МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО
ОБРАЗОВАНИЯ
«МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
УНИВЕРСИТЕТ)»

Институт № 3 «Системы управления, информатика и электроэнергетика»

Кафедра 304 «Вычислительные машины, системы и сети»

Лабороторная работа №2

по дисциплине «Алгоритмы и обработка данных»

на тему «Алгоритмы сортировки данных»

Выполнили

студенты группы М3О-225БВ-24

Егоров А.В
Федоров А.И.

Принял

Москва
2025

Задача

Для массива из ***n*** элементов выполнить сортировку по возрастанию с помощью двух указанных методов (по вариантам) для:

1. заданной произвольным образом последовательности чисел (массив один и тот же для разных сортировок);
2. уже отсортированных последовательностей в возрастающем и убывающем порядке (лучший и худший случаи для выполнения сортировки);
3. для быстрой сортировки выполнить два варианта исследования:
    - опорный элемент – последний элемент в массиве,
    - по схеме Хоара.

Этапы выполнения ЛР:

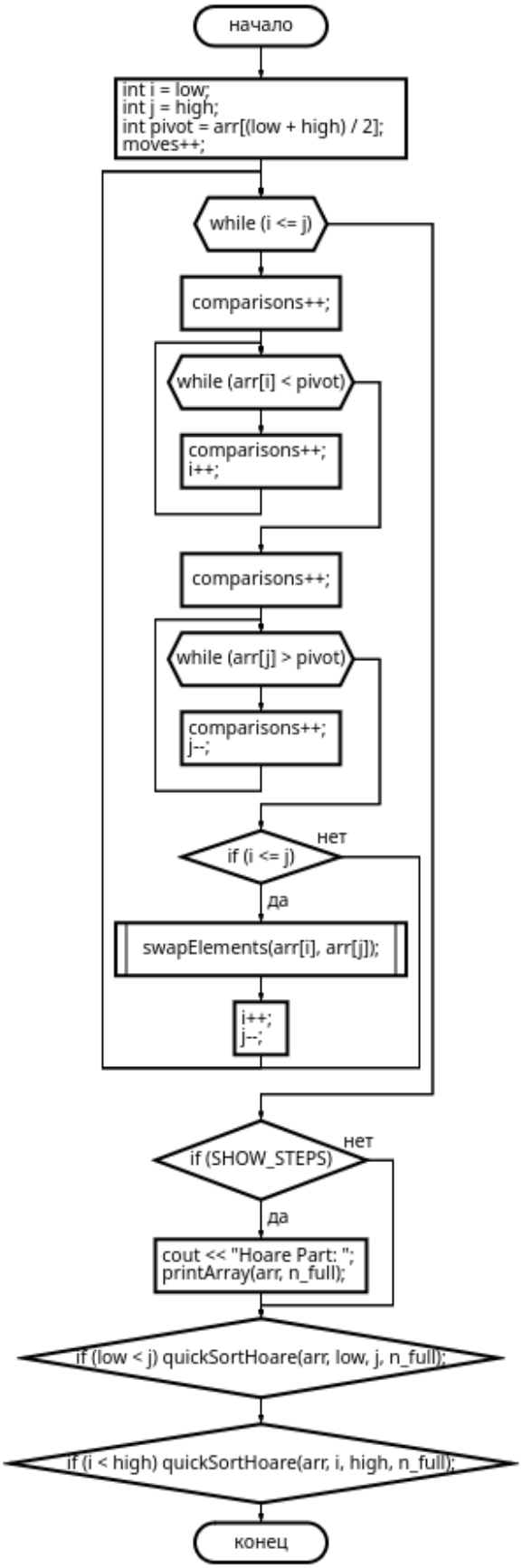
1. Промежуточные результаты сортировки представить по каждой итерации для массива размерностью ***n*** = 15 (необходимо выводить на печать весь массив на каждом промежуточном этапе сортировки). Сравнить число **необходимых сравнений и число пересылок**.
2. Выполнить сортировку массивов размерности ***n*** = 500, 1000, 10000, 50000, 100000. Сравнить **время выполнения алгоритмов, число необходимых сравнений и число пересылок**.

