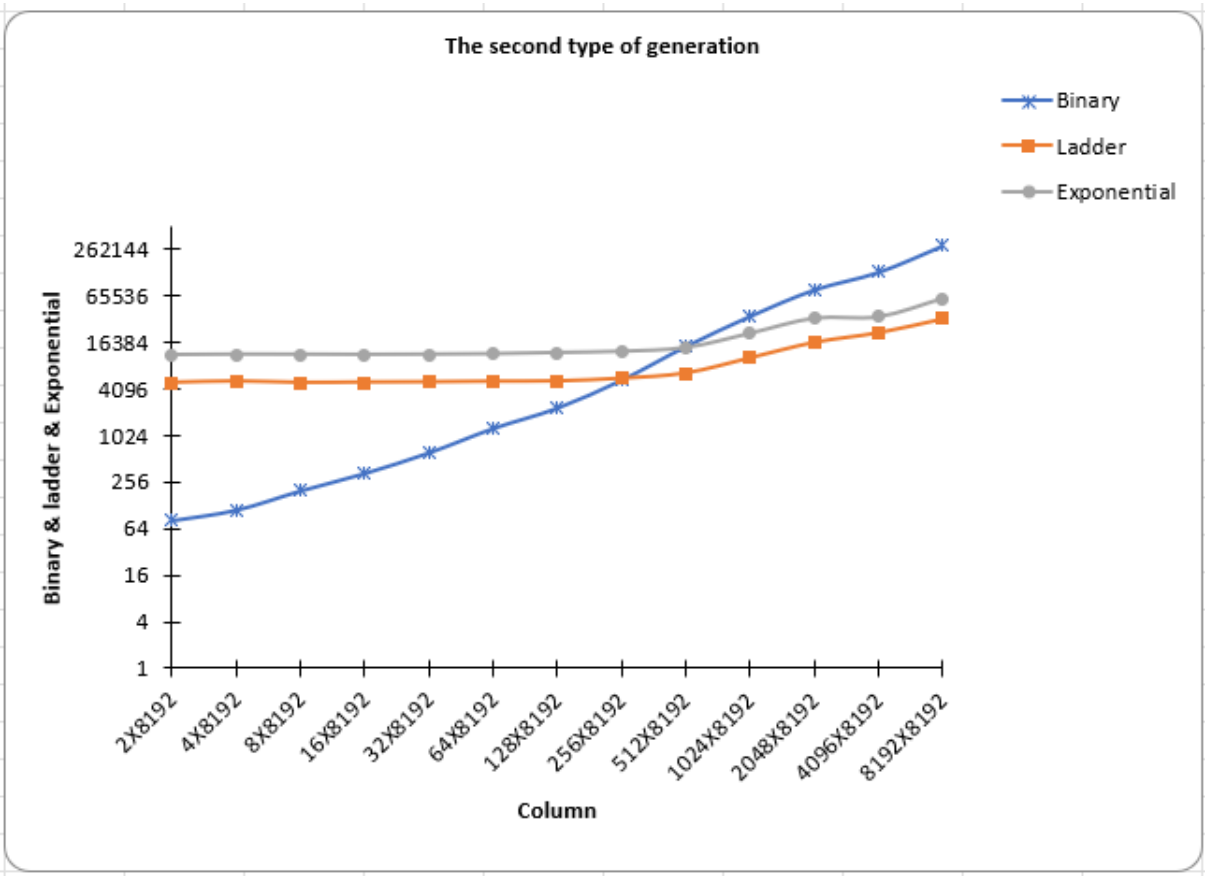
3. Визуализация в Excel

Статистика скорости работы алгоритмов автоматически заполняется в “Comparison of searches.xlsx”:

