Лабораторная работа 1. Алгоритмы поиска в матрице.

1. Программное обеспечение

- Запускалось на ноутбуке macbook pro 2017 15
- Mac os ventura
- node.js v19.0.0
- npm v8.19.2

2. Код

2.1 Реализация трех алгоритмов

```
const bin_search_on_rows = (A, target) => {
      for (const row of A) {
      let l = 0;
      let r = row.length - 1;

      while (r - l > 0) {
            let m = div(r + l,2);
            if (row[m] < target) {
                  l = m + 1;
            } else {
                  r = m;
            }
      }

      if (row[r] === target) {
            return true
      }
      }
      return false;
};

const ladder_solve = (A, target) => {
      let i = 0;
      let j = A[0].length - 1;

      while(i < A.length && j >= 0) {
      if (A[i][j] == target) {
            return true;
      } else if (A[i][j] < target) {
```

```
            i++;
      } else {
            j--;
      }
      }
      return false;
}

const ladder_exp_solve = (A, target) => {
      const N = A.length;
      const M = A[0].length;

      let i = 0;
      let j = M - 1;

      while (i < N && j >= 0) {
      if (A[i][j] == target) {
            return true;
      }
      if (A[i][j] < target) {
            i++;
      } else {
            bound = 1;
            while (j - bound >= 0 && A[i][j - bound] >= target) {
                  bound *= 2;
            }

            l = Math.max(0, j - bound - 1);
            r = j - div(bound,2);

            while (r - l > 1) {
                  m = div(l + r, 2);
                  if (A[i][m] >= target) {
                  r = m;
                  } else {
                  l = m;
                  }
            }

            if (A[i][r] == target) {
                  return true;
            } else {
                  j = r;
                  i++;
            }
      }
```

```
      }
      return false;
}
```

2.2 Реализация двух генераций

```
const generate_data_set = (M, N) => {
      const DATA_SET_NUM = process.env['DATASET'];

      let a = new Array(M);
      for (let i = 0; i < M; i++) {
      a[i] = new Array(N);
      }

      let target = 0;

      if (DATA_SET_NUM === '1') {
      for (let i = 0; i < M; i++) {
            for (let j = 0; j < N; j++) {
                  a[i][j] = (Math.floor(N/M) * i + j) * 2;
            }
      }
      target = 2*N + 1;
      } else if (DATA_SET_NUM === '2') {
      for (let i = 0; i < M; i++) {
            for (let j = 0; j < N; j++) {
                  a[i][j] = (Math.floor(N/M) * i * j) * 2;
            }
      }
      target = 16*N + 1;
      } else {
      throw new Error("DATASET env variable required. Try DATASET=1 npm
run start")
      }
      return [target, a]
}
```

2.3 Кусок кода запуска

```
const calculate_results = (x) => {
      const N = Math.pow(2, 13);
```

```
      const M = Math.pow(2, x);
      const NUMBER_OF_LAUNCHES = 95;

      process.stdout.write(M + " ");

      const [target,A] = generate_data_set(M, N);

      let start = 0;
      let measure_time = 0;
      let average_time = 0;
      for (let i = 0; i < NUMBER_OF_LAUNCHES; i++) {
      start = process.hrtime();
      bin_search_on_rows(A, target);
      measure_time = process.hrtime(start)[1];
      average_time += measure_time;
      }
      average_time /= NUMBER_OF_LAUNCHES;

      process.stdout.write(average_time + " ");

      average_time = 0;
      for (let i = 0; i < NUMBER_OF_LAUNCHES; i++) {
      start = process.hrtime();
      ladder_solve(A, target);
      measure_time = process.hrtime(start)[1];
      average_time += measure_time;
      }
      average_time /= NUMBER_OF_LAUNCHES;

      process.stdout.write(average_time + " ");

      average_time = 0;
      for (let i = 0; i < NUMBER_OF_LAUNCHES; i++) {
      start = process.hrtime();
      ladder_exp_solve(A, target);
      measure_time = process.hrtime(start)[1];
      average_time += measure_time;
      }
      average_time /= NUMBER_OF_LAUNCHES;

      process.stdout.write(average_time + "\n");
};

const main = () => {
      for (let x = 1; x < 14; x++) {
      calculate_results(x);
```

```
        }
};

main();
```