Варианты заданий

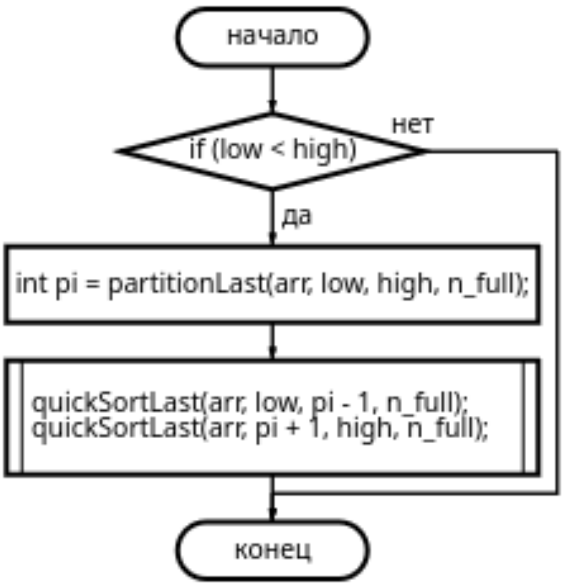
№ вар.	метод
1	выбором, быстрая
2	Шелла, быстрая
3	вставкой, быстрая
4	слиянием, быстрая

Блок-схема

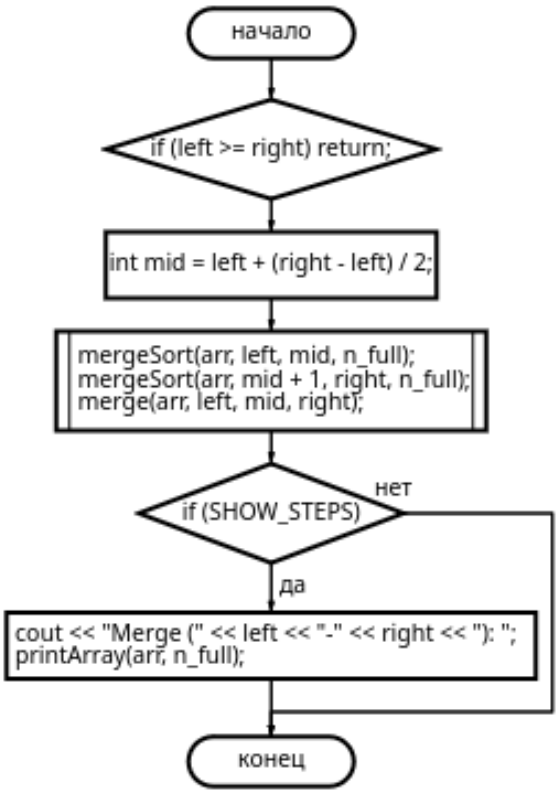
Quick Sort Hoare



Quick Sort Last



Merge Sort



Код

```
#include <iostream>
#include <iomanip>  // Для таблицы (setw)
#include <cstdlib>  // rand, srand
#include <ctime>    // time
#include <chrono>   // Для замеров времени

using namespace std;
using namespace std::chrono;

// --- Глобальные переменные для статистики ---
long long comparisons = 0; // Количество сравнений
long long moves = 0;       // Количество пересылок
bool SHOW_STEPS = false;   // Флаг вывода промежуточных шагов (для N=15)

// --- Вспомогательные функции ---

// Вывод массива
void printArray(int* arr, int n) {
    if (!SHOW_STEPS) return;
    for (int i = 0; i < n; i++) cout << arr[i] << " ";
    cout << endl;
}

// Обмен элементов (считается за 3 пересылки)
void swapElements(int& a, int& b) {
    int temp = a;
    a = b;
    b = temp;
    moves += 3;
}

// Копирование массива (чтобы сортировать одни и те же данные разными
методами)
void copyArray(int* src, int* dest, int n) {
    for (int i = 0; i < n; i++) dest[i] = src[i];
}

// Генерация данных
void generateData(int* arr, int n, int type) {
    // type: 1 - рандом, 2 - отсортирован (возр), 3 - отсортирован (убыв)
    for (int i = 0; i < n; i++) {
        if (type == 1) arr[i] = rand() % 10000;
        else if (type == 2) arr[i] = i;
        else if (type == 3) arr[i] = n - i;
    }
}

// ============================================================
// 1. СОРТИРОВКА СЛИЯНИЕМ (MERGE SORT)
// ============================================================

void merge(int* arr, int left, int mid, int right) {
    int n1 = mid - left + 1;
    int n2 = right - mid;
```

```
    // Временные массивы
    int* L = new int[n1];
    int* R = new int[n2];

    for (int i = 0; i < n1; i++) {
        L[i] = arr[left + i];
        moves++;
    }
    for (int j = 0; j < n2; j++) {
        R[j] = arr[mid + 1 + j];
        moves++;
    }

    int i = 0, j = 0, k = left;
    while (i < n1 && j < n2) {
        comparisons++;
        if (L[i] <= R[j]) {
            arr[k] = L[i];
            i++;
        } else {
            arr[k] = R[j];
            j++;
        }
        moves++; // Запись в основной массив
        k++;
    }

    while (i < n1) {
        arr[k++] = L[i++];
        moves++;
    }
    while (j < n2) {
        arr[k++] = R[j++];
        moves++;
    }

    delete[] L;
    delete[] R;
}

void mergeSort(int* arr, int left, int right, int n_full) {
    if (left >= right) return;

    int mid = left + (right - left) / 2;
    mergeSort(arr, left, mid, n_full);
    mergeSort(arr, mid + 1, right, n_full);
    merge(arr, left, mid, right);

    if (SHOW_STEPS) {
