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:

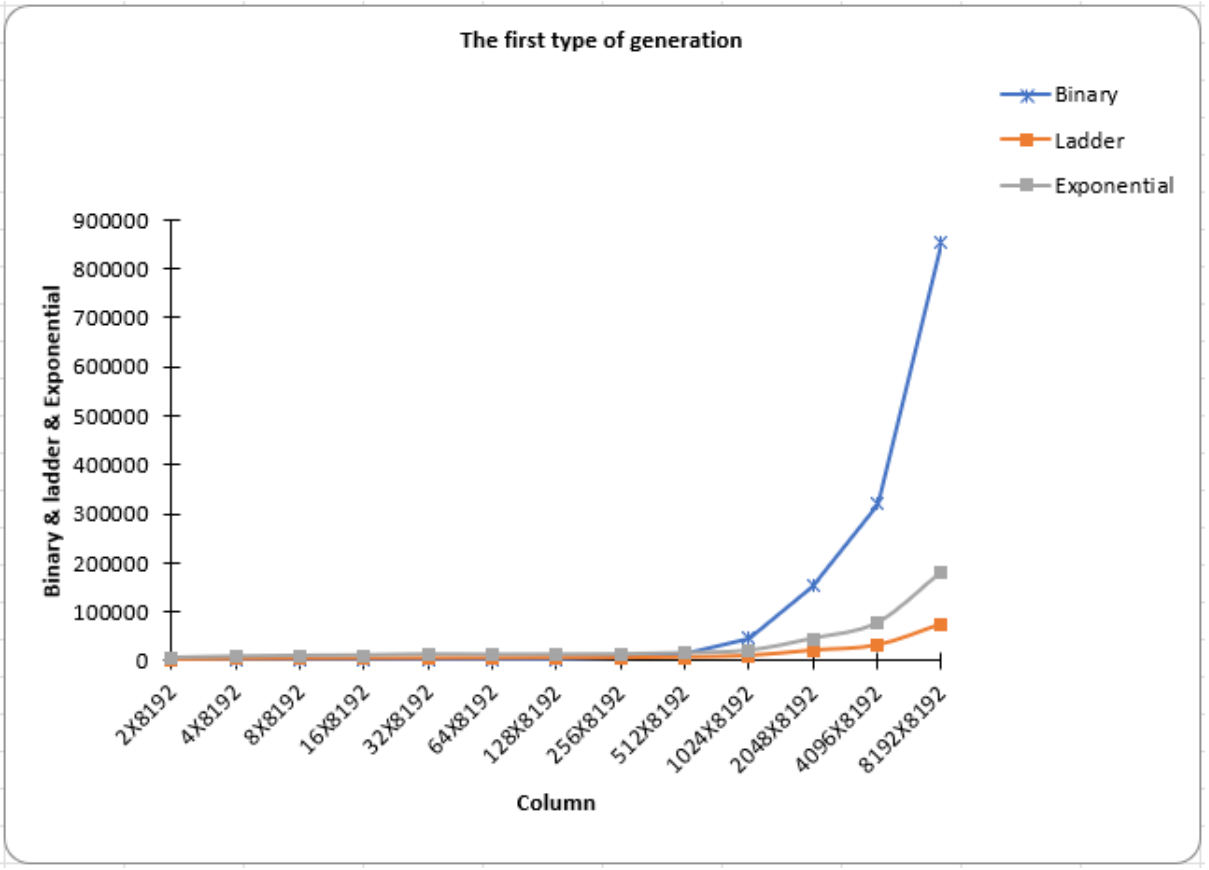
```java
public boolean ladderExpSearch(){
    int line = 0, column = columns - 1;
    while (column > -1){
        if(matrix[line][column] == key){
            return true;
        }
        else if(matrix[line][column] > key){
                column--;
        } else{
            int finish = 1;
            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;
}
```

