



МИНОБР НАУКИ РОССИИ  
Федеральное государственное бюджетное образовательное учреждение  
высшего образования  
**«МИРЭА — Российский технологический университет»**

**РТУ МИРЭА**

---

Отчёт по выполнению практического задания 4  
**Тема:** Эмпирический анализ алгоритмов сортировки  
Дисциплина Структуры и алгоритмы обработки данных

|                  |            |
|------------------|------------|
| Выполнил студент | Пак С.А.   |
| группа           | ИКБО-05-20 |

**Москва 2021**

## СОДЕРЖАНИЕ

|                            |    |
|----------------------------|----|
| 1. Отчёт по заданию 1..... | 3  |
| 2. Отчёт по заданию 2..... | 7  |
| 3. Отчёт по заданию 3..... | 12 |
| 4. Выводы.....             | 18 |

Номер варианта индивидуального задания: 2.

## 1. Отчёт по заданию 1

1. Пусть массив из 6 целых чисел, заполненный случайными числами выглядит следующим образом: [9, 1, 3, 4, 8, 2]. Упорядочим элементы этого массива в порядке возрастания с помощью алгоритма простого обмена.

Алгоритм состоит из повторяющихся проходов по массиву. За каждый проход элементы последовательно сравниваются попарно и, если порядок в паре неверный, выполняется перестановка элементов.

Таким образом, за первый проход наибольший элемент оказывается в конце массива. Поэтому при дальнейших проходах не нужно проверять самые последние элементы. То есть после первого прохода не нужно проверять последний элемент, после второго — предпоследний и т.д.

Получается, что за каждый проход в конце массива будут наибольшие элементы, а в начале наименьшие.

Если рассматривать процесс сортировки массива [9, 1, 3, 4, 8, 2] поэтапно, то получается следующая картина:

[9, 1, 3, 4, 8, 2] → [1, 3, 4, 8, 2, 9] → [1, 3, 4, 2, 8, 9] → [1, 3, 2, 4, 8, 9] → [1, 2, 3, 4, 8, 9].

2. Определим функцию роста времени выполнения алгоритма (табл.1).

Таблица 1

| Оператор                   | Количество выполнения операций |
|----------------------------|--------------------------------|
| for i ← 0 to N-1 do        | N                              |
| for j ← 0 to N-i-1 do      | $\sum_{j=1}^{N-1} t_j$         |
| if a[j] > a[j+1] then      | $\sum_{j=1}^{N-1} t_j$         |
| swap(array[j], array[j+1]) | $3 * \sum_{j=1}^{N-1} t_j$     |
| endif                      |                                |
| od                         |                                |
| od                         |                                |

$$\sum_{j=1}^{N-1} t_j = 1 + 2 + \dots + N-1 = (1+N-1)*(N-1)/2 = N*(N-1)/2 = (N^2-N) / 2.$$

Следовательно,  $T(n) = N + (N^2-N)/2 + (N^2-N)/2 + (3 / 2) * (N^2-N) = N + (5/2) * (N^2-N)$ ;

Если рассчитать  $T_T = O(n) = O(N + (5/2)*(N^2-N)) = N^2$ .

3. Сводная таблица результатов выполнения сортировки (табл.2):

Таблица 2

| <b>n</b> | <b>T(n)</b> | <b>T<sub>r</sub>=f(C+M)</b> | <b>T<sub>π</sub>=Cφ+Mφ</b> |
|----------|-------------|-----------------------------|----------------------------|
| 100      | 0,000045 с  | 10000                       | 7576                       |
| 1000     | 0,002357 с  | 1000000                     | 744619                     |
| 10000    | 0,166506 с  | 100000000                   | 74175026                   |
| 100000   | 18,704 с    | 10000000000                 | 7427200146                 |
| 1000000  | 1809,73 с   | 1000000000000               | 743248583255               |

#### 4. Код алгоритма и основной программы

Основная программа состоит из нескольких файлов: main.cpp; funcs.hpp (файл содержит вспомогательные функции и функции алгоритмов сортировки); funcs.cpp (файл содержит реализации вспомогательных функций и функций алгоритмов сортировки).

