ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»

# Отчёт по лабораторной работе № 5

# «Сортировки»

Выполнил работу

Тищенко Павел

Академическая группа J3112

Принято

Лектор, Ходненко Иван Владимирович

Санкт-Петербург

2024

## 1. Введение

**Цель работы:**

Необходимо реализовать 3 алгоритма сортировки разной сложности, проанализировать их на различных наборах данных

**Задачи:**

Изучение теоретических основ задач сортировки

Реализация алгоритма на языке программирования C++

Проведение тестов работы алгоритма с различными размерами наборов данных для оценки эффективности предложенного алгоритма

Анализ результатов

## 2. Теоретическая подготовка

Для реализации различных методов сортировки требуется осознание работы каждого алгоритма (например “ручками” на маленьких наборах данных). Далее для каждого алгоритма пишется код с использованием определенных типов данных, циклов.

Типы данных

- vector: используется для хранения динамических массивов. Позволяет изменять размер в процессе выполнения программы.

-стандартные типы данных: int, bool и тд.

## 3. Реализация

1. Осознание

Необходимо было понять, что от меня требуют, в чем заключается сложность данной мне задачи и что необходимо сделать для ее выполнения.

2. Выбор алгоритмов сортировки соответствующих требованиям лабораторной работы.

Мне хотелось не только реализовать 3 алгоритма сортировки соответствующие условию задания, но и узнать о некоторых новых для меня необычных алгоритмах сортировки

3. Реализация алгоритмов

После выбора методов сортировки, я перешел к их реализации, за счет своей специфики, для решения данной задачи мне не нужно было изучать новых типов данных или методов в c++, главной задачей было осознать сами алгоритмы на своем “псевдокоде” в голове, после чего переписать их на язык c++ не составило особого труда.

4. Тестирование

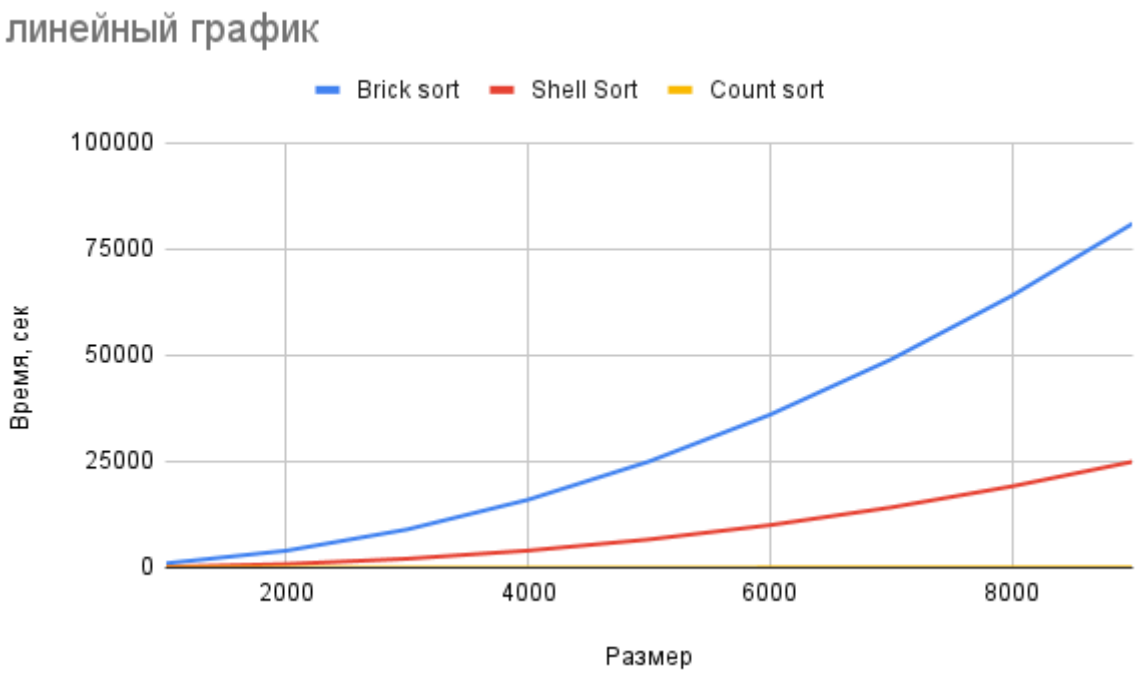
Для проверки правильности работы алгоритма были созданы численные датасеты, на которых производилось тестирование правильности алгоритмов и их скорости.