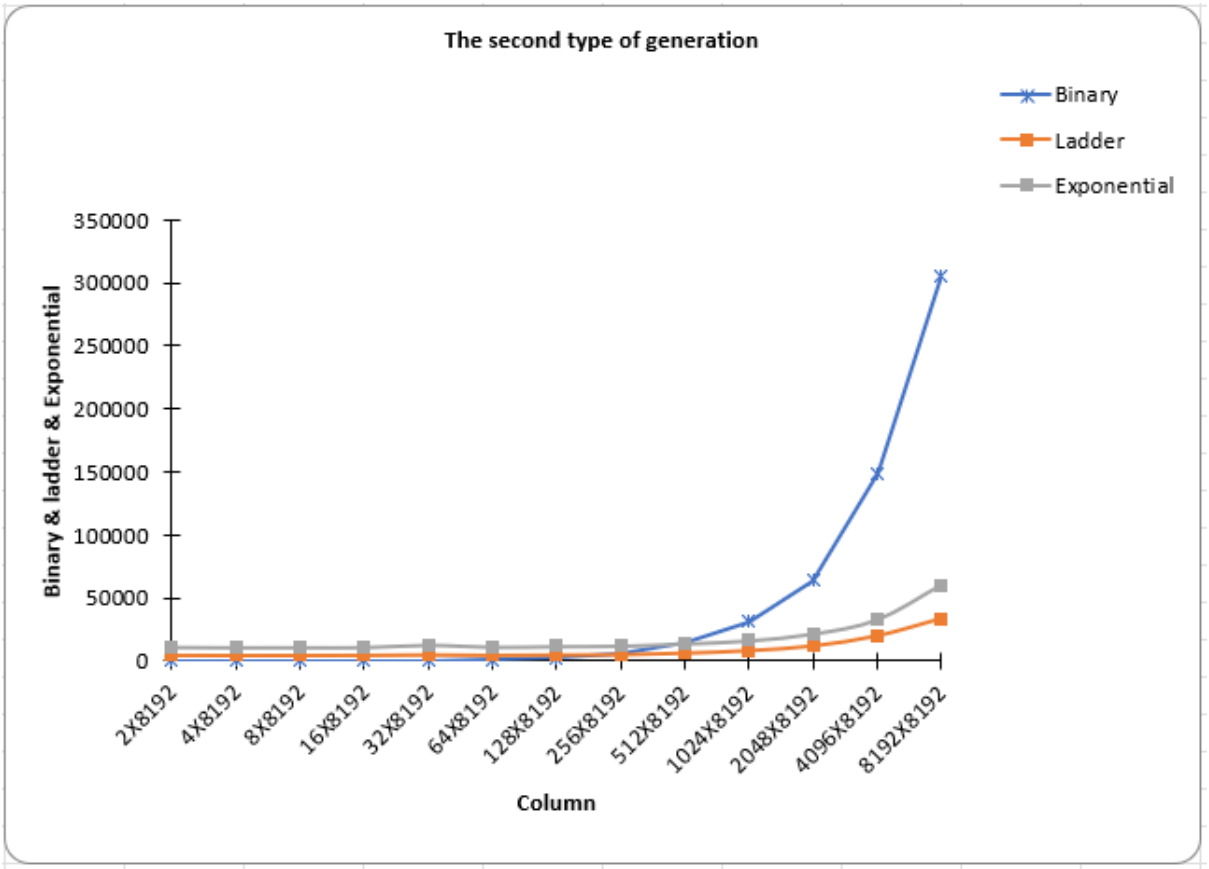
3. Визуализация в Excel

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

The first type of generation			
Matrix siz	Binary	Ladder	Exponential
2X8192	89	2504	5986
4X8192	122	3738	8488
8X8192	223	4390	9949
16X8192	339	4790	10853
32X8192	639	5056	12749
64X8192	1224	5334	12119
128X8192	2372	5884	12452
256X8192	5728	5785	12933
512X8192	15234	7332	15440
1024X8192	46361	10214	20727
2048X8192	152651	21332	45356
4096X8192	320975	31996	77477
8192X8192	853517	75465	181764

The second type of generation			
Matrix siz	Binary	Ladder	Exponential
2X8192	83	5049	11481
4X8192	116	4954	11211
8X8192	197	4974	11262
16X8192	336	5077	11347
32X8192	675	5358	13173
64X8192	1186	5046	11549
128X8192	2347	5317	12066
256X8192	5816	5546	12404
512X8192	14491	6760	14170
1024X8192	31171	8682	16511
2048X8192	63767	12292	21873
4096X8192	149187	20046	33078
8192X8192	306058	33379	60384





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 двух 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 алгоритм работает примерно в два раза медленнее поиска “Лесенкой” на любых размерах.

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).