Файл main.cpp:

```
#include <ctime>
#include <string>
#include "../includes/funcs.hpp"

using namespace std;

int main() {
    int arrayLength;

    cout << "Введите количество элементов массива: ";
    cin >> arrayLength;

    vector<int> array;

    string needRange;

    cout << endl << "Задать диапазон чисел (да/нет)? ";
    cin >> needRange;

    if (needRange == "да" || needRange == "Да" || needRange == "дА" || needRange == "ДА") {
        int arrayMinValue;
        int arrayMaxValue;

        cout << "Введите диапазон чисел: ";
        cin >> arrayMinValue >> arrayMaxValue;

        array = generate(arrayLength, arrayMinValue, arrayMaxValue);
        goto start; // не очень хорошо так делать
    }

    array = generate(arrayLength);

start:
```

```

    clock_t time = clock();
    bubbleSort(array);
    time = clock() - time;
    double elapsed = static_cast<double>(time) / CLOCKS_PER_SEC;
    cout << "Время выполнения: " << elapsed << " секунд" << endl;

    return 0;
}

```

Файл funcs.hpp:

```

#ifndef _FUNCS_HPP_
#define _FUNCS_HPP_

#include <random>
#include <vector>
#include <iostream>
#include <algorithm>

/**
 * Создаёт массив, заполненный случайными числами в заданном диапазоне, заданной
 длины
 * @param length      количество элементов массива
 * @param min         минимальное значение элемента массива
 * @param max         максимальное значение элемента массива
 */
std::vector<int> generate(const int& length, const int& min = -10, const int&
max = 25);

/**
 * Упорядочивает массив по возрастанию с помощью алгоритма сортировки пузырьком
 * @param array       упорядочиваемый массив
 */
void bubbleSort(std::vector<int>& array);

/**
 * Выводит массив на экран в формате [эл.1, эл.2, ..., эл.N]
 * @param out         поток вывода
 * @param array       массив, выводимый на экран
 */
std::ostream& operator<<(std::ostream& out, const std::vector<int>& array);

#endif

```

Файл funcs.cpp:

```

#include "../funcs.hpp"

/**
 * Создаёт массив, заполненный случайными числами в заданном диапазоне, заданной
 длины
 * @param length      количество элементов массива

```

```

* @param min            минимальное значение элемента массива
* @param max            максимальное значение элемента массива
*/
std::vector<int> generate(const int& length, const int& min, const int& max) {
    using namespace std;

    random_device rd;
    mt19937 gen(rd());
    uniform_int_distribution<int> distr(min, max);

    vector<int> array(length);

    for (int& element : array) {
        element = distr(gen);
    }

    return array;
}

/**
 * Упорядочивает массив по возрастанию с помощью алгоритма сортировки пузырьком
 * @param array          упорядочиваемый массив
 */
void bubbleSort(std::vector<int>& array) {
    const int N = array.size();

    unsigned long long comparisons = 0;
    unsigned long long transferrings = 0;

    for (int i = 0; i < N - 1; ++i) {
        for (int j = 0; j < N - i - 1; ++j) { // до самого последнего элемента не
            нужно доходить
            ++comparisons;
            if (array[j] > array[j + 1]) {
                ++transferrings;
                std::swap(array[j], array[j + 1]);
            }
        }
    }

    std::cout << "Выполнено сравнений: " << comparisons << std::endl;
    std::cout << "Выполнено перемещений: " << transferrings << std::endl;
}

/**
 * Выводит массив на экран в формате [эл.1, эл.2, ..., эл.N]
 * @param out            поток вывода
 * @param array          массив, выводимый на экран
 */
std::ostream& operator<<(std::ostream& out, const std::vector<int>& array) {
    const int length = static_cast<int>(array.size());

    out << "[" << array[0];

    for (int i = 1; i < length; ++i) {
        out << ", " << array[i];
    }
}

```

```

out << "];
return out;
}

```

5. График зависимости теоретической и практической вычислительной сложности алгоритма.

На рис.1 представлены два графика. Оранжевым цветом обозначен график практической вычислительной сложности алгоритма. Синим цветом обозначен график теоретической вычислительной сложности алгоритма.