## 4. Экспериментальная часть

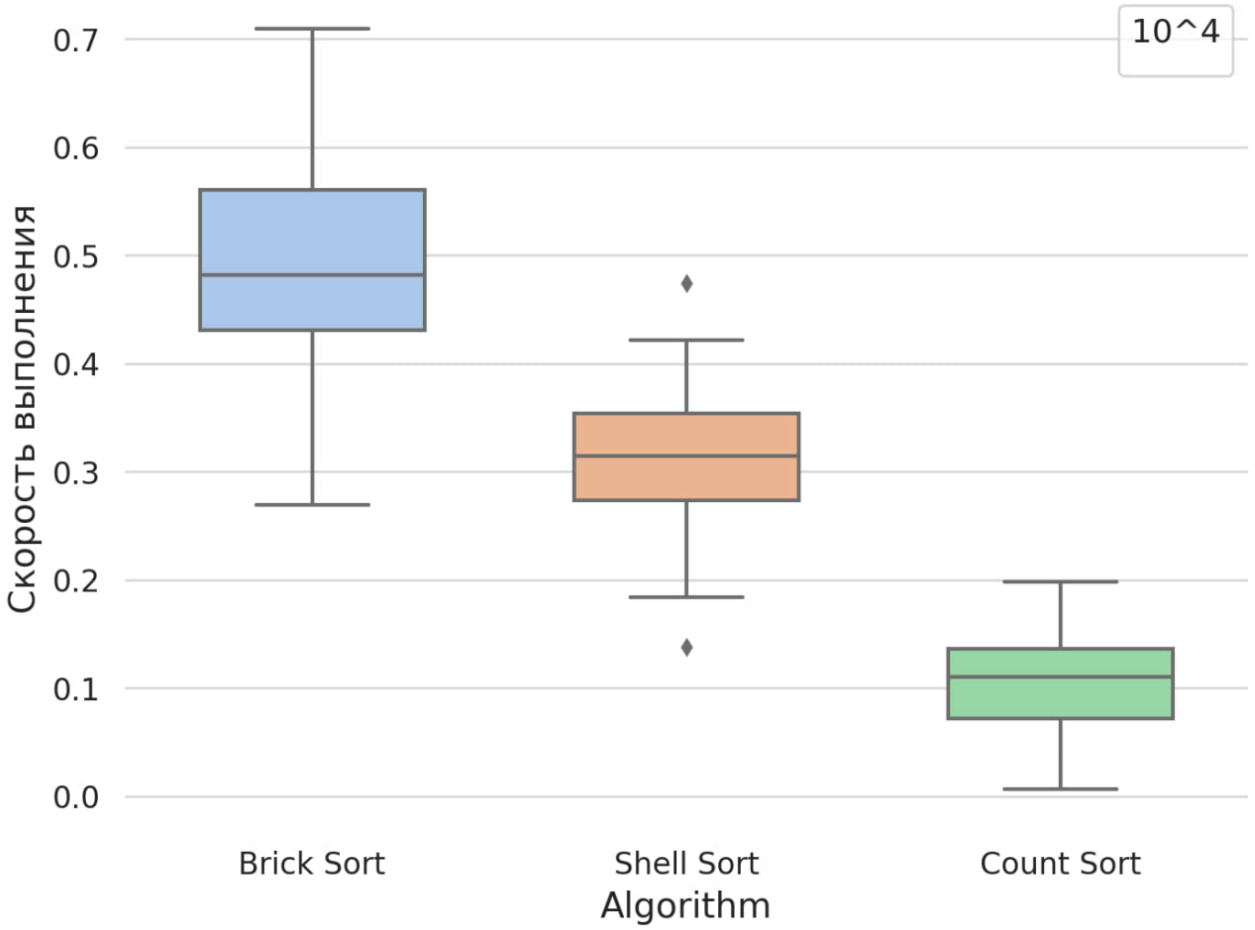
“Подсчет памяти и асимптотики алгоритмов” Таблица 1

| алгоритм | сложность | пространственная сложность |
|---|---|---|
| brickSort | O(N^2) | O(1) |
| shellSort | <O(N^2) (ср. скорость O(N*log N)) | O(N) |
| countingSort | O(N*k) | O(N*k) |

