Министерство цифрового развития, связи и массовых коммуникаций

Ордена Трудового Красного Знамени федеральное государственное бюджетное

образовательное учреждение высшего образования

«Московский технический университет связи и информатики»

Отчёт по лабораторной работе № 1

«Методы сортировки»

по дисциплине «Системы и алгоритмы обработки данных»

Выполнил: студент группы

БВТ1905

Ахрамешин Алексей Сергеевич

Проверил:

Павликов Артем Евгеньевич

Москва

2021

Оглавление

Цель работы 3

Выполнение 4

Снимки экрана выполнения программы 9

Вывод 12

Цель работы

В ходе выполнения лабораторной работы №1 мне необходимо реализовать методы сортировок строк числовой матрицы. Оценить время работы каждого алгоритма сортировки.

Сортировки:

- Выбором
- Вставкой
- Обменом
- Шелла
- Турнирная
- Пирамидальная
- Быстрая

Выполнение

Код программы:

```
function generateMatrix(m = 5, n = 5, minLimit = -250, maxLimit = 1010){
    const matrix = new Array(m);
    for (let i = 0; i < matrix.length; i++) {
        matrix[i] = [];
        for (let j = 0; j < n; j++) {
            matrix[i][j] = minLimit + Math.floor(Math.random() * (maxLimit - minLimit
+ 1));
        }
    }
    return matrix;
}

function flattenMatrix(matrix){
    return matrix.reduce((flatArray, row) => [...flatArray, ...row], []);
}

function getMatrixFromArray(array, m, n){
    const matrix = new Array(m);
    for (let i = 0; i < matrix.length; i++) {
        matrix[i] = array.slice(i * m, (i + 1) * n);
    }
    return matrix;
}

function swap(array, first, second) {
      [array[first], array[second]] = [array[second], array[first]];
}

function nativeSort(matrix) {
    const array = flattenMatrix(matrix);

    array.sort((first, second) => second < first ? 1 : -1);

    return getMatrixFromArray(array, matrix.length, matrix[0].length)
}

function selectionSort(matrix) {
    const array = flattenMatrix(matrix);

    for (let i = 0; i < array.length; i++) {
      let min = i;

      for (let j = i + 1; j < array.length; j++) {
        if (array[j] < array[min]) {
          min = j;
        }
```

```
      }

      if (min !== i) {
        swap(array, i, min);
      }
    }

    return getMatrixFromArray(
      array,
      matrix.length,
      matrix[0].length
    );
  }

function  insertionSort(matrix) {
    const array = flattenMatrix(matrix);

    for (let i = 1; i < array.length; i++) {
      const key = array[i];
      let j = i - 1;

      while (j >= 0 && array[j] > key) {
        array[j + 1] = array[j];
        j--;
      }

      array[j + 1] = key;
    }

    return getMatrixFromArray(
      array,
      matrix.length,
      matrix[0].length
    );
  }

function  bubbleSort(matrix) {
    const array = flattenMatrix(matrix);

    for (let i = 0; i < array.length; i++) {
      for (let j = 0; j < array.length; j++) {
        if (array[j] > array[j + 1]) {
          swap(array, j, j + 1);
        }
      }
    }

    return getMatrixFromArray(
      array,
      matrix.length,