        cout << "Merge (" << left << "-" << right << "): ";
        printArray(arr, n_full);
    }
}

// ============================================================
// 2. БЫСТРАЯ СОРТИРОВКА (QUICK SORT)
```

```
// ============================================================

// ВАРИАНТ А: Опорный - ПОСЛЕДНИЙ (Lomuto partition)
int partitionLast(int* arr, int low, int high, int n_full) {
    int pivot = arr[high]; // Опорный - последний
    moves++; // чтение в pivot
    int i = (low - 1);

    for (int j = low; j <= high - 1; j++) {
        comparisons++;
        if (arr[j] < pivot) {
            i++;
            swapElements(arr[i], arr[j]);
        }
    }
    swapElements(arr[i + 1], arr[high]);

    if (SHOW_STEPS) {
        cout << "PivotLast idx=" << i+1 << ": ";
        printArray(arr, n_full);
    }
    return (i + 1);
}

void quickSortLast(int* arr, int low, int high, int n_full) {
    if (low < high) {
        int pi = partitionLast(arr, low, high, n_full);
        quickSortLast(arr, low, pi - 1, n_full);
        quickSortLast(arr, pi + 1, high, n_full);
    }
}

// ВАРИАНТ Б: Опорный - СЕРЕДИНА (Hoare partition)
// Эффективнее на отсортированных данных
void quickSortHoare(int* arr, int low, int high, int n_full) {
    int i = low;
    int j = high;
    int pivot = arr[(low + high) / 2];
    moves++; // чтение pivot

    // Цикл разделения
    while (i <= j) {
        comparisons++;
        while (arr[i] < pivot) {
            comparisons++;
            i++;
        }
        comparisons++;
        while (arr[j] > pivot) {
            comparisons++;
            j--;
        }

        if (i <= j) {
            swapElements(arr[i], arr[j]);
            i++;
            j--;
```

```
        }
    }

    if (SHOW_STEPS) {
        cout << "Hoare Part: ";
        printArray(arr, n_full);
    }

    if (low < j) quickSortHoare(arr, low, j, n_full);
    if (i < high) quickSortHoare(arr, i, high, n_full);
}

// ============================================================
// УПРАВЛЕНИЕ ТЕСТАМИ
// ============================================================

void runTest(int* original, int n, int algoType, string name) {
    int* arr = new int[n];
    copyArray(original, arr, n);

    comparisons = 0;
    moves = 0;

    // Таймер
    auto start = high_resolution_clock::now();

    if (algoType == 1) { // Merge
        mergeSort(arr, 0, n - 1, n);
    }
    else if (algoType == 2) { // Quick Last
        // Защита от переполнения стека на больших отсортированных массивах
        // В рамках лабы это можно объяснить как "недостаток алгоритма"
        if (n > 40000 && (original[0] < original[1] || original[0] >
original[1])) {
            cout << left << setw(20) << name << " | " << setw(10) <<
"SKIPPED (Stack Overflow)" << endl;
            delete[] arr;
            return;
        }
        quickSortLast(arr, 0, n - 1, n);