3. Результаты запусков

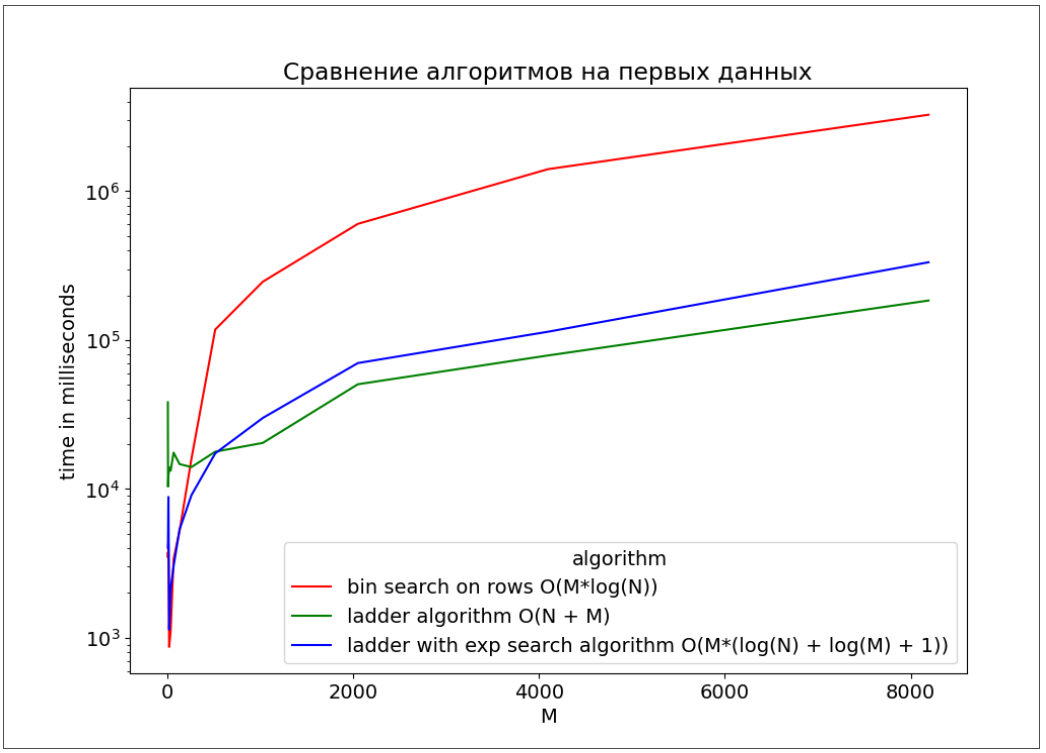
3.1. Три алгоритма на первых данных

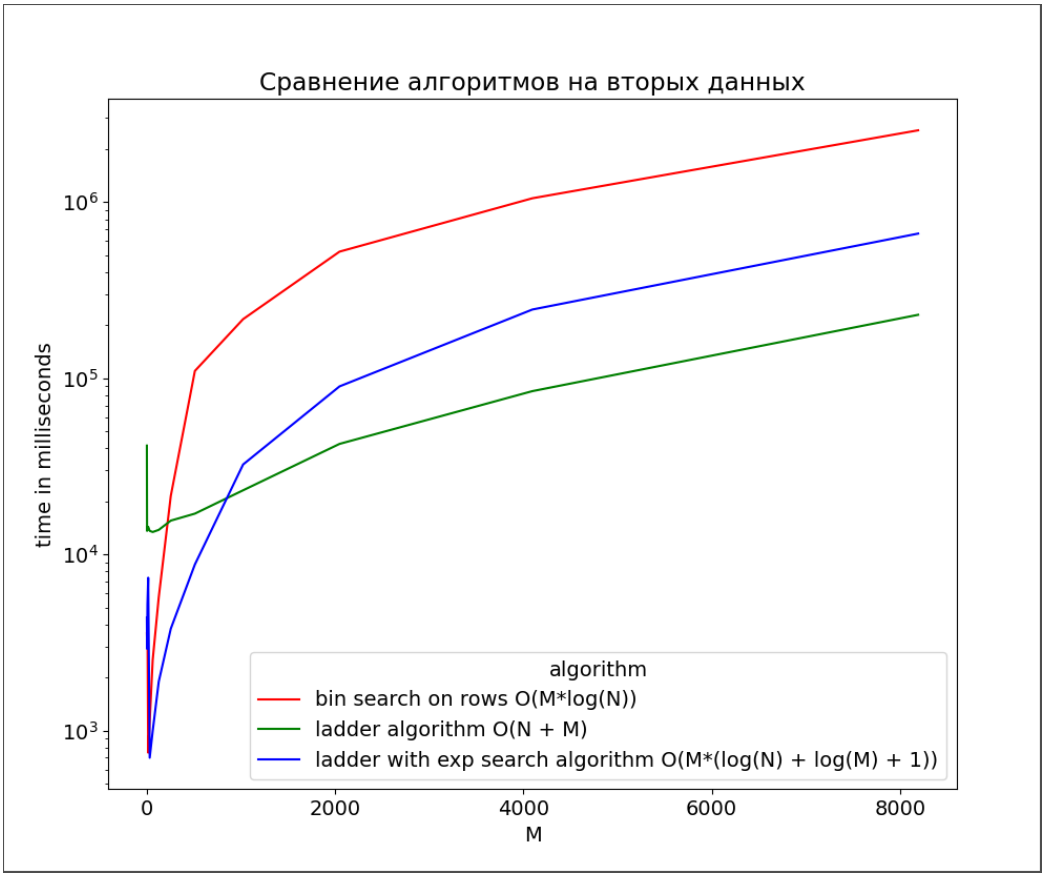
First type of generation			
Matrix size	Binary search	Ladder search	Exponentional search
2x8192	50626	319028	83899
4x8192	69593	418958	53822
8x8192	12994	410522	15319
16x8192	33924	14187	21291
32x8192	23455	14427	42942
64x8192	57700	20058	72047
128x8192	171977	33264	126201
256x8192	507537	36580	287869
512x8192	485733	39595	54023
1024x8192	491326	65064	48402
2048x8192	936493	156669	101193
4096x8192	1694306	410071	193579