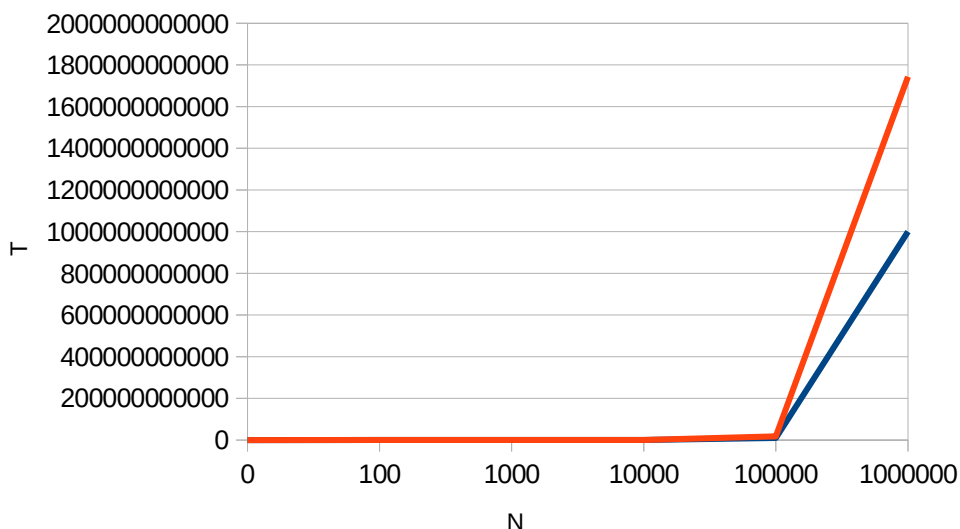


Рис.1 Графики сложности алгоритма

6. По результатам таблицы можно сделать вывод о том, что практическая сложность имеет приблизительно квадратичную зависимость от количества элементов  $N$ , также как и теоретическая сложность.

## 2. Отчёт по заданию 2

1. Сводная таблица результатов при применении метода к массиву, упорядоченному по возрастанию (табл.3):

Таблица 3

| $n$     | $T(n)$     | $T_r=f(C+M)$  | $T_n=C\phi+M\phi$ |
|---------|------------|---------------|-------------------|
| 100     | 0,000047 с | 10000         | 4950              |
| 1000    | 0,000931 с | 1000000       | 499500            |
| 10000   | 0,045861 с | 100000000     | 49995000          |
| 100000  | 3,99932 с  | 10000000000   | 4999950000        |
| 1000000 | 271,076 с  | 1000000000000 | 499999500000      |

2. Сводная таблица результатов при применении метода к массиву, упорядоченному по убыванию (табл.4):

Таблица 4

| <b>n</b> | <b>T(n)</b> | <b><math>T_r=f(C+M)</math></b> | <b><math>T_n=C\phi+M\phi</math></b> |
|----------|-------------|--------------------------------|-------------------------------------|
| 100      | 0,000062 с  | 10000                          | 9900                                |
| 1000     | 0,002522 с  | 1000000                        | 999000                              |
| 10000    | 0,104109 с  | 100000000                      | 99990000                            |
| 100000   | 9,56826 с   | 10000000000                    | 9999900000                          |
| 1000000  | 1076,96 с   | 1000000000000                  | 999999000000                        |

3. Код программы, которая выполняет тестирование алгоритма.  
Файл main.cpp:

```
#include <ctime>
#include <string>
#include "../includes/funcs.hpp"

using namespace std;

int main() {
    int arrayLength;

    cout << "Введите количество элементов массива: ";
    cin >> arrayLength;

    string increasing;

    cout << "Создать возрастающую последовательность (да/нет)? ";
    cin >> increasing;

    vector<int> array;

    if (increasing == "да" || increasing == "ДА" || increasing == "Да" ||
        increasing == "дА") {
        array = generateSorted(arrayLength);
        goto start;
    }

    array = generateSorted(arrayLength, false);

start:
    clock_t time = clock();
    bubbleSort(array);
    time = clock() - time;
    double elapsed = static_cast<double>(time) / CLOCKS_PER_SEC;
    cout << "Время выполнения: " << elapsed << " секунд" << endl;

    return 0;
}
```