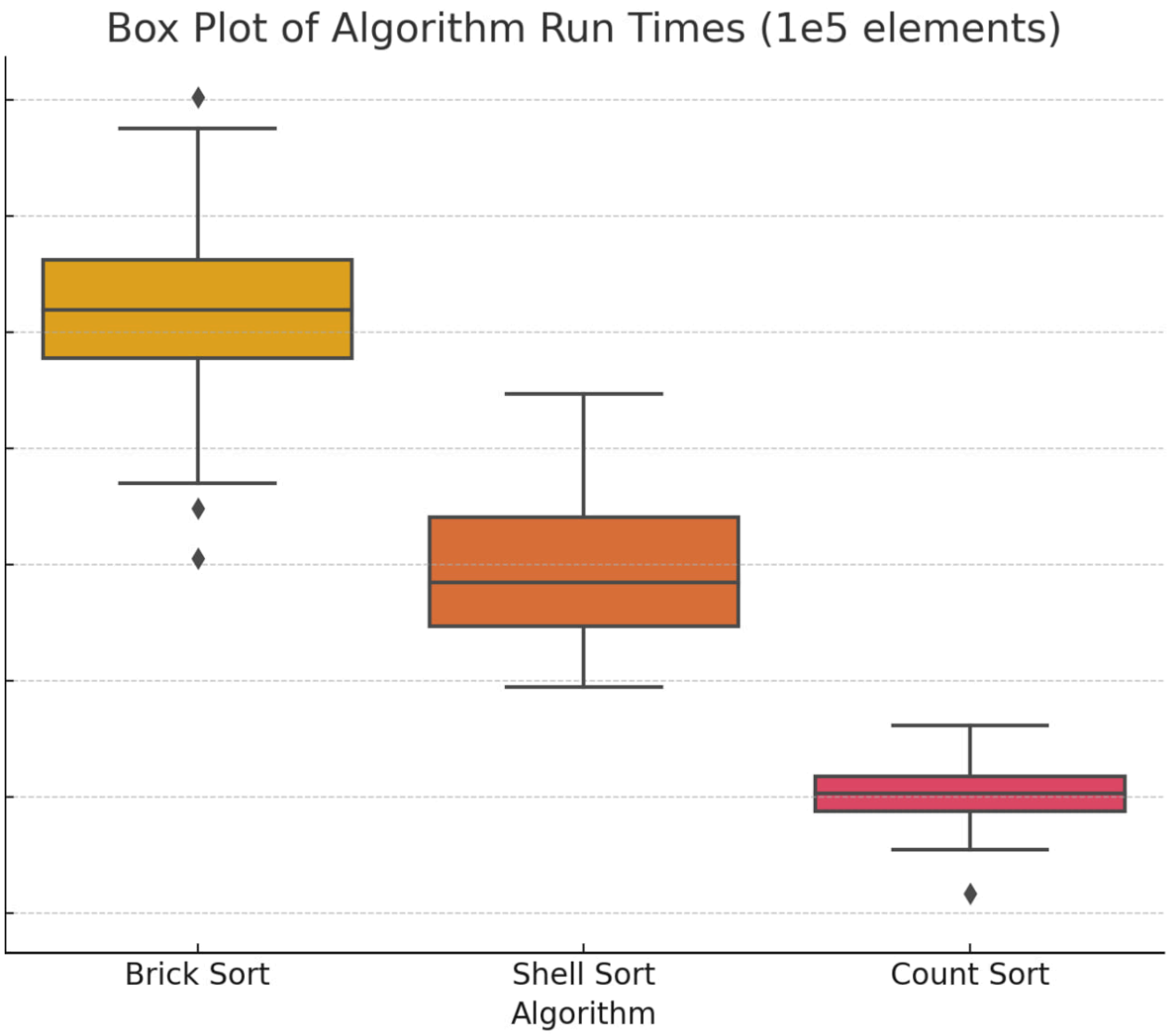
“Линейный график работы алгоритмов” Рисунок 1



“Box-plot графики для времени работы алгоритмов с числом элементов 1e4” Рисунок 2



“Box-plot графики для времени работы алгоритмов с числом элементов 1e5” Рисунок 3




**5. Заключение + Выводы**

В ходе выполнения данной лабораторной работы были успешно реализованы три алгоритма сортировки: Brick Sort, Shell Sort и Counting Sort. Каждый из этих алгоритмов был протестирован на различных наборах данных.