8192x8192	3350651	918028	1020042

3.2. Три алгоритма на вторых данных

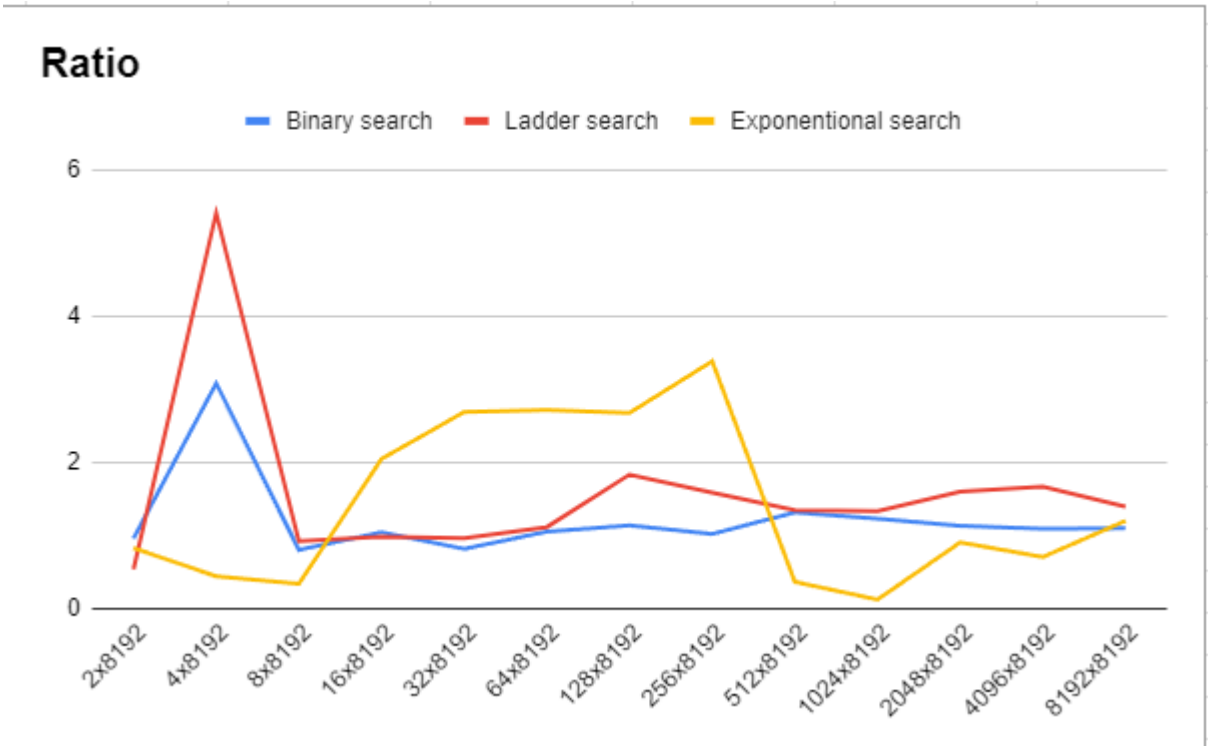
Second type of generation			
Matrix size	Binary search	Ladder search	Exponentional search
2x8192	52950	598234	101843
4x8192	22613	77467	123286
8x8192	16290	448430	45938
16x8192	32652	14517	10420
32x8192	28885	15068	15990
64x8192	55084	18064	26560
128x8192	151748	18184	47219
256x8192	499426	23114	85162
512x8192	370472	29445	149945
1024x8192	401121	49053	409466
2048x8192	830645	98095	112363
4096x8192	1558684	246938	276303
8192x8192	3061222	659931	852726