Файл funcs.hpp:

```
#ifndef _FUNCS_HPP_
#define _FUNCS_HPP_

#include <random>
#include <vector>
#include <iostream>
#include <algorithm>

/**
 * Создаёт массив, заполненный случайными числами в заданном диапазоне, заданной
 длины
 * @param length      количество элементов массива
 * @param min          минимальное значение элемента массива
 * @param max          максимальное значение элемента массива
 */
std::vector<int> generate(const int& length, const int& min = -10, const int&
max = 25);

/**
 * Создаёт массив, упорядоченный по умолчанию в порядке возрастания
 * @param length      количество элементов массива
 * @param increasing   если true, то создаётся массив, элементы которого
 расположены в порядке возрастания
 */
std::vector<int> generateSorted(const int& length, const bool& increasing =
true);

/**
 * Упорядочивает массив по возрастанию с помощью алгоритма сортировки пузырьком
 * @param array        упорядочиваемый массив
 */
void bubbleSort(std::vector<int>& array);

/**
 * Выводит массив на экран в формате [эл.1, эл.2, ..., эл.N]
 * @param out          поток вывода
 * @param array        массив, выводимый на экран
 */
std::ostream& operator<<(std::ostream& out, const std::vector<int>& array);

#endif
```

Файл funcs.cpp:

```
#include "funcs.hpp"

/**
 * Создаёт массив, заполненный случайными числами в заданном диапазоне, заданной
 длины
 * @param length      количество элементов массива
 * @param min          минимальное значение элемента массива
```

```

* @param max            максимальное значение элемента массива
*/
std::vector<int> generate(const int& length, const int& min, const int& max) {
    using namespace std;

    random_device rd;
    mt19937 gen(rd());
    uniform_int_distribution<int> distr(min, max);

    vector<int> array(length);

    for (int& element : array) {
        element = distr(gen);
    }

    return array;
}

/**
 * Создаёт массив, упорядоченный по умолчанию в порядке возрастания
 * @param length        количество элементов массива
 * @param increasing     если true, то создаётся массив, элементы которого
расположены в порядке возрастания
 */
std::vector<int> generateSorted(const int& length, const bool& increasing) {
    std::vector<int> array(length);

    if (increasing) {
        for (int i = 0; i < length; ++i) {
            array[i] = i;
        }

        return array;
    }

    for (int i = length - 1, j = 0; i >= 0; --i, ++j) {
        array[j] = i;
    }

    return array;
}

/**
 * Упорядочивает массив по возрастанию с помощью алгоритма сортировки пузырьком
 * @param array          упорядочиваемый массив
 */
void bubbleSort(std::vector<int>& array) {
    const int N = array.size();

    unsigned long long comparisons = 0;
    unsigned long long transferrings = 0;

    for (int i = 0; i < N - 1; ++i) {
        for (int j = 0; j < N - i - 1; ++j) { // до самого последнего элемента не
        нужно доходить
            ++comparisons;
            if (array[j] > array[j + 1]) {

```

```

        ++transferrings;
        std::swap(array[j], array[j + 1]);
    }
}

std::cout << "Выполнено сравнений: " << comparisons << std::endl;
std::cout << "Выполнено перемещений: " << transferrings << std::endl;
}

/**
 * Выводит массив на экран в формате [эл.1, эл.2, ..., эл.N]
 * @param out        поток вывода
 * @param array       массив, выводимый на экран
 */
std::ostream& operator<<(std::ostream& out, const std::vector<int>& array) {
    const int length = static_cast<int>(array.size());

    out << "[" << array[0];

    for (int i = 1; i < length; ++i) {
        out << ", " << array[i];
    }

    out << "]";

    return out;
}

```

4. Графики зависимости теоретической и практической вычислительной сложности алгоритма.

На рисунке (рис.2) синим цветом обозначен график сложности алгоритма в среднем случае, оранжевым цветом — в лучшем случае, и жёлтым цветом — в худшем случае.