Результаты показали, что:

- Brick Sort обладает простой логикой, однако её временная сложность делает её неэффективной для больших объёмов данных.

- Sell sort продемонстрировала значительное улучшение производительности по сравнению с bricksort, за счет улучшения временной сложности алгоритма

- Сортировка подсчётом оказалась очень быстрой для диапазона целых чисел, ограниченного небольшим количеством уникальных значений, поскольку её временная и пространственная сложность зависят от диапазона значений, а не от количества элементов.

Таким образом, выбор подходящего алгоритма зависит от конкретной задачи и характеристик входных данных. В большинстве случаев выбор стоит между shellSort и countingSort, в зависимости от ограниченности количества уникальных значений

Возможными направлениями для дальнейшего исследования являются:

1. Реализация и тестирование других известных алгоритмов сортировки.
2. Исследование влияния начальных условий на производительность алгоритмов, например, влияние степени упорядоченности входных данных.
3. Оптимизация существующих реализаций путем улучшения структур данных или использования параллельных вычислений.

6. Приложения

ПРИЛОЖЕНИЕ А

Листинг кода файла sortirovki.cpp

```
#include <iostream>
#include <vector>
using namespace std;

void brickSort(vector<int>& arr) {
    bool isSorted = false;

    while (!isSorted) {
        isSorted = true;
         for (size_t i = 1; i <= arr.size() - 2; i
+= 2) {
            if (arr[i] > arr[i + 1]) {
                swap(arr[i], arr[i + 1]);
                isSorted = false;
            }
        }
         for (size_t i = 0; i <= arr.size() - 2; i
+= 2) {
            if (arr[i] > arr[i + 1]) {
                swap(arr[i], arr[i + 1]);
                isSorted = false;
            }
        }
```

```
        }
    }

    void shellSort(vector<int>& arr) {
        for (int gap = arr.size() / 2; gap > 0; gap /=
2) {
            for (size_t i = gap; i < arr.size(); i++) {
                int temp = arr[i];
                size_t j;
                 for (j = i; j >= gap && arr[j - gap] >
temp; j -= gap) {
                    arr[j] = arr[j - gap];
                }
                arr[j] = temp;
            }
        }
    }

    void countingSort(vector<int>& arr) {
        if (arr.empty()) return;

        int minElem = arr[0];
        int maxElem = arr[0];
        for (int num : arr) {
            if (num < minElem) minElem = num;
            if (num > maxElem) maxElem = num;
        }
        int range = maxElem - minElem + 1;

        // Массив подсчёта
        vector<int> count(range, 0);
        for (int num : arr) {
            count[num - minElem]++;
```

```
        }

        size_t index = 0;
        for (int i = 0; i < range; i++) {
            while (count[i] > 0) {
                arr[index++] = i + minElem;
                count[i]--;
            }
        }
    }

    int main() {
         vector<int> arr = {34, 2, 78, 1, 56, 99, 23,
12};

        cout << "Исходный массив: ";
        for (int num : arr) {
            cout << num << " ";
        }
        cout << endl;

        // brickSort(arr);
        shellSort(arr);
        // countingSort(arr);

        cout << "Отсортированный массив: ";
        for (int num : arr) {
            cout << num << " ";
        }
        cout << endl;

        return 0;
    }
```