      matrix[0].length
    );
  }
```

```
function shellSort(matrix) {
    const array = flattenMatrix(matrix);

    let gap = Math.floor(array.length / 2);

    while (gap > 0) {
      for (let i = gap; i < array.length; i++) {
        const key = array[i];
        let j = i;

        while (j >= gap && array[j - gap] > key) {
          array[j] = array[j - gap];
          j -= gap;
        }

        array[j] = key;
      }

      gap = Math.floor(gap / 2);
    }

    return getMatrixFromArray(
      array,
      matrix.length,
      matrix[0].length
    );
  }


  function  quickSort(matrix) {
    const array = flattenMatrix(matrix);

    function _sort(array) {
      if (array.length < 2) return array;

      const pivot = array[0];
      const left = [];
      const right = [];

      for (let i = 1; i < array.length; i++) {
        if (pivot > array[i]) {
          left.push(array[i]);
        } else {
          right.push(array[i]);
        }
      }

      return [..._sort(left), pivot, ..._sort(right)];
    }


    return getMatrixFromArray(
```

```
      _sort(array),
      matrix.length,
      matrix[0].length
    );
  }

  function heapSort(matrix) {
    const array = flattenMatrix(matrix);
    const length = array.length;

    function _heapify(array, length, i) {
      let largest = i;
      let left = i * 2 + 1;
      let right = left + 1;

      if (left < length && array[left] > array[largest]) {
        largest = left;
      }

      if (right < length && array[right] > array[largest]) {
        largest = right;
      }

      if (largest !== i) {
        swap(array, i, largest);

        _heapify(array, length, largest);
      }
    }

    for (let i = Math.floor(length / 2 - 1); i >= 0; i--) {
      _heapify(array, length, i);
    }

    for (let k = length - 1; k >= 0; k--) {
      swap(array, 0, k);
      _heapify(array, k, 0);
    }

    return getMatrixFromArray(
      array,
      matrix.length,
      matrix[0].length
    );
  }

function compareWithNativeSort(matrix, sort) {
  console.log("Initial matrix: ", matrix);
  const start = Date.now();
  const sortedMatrix = sort(matrix);
  const end = Date.now();

  const startNative = Date.now();
```

```
  nativeSort(matrix);
  const endNative = Date.now();

  console.log("Sorted matrix: ", sortedMatrix);
  console.log(`s sort: ${end - start} ms`)
  console.log(`Native sort: ${endNative - startNative} ms`);
}


const matrix = generateMatrix();

    console.log("\n<--- Selection sort --->");
    compareWithNativeSort(matrix, selectionSort);
    console.log("\n<--- Insertion sort --->");
    compareWithNativeSort(matrix, insertionSort);
    console.log("\n<--- Bubble sort --->");
    compareWithNativeSort(matrix, bubbleSort);
    console.log("\n<--- Shell sort --->");
    compareWithNativeSort(matrix, shellSort);
    console.log("\n<--- Quick sort --->");
    compareWithNativeSort(matrix, quickSort);
    console.log("\n<--- Heap sort --->");
    compareWithNativeSort(matrix, heapSort);
```