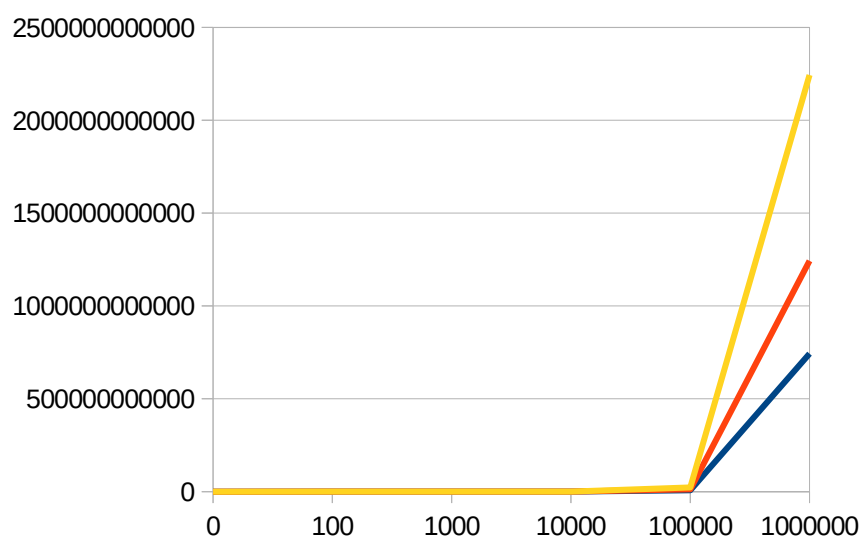


Рис.2 Графики сложности алгоритма для массива, упорядоченного по

## убыванию

5. В алгоритме используется только массив на  $n$  элементов, поэтому ёмкостная сложность алгоритма сортировки простого обмена будет зависеть линейно от  $n$ . То есть  $O(n)$ .

6. При небольших объёмах данных ( $n$  меньше либо равно 100) различия во времени выполнения минимальны. Однако с ростом  $n$  время в худшем случае становится в конечном счёте в почти 4 раза больше, чем в лучшем случае.

### 3. Отчёт по заданию 3

1. Пусть массив из 6 элементов, заполненный случайными числами выглядит следующим образом: [9, 1, 3, 4, 8, 2]. Упорядочим его элементы в порядке возрастания с помощью алгоритма сортировки простой вставки.

Алгоритм можно описать так: элементы входной последовательности просматриваются по одному, и каждый новый поступивший элемент размещается в подходящее место среди ранее упорядоченных элементов.

Покажем этапы сортировки массива: [9, 1, 3, 4, 8, 2] → [1, 9, 3, 4, 8, 2] → [1, 3, 9, 4, 8, 2] → [1, 3, 4, 9, 8, 2] → [1, 3, 4, 8, 9, 2] → [1, 2, 3, 4, 8, 9].

2. Определим функцию роста времени выполнения алгоритма (табл.5).

Таблица 5

| Операторы             | Количество выполнений оператора |
|-----------------------|---------------------------------|
| for i ← 1 to N do     | N                               |
| for j ← i to 1 do     | $\sum_{j=1}^{N-1} t_j$          |
| if a[j] < a[j-1] then | $\sum_{j=1}^{N-1} t_j$          |
| swap(a[j], a[j-1])    | $3 * \sum_{j=1}^{N-1} t_j$      |
| endif                 |                                 |
| od                    |                                 |
| od                    |                                 |

$$\sum_{j=1}^{N-1} t_j = 1 + 2 + \dots + (N-1) = (1 + N-1) * (N-1) / 2 = (N^2 - N) / 2.$$

Следовательно,  $T(N) = N + (N^2 - N) / 2 + (N^2 - N) / 2 + 3 / 2 * (N^2 - N) / 2 = N + 5 / 2 * (N^2 - N)$ .

Если рассчитать  $T_r = O(n) = O(T(N)) = O(N + 5 / 2 * (N^2 - N)) = N^2$ .