4. Визуализация





Отношение времени работы на первом типе генерации матриц ко времени работы на втором типе генерации матриц



5. Выводы

5.1 Первые два графика.

Решение бинарным поиском отстает от двух других, что и понятно из функций асимптотики данных алгоритмов.

Разберемся с решениями лесенкой и экспоненциальным на первых данных. На самом деле, при маленьких M, они не сильно отличаются друг от друга, так как самый большой элемент матрицы (A[M-1][N-1] = (N/M*(M-1) + N - 1)*2 = N*2 - N/M*2 - 2) будет меньше таргета 2N + 1, а значит наши алгоритмы пройдутся из правого верхнего угла до правого нижнего угла за линейное время. Отличие в графиках объясняется сборкой мусора на движке v8 для Javascript.

На вторых данных лесенка и экспоненциальный показали разные результаты при маленьких M. Разберемся почему. Самый большой элемент в матрице для вторых данных - это A[M-1][N-1] = N/M*(M-1)(N-1)*2 = (N - N/M)(N - 1)*2 = (N^2 - N - N^2/M + N/M) * 2. Заметим, что target = 16N + 1 будет находится внутри матрицы начиная с некоторого M. Путем несложных вычислений, получаем, что с x > 0 наш таргет будет меньше максимального элемента матрицы, а для наших данных (x лежит в отрезке [1, 13]) это выполняется всегда. Таким образом, наш экспоненциальный поиск на при

маленьких M (на графике от 2 до 1000) будет превосходить над линейным по строке. Это и видно на самом графике. А после преобладает линейный, так как экспоненциальный поиск хорош только в случае, когда данные лежат в начале поиска, что в нашем случае не валидно, так как данные при некоторых M лежат далеко внутри матрицы, что и дает фору для линейного поиска.

5.2. Разберем третий график.

На первых данных таргета просто нет в матрице (таргет больше самого максимального элемента) при маленьких M, тогда он просто идёт по правому краю (от левого верхнего угла до правого нижнего), а во втором дата сете он появляется за счёт того, что данные очень быстро растут. Так, он начинает запускать экспоненциальный поиск по строке, что замедляет его по сравнению с первыми данными, который просто идет по столбцу линейным поиском. Далее лидерство переходит экспоненциальному по вторым данным, так как в первом таргет уже находится внутри матрицы. Также во вторых данных матрица симметрична (A[i][j] == A[j][i]), поэтому, таргет будет искаться в правой половине матрицы, что значительно ускоряет его по сравнению с первым набором данных.