    }
    else if (algoType == 3) { // Quick Hoare
        quickSortHoare(arr, 0, n - 1, n);
    }

    auto end = high_resolution_clock::now();
    long long time_ns = duration_cast<nanoseconds>(end - start).count();

    if (!SHOW_STEPS) {
        cout << left << setw(20) << name << " | "
             << setw(10) << time_ns / 1000 << " mks | " // в микросекундах
для удобства
             << setw(15) << comparisons << " | "
             << setw(15) << moves << endl;
    }

    delete[] arr;
```

```
}

int main() {
    setlocale(LC_ALL, "Russian");
    srand(time(0));

    // --- ЧАСТЬ 1: Демонстрация (N = 15) ---
    cout << "=== ЭТАП 1: Визуализация (N=15) ===" << endl;
    int n_demo = 15;
    int* arr_demo = new int[n_demo];
    generateData(arr_demo, n_demo, 1); // Рандом

    cout << "Исходный массив: ";
    SHOW_STEPS = true;
    printArray(arr_demo, n_demo);
    cout << endl;

    cout << "--- Сортировка Слиянием ---" << endl;
    runTest(arr_demo, n_demo, 1, "Merge");

    cout << "\n--- Quick Sort (Last Pivot) ---" << endl;
    runTest(arr_demo, n_demo, 2, "QuickLast");

    cout << "\n--- Quick Sort (Hoare) ---" << endl;
    runTest(arr_demo, n_demo, 3, "QuickHoare");

    delete[] arr_demo;
    SHOW_STEPS = false;
    cout << endl << endl;

    // --- ЧАСТЬ 2: Большие массивы ---
    cout << "=== ЭТАП 2: Анализ производительности ===" << endl;
    int sizes[] = {500, 1000, 10000, 50000, 100000};

    for (int i = 0; i < 5; i++) {
        int n = sizes[i];
        cout << string(80, '=') << endl;
        cout << "Размер массива N = " << n << endl;
        cout << string(80, '-') << endl;
        cout << left << setw(20) << "Алгоритм" << " | "
             << setw(14) << "Время (мкс)" << " | "
             << setw(15) << "Сравнения" << " | "
             << setw(15) << "Пересылки" << endl;
        cout << string(80, '-') << endl;

        int* arr_src = new int[n];

        // 1. Случайный массив
        cout << "[ Случайные данные ]" << endl;
        generateData(arr_src, n, 1);
        runTest(arr_src, n, 1, "Слиянием");
        runTest(arr_src, n, 2, "Quick (Last)");
        runTest(arr_src, n, 3, "Quick (Hoare)");

        // 2. Отсортированный (Best/Worst case)
        cout << "\n[ Отсортированы (возр) ]" << endl;
        generateData(arr_src, n, 2);
```

```
        runTest(arr_src, n, 1, "Слиянием");
        runTest(arr_src, n, 2, "Quick (Last)"); // Внимание: тут будет плохо
        runTest(arr_src, n, 3, "Quick (Hoare)");

        // 3. Обратно отсортированный
        cout << "\n[ Отсортированы (убыв) ]" << endl;
        generateData(arr_src, n, 3);
        runTest(arr_src, n, 1, "Слиянием");
        runTest(arr_src, n, 2, "Quick (Last)"); // И тут будет плохо
        runTest(arr_src, n, 3, "Quick (Hoare)");

        delete[] arr_src;
        cout << endl;
    }

    return 0;
}
```