3. Сводная таблица результатов выполнения сортировки (табл.6):

Таблица 6

| n | T(n) | $T_r = f(C+M)$ | $T_n = C\phi + M\phi$ |
|---|------|----------------|-----------------------|
|---|------|----------------|-----------------------|

|         |            |               |              |
|---------|------------|---------------|--------------|
| 100     | 0,00005 с  | 10000         | 7081         |
| 1000    | 0,00132 с  | 1000000       | 737927       |
| 10000   | 0,075068 с | 100000000     | 74343015     |
| 100000  | 7,09803 с  | 10000000000   | 7428628694   |
| 1000000 | 465,945 с  | 1000000000000 | 743063422252 |

#### 4. Код всей программы.

Файл main.cpp:

```
#include <ctime>
#include <string>
#include "../includes/funcs.hpp"

using namespace std;

int main() {
    int arrayLength;

    cout << "Введите количество элементов массива: ";
    cin >> arrayLength;

    string needRange;

    cout << "Задать диапазон (да/нет)? ";
    cin >> needRange;

    vector<int> array;

    if (needRange == "да" || needRange == "ДА" || needRange == "Да" || needRange
== "дА") {
        int arrayMinVal;
        int arrayMaxVal;

        cout << "Введите диапазон чисел: ";
        cin >> arrayMinVal >> arrayMaxVal;

        array = generate(arrayLength, arrayMinVal, arrayMaxVal);
        goto start;
    }

    array = generate(arrayLength);

start:
    clock_t time = clock();

    insertionSort(array);
    time = clock() - time;

    double elapsed = static_cast<double>(time) / CLOCKS_PER_SEC;
    cout << "Время выполнения: " << elapsed << " секунд" << endl;

    return 0;
}
```

```
}
```

Файл funcs.hpp:

```
#ifndef _FUNCS_HPP_
#define _FUNCS_HPP_

#include <random>
#include <vector>
#include <iostream>
#include <algorithm>

/**
 * Создаёт массив, заполненный случайными числами в заданном диапазоне, заданной
 длины
 * @param length      количество элементов массива
 * @param min         минимальное значение элемента массива
 * @param max         максимальное значение элемента массива
 */
std::vector<int> generate(const int& length, const int& min = -10, const int&
max = 25);

/**
 * Создаёт массив, упорядоченный по умолчанию в порядке возрастания
 * @param length      количество элементов массива
 * @param increasing   если true, то создаётся массив, элементы которого
 расположены в порядке возрастания
 */
std::vector<int> generateSorted(const int& length, const bool& increasing =
true);

/**
 * Упорядочивает массив по возрастанию с помощью алгоритма сортировки пузырьком
 * @param array       упорядочиваемый массив
 */
void bubbleSort(std::vector<int>& array);

/**
 * Упорядочивает массив по возрастанию с помощью алгоритма сортировки вставками
 * @param array       упорядочиваемый массив
 */
void insertionSort(std::vector<int>& array);

/**
 * Выводит массив на экран в формате [эл.1, эл.2, ..., эл.N]
 * @param out         поток вывода
 * @param array       массив, выводимый на экран
 */
std::ostream& operator<<(std::ostream& out, const std::vector<int>& array);

#endif
```

Файл funcs.cpp:

```

#include "./funcs.hpp"

/**
 * Создаёт массив, заполненный случайными числами в заданном диапазоне, заданной
 длины
 * @param length      количество элементов массива
 * @param min         минимальное значение элемента массива
 * @param max         максимальное значение элемента массива
 */
std::vector<int> generate(const int& length, const int& min, const int& max) {
    using namespace std;

    random_device rd;
    mt19937 gen(rd());
    uniform_int_distribution<int> distr(min, max);

    vector<int> array(length);

    for (int& element : array) {
        element = distr(gen);
    }

    return array;
}

/**
 * Создаёт массив, упорядоченный по умолчанию в порядке возрастания
 * @param length      количество элементов массива
 * @param increasing   если true, то создаётся массив, элементы которого
 расположены в порядке возрастания
 */
std::vector<int> generateSorted(const int& length, const bool& increasing) {
    std::vector<int> array(length);

    if (increasing) {
        for (int i = 0; i < length; ++i) {
            array[i] = i;
        }

        return array;
    }

    for (int i = length - 1, j = 0; i >= 0; --i, ++j) {
        array[j] = i;
    }

    return array;
}

/**
 * Упорядочивает массив по возрастанию с помощью алгоритма сортировки пузырьком
 * @param array        упорядочиваемый массив
 */
void bubbleSort(std::vector<int>& array) {
    const int N = array.size();