Снимки экрана выполнения программы

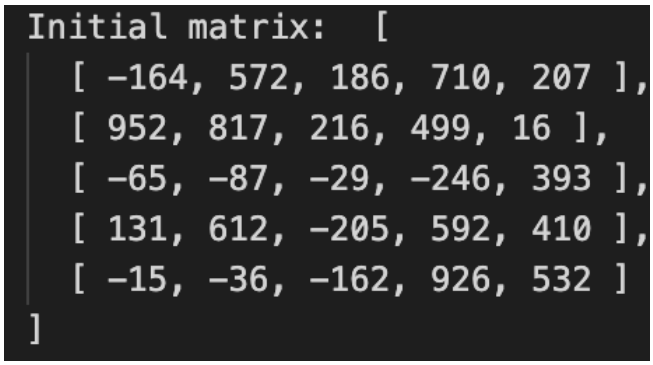

Рис. 1 – Исходная матрица

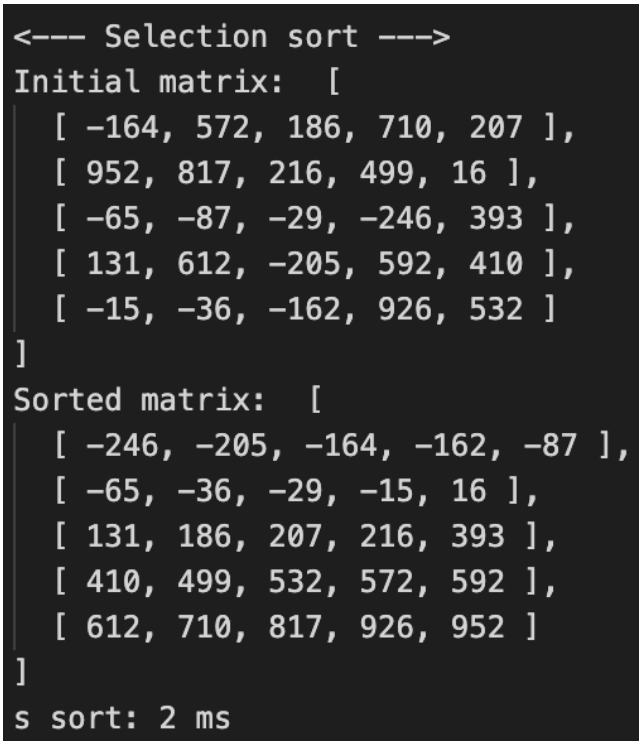

Рис. 2 – Сортировка выбором

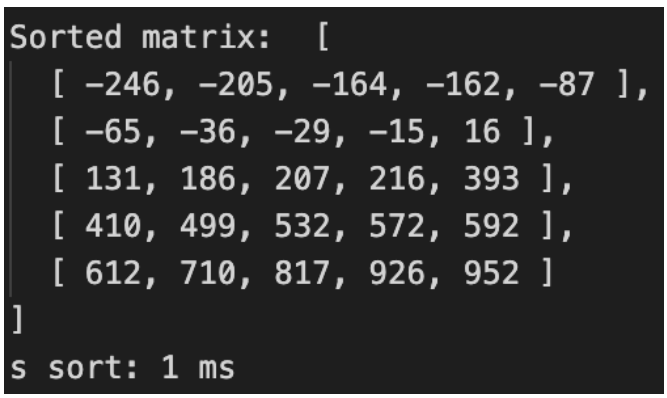

Рис. 3 – Сортировка вставкой

```
Sorted matrix:  [
  [ -246, -205, -164, -162, -87 ],
  [ -65, -36, -29, -15, 16 ],
  [ 131, 186, 207, 216, 393 ],
  [ 410, 499, 532, 572, 592 ],
  [ 612, 710, 817, 926, 952 ]
]
s sort: 1 ms
```

Рис. 4 – Сортировка обменом

```
Sorted matrix:  [
  [ -246, -205, -164, -162, -87 ],
  [ -65, -36, -29, -15, 16 ],
  [ 131, 186, 207, 216, 393 ],
  [ 410, 499, 532, 572, 592 ],
  [ 612, 710, 817, 926, 952 ]
]
s sort: 0 ms
```

Рис. 5 – Сортировка Шелла

```
Sorted matrix:  [
  [ -246, -205, -164, -162, -87 ],
  [ -65, -36, -29, -15, 16 ],
  [ 131, 186, 207, 216, 393 ],
  [ 410, 499, 532, 572, 592 ],
  [ 612, 710, 817, 926, 952 ]
]
s sort: 0 ms
```

Рис. 6 – Быстрая сортировка

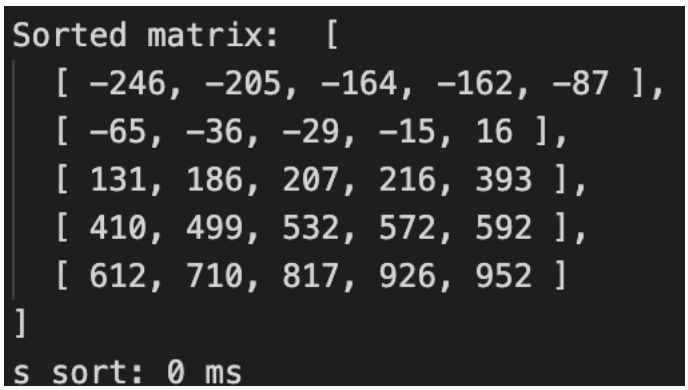

Рис. 7 – Пирамидальная сортировка

Вывод

В результате выполнения лабораторной работы №1 я реализовал алгоритмы всех заданных сортировок числовой матрицы с измерением времени их выполнения.

Результаты измерений оказались следующими:

Сортировка выбором: 2мс

Сортировка вставкой: 1мс

Сортировка обменом: 1мс

Сортировка Шелла: 0мс

Быстрая сортировка: 0мс

Пирамидальная сортировка: 0мс