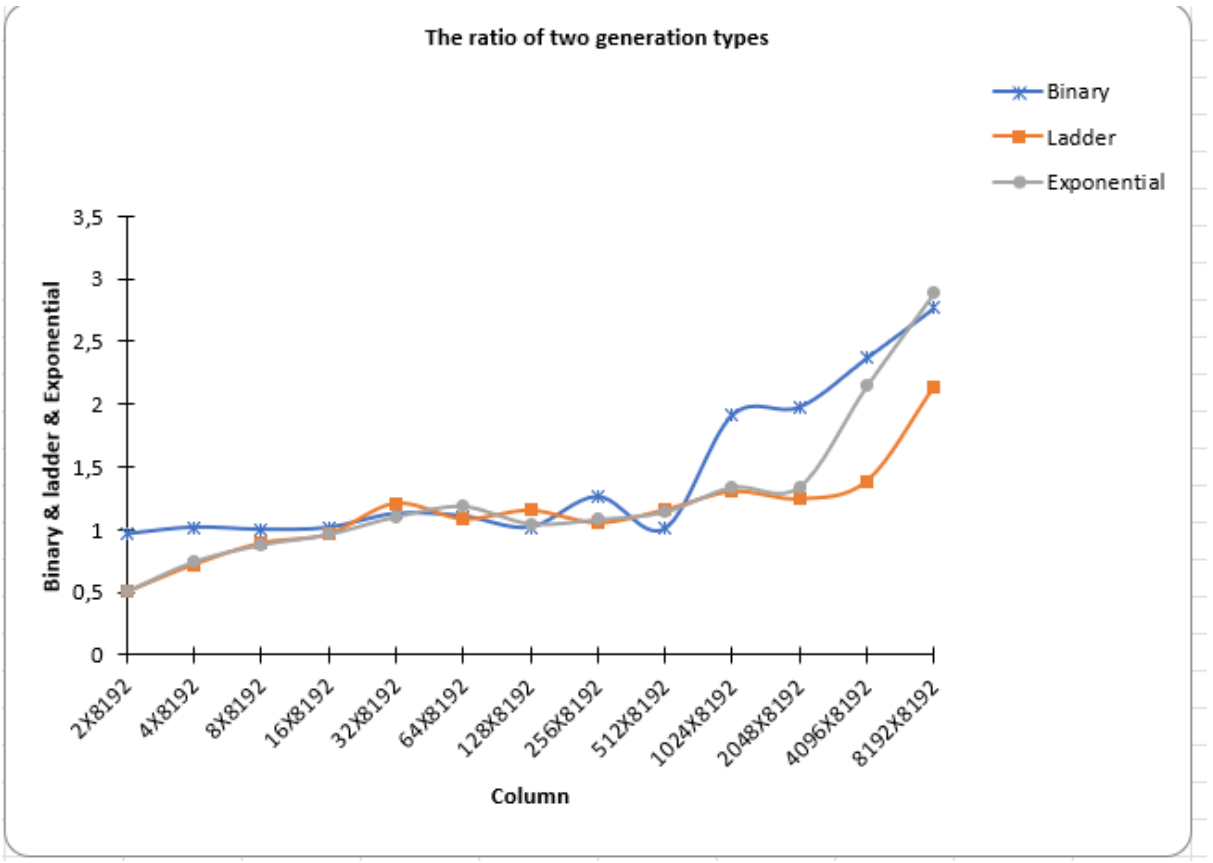
The first type of generation			
Matrix siz	Binary	Ladder	Exponential
2X8192	79	2501	5684
4X8192	114	3724	8482
8X8192	199	4409	10024
16X8192	340	4812	10898
32X8192	699	6080	12595
64X8192	1420	5550	13729
128X8192	2398	5954	12560
256X8192	6844	5906	13457
512X8192	14881	7613	15900
1024X8192	69719	13589	28678
2048X8192	158020	20778	45248
4096X8192	319992	31037	75459
8192X8192	823697	72676	176571

The second type of generation			
Matrix siz	Binary	Ladder	Exponential
2X8192	82	4943	11283
4X8192	112	5143	11419
8X8192	199	4925	11400
16X8192	335	4984	11331
32X8192	620	5022	11418
64X8192	1282	5108	11572
128X8192	2355	5156	11983
256X8192	5421	5591	12441
512X8192	14783	6566	13925
1024X8192	36405	10416	21413
2048X8192	79801	16671	33683
4096X8192	134891	22329	35145
8192X8192	297116	33954	61116





The ratio of two generation types			
Matrix siz	Binary	Ladder	Exponential
2X8192	0,963415	0,505968	0,503767
4X8192	1,017857	0,724091	0,742797
8X8192	1	0,895228	0,879298
16X8192	1,014925	0,96549	0,961786
32X8192	1,127419	1,210673	1,103083
64X8192	1,107644	1,086531	1,186398
128X8192	1,018259	1,154771	1,048152
256X8192	1,262498	1,056341	1,081665
512X8192	1,006629	1,159458	1,141831
1024X8192	1,915094	1,304627	1,33928
2048X8192	1,980176	1,246356	1,343348
4096X8192	2,372226	1,389986	2,147076
8192X8192	2,772308	2,140425	2,889113



4. Сравнение алгоритмов

Графики не столь репрезентативны, как числовые результаты тестов, они были использованы скорее как инструмент для визуализации. Для сравнения были использованы данные из трех таблиц, представленных выше.

4.1. Бинарный поиск показывает себя в разы быстрее двух остальных поисков на небольшом количестве строк (до **256**). При размере матрицы **256 x 8192** время работы поиска ‘Лесенкой’ становится равным времени работы бинарного поиска.

4.2. Начиная с размера матрицы **512 x 8192**, разница становится все более заметной, с увеличением строк бинарный поиск работает все хуже и хуже в сравнении с двумя другими алгоритмами.

4.3. При достижении максимального размера матрицы **8192 x 8192** (что было предсказуемо, т.к. **8192 * log(8912) = 107000** при бинарном поиске, а **8912 + 8192 = 16384** при поиске “Лесенкой”).

4.4. Экспоненциальный поиск работает примерно в два раза медленнее поиска “Лесенкой” на любых размерах.

4.5. Алгоритмы работают быстрее на втором типе генерации данных практически на любом размере матрицы (дольше работают только экспоненциальный и поиск “Лесенкой” на небольших матрицах).

5. Вывод

5.1. Я написал три алгоритма на Java, вывел данные измерений в Excel файл и визуализировал их с помощью графиков.

5.2. Сравнил время работы этих алгоритмов на двух типах генерации данных.

5.3. Бинарный поиск стоит использовать при небольшом количестве строк, т.к при таких данных **M*log(N) < M + N** (например: **256*log(8192) < 256 + 8192**). При этом на больших данных эффективен поиск “Лесенкой” (расчеты в **пункте 4.3**). Экспоненциальный поиск всегда проигрывает в скорости поиску ‘Лесенкой’ примерно в **2 раза** (именно при таких способах генерации данных).