ПРИЛОЖЕНИЕ В

Листинг кода файла tests_sortirovki.cpp

```
#include <gtest/gtest.h>
#include <vector>


extern void brickSort(std::vector<int>& arr);
extern void shellSort(std::vector<int>& arr);
extern void countingSort(std::vector<int>& arr);

// Тесты для brickSort
TEST(BrickSortTest, EmptyVector) {
    std::vector<int> emptyVec;
    brickSort(emptyVec);
    EXPECT_EQ(emptyVec.size(), 0);
}

TEST(BrickSortTest, SingleElement) {
    std::vector<int> singleVec{42};
    brickSort(singleVec);
    EXPECT_EQ(singleVec.size(), 1);
    EXPECT_EQ(singleVec[0], 42);
}

TEST(BrickSortTest, SortedVector) {
    std::vector<int> sortedVec{1, 2, 3, 4, 5};
    brickSort(sortedVec);
     EXPECT_EQ(sortedVec, (std::vector<int>{1, 2, 3,
4, 5}));
}

TEST(BrickSortTest, ReverseSortedVector) {
```

```
        std::vector<int> reverseSortedVec{5, 4, 3, 2, 1};
        brickSort(reverseSortedVec);
                                EXPECT_EQ(reverseSortedVec, (std::vector<int>{1, 2, 3, 4, 5}));
    }

    TEST(BrickSortTest, RandomVector) {
        std::vector<int> randomVec{34, 2, 78, 1, 56, 99, 23, 12};
        brickSort(randomVec);
          EXPECT_EQ(randomVec, (std::vector<int>{1, 2, 12, 23, 34, 56, 78, 99}));
    }

    // Тесты для shellSort
    TEST(ShellSortTest, EmptyVector) {
        std::vector<int> emptyVec;
        shellSort(emptyVec);
        EXPECT_EQ(emptyVec.size(), 0);
    }

    TEST(ShellSortTest, SingleElement) {
        std::vector<int> singleVec{42};
        shellSort(singleVec);
        EXPECT_EQ(singleVec.size(), 1);
        EXPECT_EQ(singleVec[0], 42);
    }

    TEST(ShellSortTest, SortedVector) {
        std::vector<int> sortedVec{1, 2, 3, 4, 5};
        shellSort(sortedVec);
```

```
        EXPECT_EQ(sortedVec, (std::vector<int>{1, 2, 3, 4, 5}));
    }

    TEST(ShellSortTest, ReverseSortedVector) {
        std::vector<int> reverseSortedVec{5, 4, 3, 2, 1};
        shellSort(reverseSortedVec);
                                EXPECT_EQ(reverseSortedVec, (std::vector<int>{1, 2, 3, 4, 5}));
    }

    TEST(ShellSortTest, RandomVector) {
        std::vector<int> randomVec{34, 2, 78, 1, 56, 99, 23, 12};
        shellSort(randomVec);
          EXPECT_EQ(randomVec, (std::vector<int>{1, 2, 12, 23, 34, 56, 78, 99}));
    }

    // Тесты для countingSort
    TEST(CountingSortTest, EmptyVector) {
        std::vector<int> emptyVec;
        countingSort(emptyVec);
        EXPECT_EQ(emptyVec.size(), 0);
    }

    TEST(CountingSortTest, SingleElement) {
        std::vector<int> singleVec{42};
        countingSort(singleVec);
        EXPECT_EQ(singleVec.size(), 1);
        EXPECT_EQ(singleVec[0], 42);
    }
```

```
    TEST(CountingSortTest, SortedVector) {
        std::vector<int> sortedVec{1, 2, 3, 4, 5};
        countingSort(sortedVec);
        EXPECT_EQ(sortedVec, (std::vector<int>{1, 2, 3,
4, 5}));
    }

    TEST(CountingSortTest, ReverseSortedVector) {
         std::vector<int> reverseSortedVec{5, 4, 3, 2,
1};
        countingSort(reverseSortedVec);
                            EXPECT_EQ(reverseSortedVec,
(std::vector<int>{1, 2, 3, 4, 5}));
    }

    TEST(CountingSortTest, RandomVector) {
         std::vector<int> randomVec{34, 2, 78, 1, 56,
99, 23, 12};
        countingSort(randomVec);
          EXPECT_EQ(randomVec, (std::vector<int>{1, 2,
12, 23, 34, 56, 78, 99}));
    }

    int main(int argc, char** argv) {
        ::testing::InitGoogleTest(&argc, argv);
        return RUN_ALL_TESTS();
    }
```