```

```

    unsigned long long comparisons = 0;
    unsigned long long transferrings = 0;

    for (int i = 0; i < N - 1; ++i) {
        for (int j = 0; j < N - i - 1; ++j) {    // до самого последнего элемента не
        нужно доходить
            ++comparisons;
            if (array[j] > array[j + 1]) {
                ++transferrings;
                std::swap(array[j], array[j + 1]);
            }
        }
    }

    std::cout << "Выполнено сравнений: " << comparisons << std::endl;
    std::cout << "Выполнено перемещений: " << transferrings << std::endl;
}

/**
 * Упорядочивает массив по возрастанию с помощью алгоритма сортировки вставками
 * @param array        упорядочиваемый массив
 */
void insertionSort(std::vector<int>& array) {
    const int N = array.size();

    unsigned long long comparisons = static_cast<unsigned long long>(N / 2 * (N -
1));
    unsigned long long transferrings = 0;

    for (int i = 1; i < N; ++i) {
        for (int j = i; j > 0 && array[j] < array[j - 1]; --j) {
            // ++comparisons;
            // if (array[j] < array[j - 1]) {
            //     std::swap(array[j], array[j - 1]);
            //     ++transferrings;
            // }
            std::swap(array[j], array[j - 1]);
            ++transferrings;
        }
    }

    std::cout << "Сравнений: " << comparisons << std::endl;
    std::cout << "Перемещений: " << transferrings << std::endl;
}

/**
 * Выводит массив на экран в формате [эл.1, эл.2, ..., эл.N]
 * @param out        поток вывода
 * @param array        массив, выводимый на экран
 */
std::ostream& operator<<(std::ostream& out, const std::vector<int>& array) {
    const int length = static_cast<int>(array.size());

    out << "[" << array[0];

    for (int i = 1; i < length; ++i) {

```



```

    out << ", " << array[i];
}

out << "];

return out;
}

```

5. Графики зависимости теоретической и практической сложности алгоритма

Синим цветом (рис.4) обозначен график теоретической сложности, оранжевым — график сложности алгоритма сортировки вставками, жёлтым — график сложности алгоритма сортировки пузырьком.

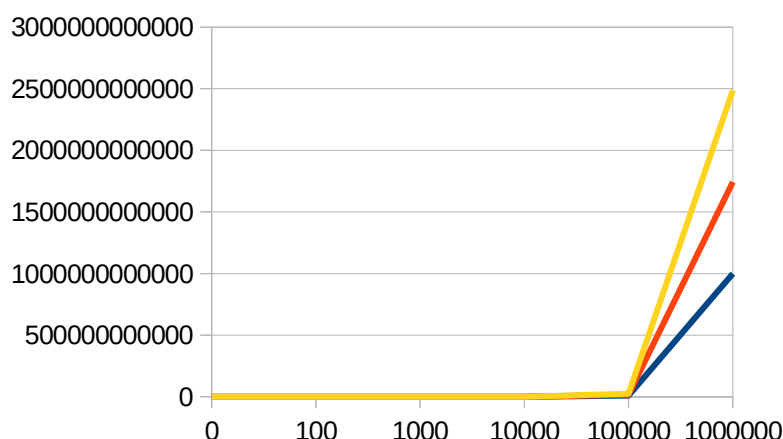


Рис.4 Графики алгоритмов сортировки вставками и пузырьком

6. Эффективность алгоритма — это свойства алгоритма, которое связано с вычислительными ресурсами, используемыми алгоритмом. Алгоритм должен быть проанализирован с целью определения необходимых алгоритму ресурсов.

7. Из результатов видно, что алгоритм сортировки пузырьком на массиве из 100 элементов был немного быстрее. Однако при росте количества элементов массива алгоритм сортировки вставками был быстрее и эффективнее.

Следовательно, алгоритм сортировки простой вставкой эффективнее, чем алгоритм сортировки простого обмена.

## 4. Выводы

В ходе работы были изучены алгоритмы сортировки пузырьком и сортировки вставками. Измерено время работы обоих алгоритмов, и было выяснено экспериментальным путём, что алгоритм сортировки вставками быстрее, чем алгоритм сортировки пузырьком.