Тесты программы

```
❯ ./a.out
=== ЭТАП 1: Визуализация (N=15) ===
Исходный массив: 989 5431 7397 7115 5073 1907 9952 1761 2540 8757 6346 9038 8837 5577 5257

--- Сортировка Слиянием ---
Merge (0-1): 989 5431 7397 7115 5073 1907 9952 1761 2540 8757 6346 9038 8837 5577 5257
Merge (2-3): 989 5431 7115 7397 5073 1907 9952 1761 2540 8757 6346 9038 8837 5577 5257
Merge (0-3): 989 5431 7115 7397 5073 1907 9952 1761 2540 8757 6346 9038 8837 5577 5257
Merge (4-5): 989 5431 7115 7397 1907 5073 9952 1761 2540 8757 6346 9038 8837 5577 5257
Merge (6-7): 989 5431 7115 7397 1907 5073 1761 9952 2540 8757 6346 9038 8837 5577 5257
Merge (4-7): 989 5431 7115 7397 1761 1907 5073 9952 2540 8757 6346 9038 8837 5577 5257
Merge (0-7): 989 1761 1907 5073 5431 7115 7397 9952 2540 8757 6346 9038 8837 5577 5257
Merge (8-9): 989 1761 1907 5073 5431 7115 7397 9952 2540 8757 6346 9038 8837 5577 5257
Merge (10-11): 989 1761 1907 5073 5431 7115 7397 9952 2540 8757 6346 9038 8837 5577 5257
Merge (8-11): 989 1761 1907 5073 5431 7115 7397 9952 2540 6346 8757 9038 8837 5577 5257
Merge (12-13): 989 1761 1907 5073 5431 7115 7397 9952 2540 6346 8757 9038 5577 8837 5257
Merge (12-14): 989 1761 1907 5073 5431 7115 7397 9952 2540 6346 8757 9038 5257 5577 8837
Merge (8-14): 989 1761 1907 5073 5431 7115 7397 9952 2540 5257 5577 6346 8757 8837 9038
Merge (0-14): 989 1761 1907 2540 5073 5257 5431 5577 6346 7115 7397 8757 8837 9038 9952

--- Quick Sort (Last Pivot) ---
PivotLast idx=5: 989 5073 1907 1761 2540 5257 9952 7115 5431 8757 6346 9038 8837 5577 7397
PivotLast idx=3: 989 1907 1761 2540 5073 5257 9952 7115 5431 8757 6346 9038 8837 5577 7397
PivotLast idx=1: 989 1761 1907 2540 5073 5257 9952 7115 5431 8757 6346 9038 8837 5577 7397
PivotLast idx=10: 989 1761 1907 2540 5073 5257 7115 5431 6346 5577 7397 9038 8837 8757 9952
PivotLast idx=7: 989 1761 1907 2540 5073 5257 5431 5577 6346 7115 7397 9038 8837 8757 9952
PivotLast idx=9: 989 1761 1907 2540 5073 5257 5431 5577 6346 7115 7397 9038 8837 8757 9952
PivotLast idx=14: 989 1761 1907 2540 5073 5257 5431 5577 6346 7115 7397 9038 8837 8757 9952
PivotLast idx=11: 989 1761 1907 2540 5073 5257 5431 5577 6346 7115 7397 8757 8837 9038 9952
PivotLast idx=13: 989 1761 1907 2540 5073 5257 5431 5577 6346 7115 7397 8757 8837 9038 9952

--- Quick Sort (Hoare) ---
Hoare Part: 989 1761 7397 7115 5073 1907 9952 5431 2540 8757 6346 9038 8837 5577 5257
Hoare Part: 989 1761 7397 7115 5073 1907 9952 5431 2540 8757 6346 9038 8837 5577 5257
Hoare Part: 989 1761 2540 1907 5073 7115 9952 5431 7397 8757 6346 9038 8837 5577 5257
Hoare Part: 989 1761 1907 2540 5073 7115 9952 5431 7397 8757 6346 9038 8837 5577 5257
Hoare Part: 989 1761 1907 2540 5073 7115 5257 5431 7397 5577 6346 9038 8837 8757 9952
Hoare Part: 989 1761 1907 2540 5073 5431 5257 7115 7397 5577 6346 9038 8837 8757 9952
Hoare Part: 989 1761 1907 2540 5073 5257 5431 7115 7397 5577 6346 9038 8837 8757 9952
Hoare Part: 989 1761 1907 2540 5073 5257 5431 7115 7397 5577 6346 9038 8837 8757 9952
Hoare Part: 989 1761 1907 2540 5073 5257 5431 7115 6346 5577 7397 9038 8837 8757 9952
Hoare Part: 989 1761 1907 2540 5073 5257 5431 5577 6346 7115 7397 9038 8837 8757 9952
Hoare Part: 989 1761 1907 2540 5073 5257 5431 5577 6346 7115 7397 8757 8837 9038 9952
Hoare Part: 989 1761 1907 2540 5073 5257 5431 5577 6346 7115 7397 8757 8837 9038 9952
```

```
=== ЭТАП 2: Анализ производительности ===
================================================================================
Размер массива N = 500
--------------------------------------------------------------------------------
Алгоритм     | Время (мкс) | Сравнения | Пересылки
--------------------------------------------------------------------------------
[ Случайные данные ]
Слиянием     | 46          mks | 3874            | 8976
Quick (Last)         | 30          mks | 5375            | 10124
Quick (Hoare)        | 32          mks | 6708            | 3935

[ Отсортированы (возр) ]
Слиянием     | 27          mks | 2272            | 8976
Quick (Last)         | 486         mks | 124750          | 376246
Quick (Hoare)        | 7           mks | 4008            | 1020

[ Отсортированы (убыв) ]
Слиянием     | 27          mks | 2216            | 8976
Quick (Last)         | 309         mks | 124750          | 188746
Quick (Hoare)        | 7           mks | 4014            | 1767

================================================================================
Размер массива N = 1000
--------------------------------------------------------------------------------
Алгоритм     | Время (мкс) | Сравнения | Пересылки
--------------------------------------------------------------------------------
[ Случайные данные ]
Слиянием     | 97          mks | 8718            | 19952
Quick (Last)         | 66          mks | 11712           | 18173
Quick (Hoare)        | 64          mks | 12756           | 8806

[ Отсортированы (возр) ]
Слиянием     | 55          mks | 5044            | 19952
Quick (Last)         | 1849        mks | 499500          | 1502496
Quick (Hoare)        | 13          mks | 9009            | 2044

[ Отсортированы (убыв) ]
Слиянием     | 55          mks | 4932            | 19952
Quick (Last)         | 1212        mks | 499500          | 752496
Quick (Hoare)        | 14          mks | 9016            | 3541
```

```
================================================================================
Размер массива N = 10000
--------------------------------------------------------------------------------
Алгоритм     | Время (мкс) | Сравнения | Пересылки
--------------------------------------------------------------------------------
[ Случайные данные ]
Слиянием     | 1196        mks | 120368          | 267232
Quick (Last)         | 802         mks | 156296          | 238222
Quick (Hoare)        | 758         mks | 181847          | 110610

[ Отсортированы (возр) ]
Слиянием     | 646         mks | 69008           | 267232
Quick (Last)         | 186325      mks | 49995000        | 150024996
Quick (Hoare)        | 157         mks | 125439          | 23616

[ Отсортированы (убыв) ]
Слиянием     | 640         mks | 64608           | 267232
Quick (Last)         | 121288      mks | 49995000        | 75024996
Quick (Hoare)        | 164         mks | 125452          | 38617

================================================================================
Размер массива N = 50000
--------------------------------------------------------------------------------
Алгоритм     | Время (мкс) | Сравнения | Пересылки
--------------------------------------------------------------------------------
[ Случайные данные ]
Слиянием     | 6768        mks | 718015          | 1568928
Quick (Last)         | SKIPPED (Stack Overflow)
Quick (Hoare)        | 4179        mks | 991147          | 676864

[ Отсортированы (возр) ]
Слиянием     | 3700        mks | 401952          | 1568928
Quick (Last)         | SKIPPED (Stack Overflow)
Quick (Hoare)        | 933         mks | 750015          | 131068

[ Отсортированы (убыв) ]
Слиянием     | 3709        mks | 382512          | 1568928
Quick (Last)         | SKIPPED (Stack Overflow)
Quick (Hoare)        | 961         mks | 750028          | 206065
```

```
================================================================================
Размер массива N = 100000
--------------------------------------------------------------------------------
Алгоритм     | Время (мкс) | Сравнения | Пересылки
--------------------------------------------------------------------------------
[ Случайные данные ]
Слиянием     | 14134       mks | 1536318         | 3337856
Quick (Last)         | SKIPPED (Stack Overflow)
Quick (Hoare)        | 8452        mks | 2049996         | 1474378

[ Отсортированы (возр) ]
Слиянием     | 7901        mks | 853904          | 3337856
Quick (Last)         | SKIPPED (Stack Overflow)
Quick (Hoare)        | 1837        mks | 1600016         | 262140

[ Отсортированы (убыв) ]
Слиянием     | 7751        mks | 815024          | 3337856
Quick (Last)         | SKIPPED (Stack Overflow)
Quick (Hoare)        | 1996        mks | 1600030         | 412137
```

Вывод

В ходе лабораторной работы мы реализовали и протестировали два метода сортировки: слиянием и быструю (в двух вариантах). Сравнивали их по времени, количеству сравнений и пересылок на массивах разного размера.