



МИНИСТЕРСТВО НАУКИ ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ  
Федеральное государственное бюджетное образовательное учреждение  
высшего образования  
«МИРЭА - Российский технологический университет»

**РТУ МИРЭА**

---

Отчет по выполнению практического задания №2

**Тема:**

**«Эмпирический анализ сложности простых  
алгоритмов сортировки»**

Дисциплина: «Структуры и алгоритмы обработки данных»

Выполнил студент:

Подоплелов А.С.

Группа:

ИНБО-21-23

Москва - 2024

## СОДЕРЖАНИЕ

ЦЕЛЬ РАБОТЫ.....	2
1.1 УСЛОВИЕ ЗАДАНИЯ 1.....	2
1.2 УСЛОВИЕ ЗАДАНИЯ 2.....	3
1.3 УСЛОВИЕ ЗАДАНИЯ 3.....	4
РЕШЕНИЕ ЗАДАНИЯ 1.....	4
2. 1 ОПИСАНИЕ И РЕАЛИЗАЦИЯ ПЕРВОГО АЛГОРИТМА.....	4
2.2 АНАЛИЗ ЭФФЕКТИВНОСТИ ПЕРВОГО АЛГОРИТМА.....	5
2.3 ВЫВОД ОБ ЭМПИРИЧЕСКОЙ ВЫЧИСЛИТЕЛЬНОЙ СЛОЖНОСТИ АЛГОРИТМА НА ОСНОВЕ СКОРОСТИ РОСТА ФУНКЦИИ.....	6
РЕШЕНИЕ ЗАДАНИЯ 2.....	6
РЕШЕНИЕ ЗАДАНИЯ 3.....	7
3.1 ОПИСАНИЕ И РЕАЛИЗАЦИЯ ВТОРОГО АЛГОРИТМА.....	7
3.2 АНАЛИЗ ЭФФЕКТИВНОСТИ ВТОРОГО АЛГОРИТМА.....	8
3.3 ВЫВОД ОБ ЭМПИРИЧЕСКОЙ ВЫЧИСЛИТЕЛЬНОЙ СЛОЖНОСТИ ВТОРОГО АЛГОРИТМА.....	8
ВЫВОД.....	10
СПИСОК ИНФОРМАЦИОННЫХ ИСТОЧНИКОВ.....	10

## ЦЕЛЬ РАБОТЫ

Актуализация знаний и приобретение практических умений по эмпирическому определению вычислительной сложности алгоритмов.

### 1.1 УСЛОВИЕ ЗАДАНИЯ 1

Оценить эмпирически вычислительную сложность алгоритма простой сортировки на массиве, заполненном случайными числами (средний случай).

1. Составить функцию простой сортировки одномерного целочисленного массива  $A[n]$ , используя алгоритм простого обмена («пузырек», Exchange sort). Провести тестирование программы на исходном массиве, где  $n=10$ .

2. Используя теоретический подход, определить для алгоритма:

- а. Что будет ситуациями лучшего, среднего и худшего случаев.
- б. Функции роста времени работы алгоритма от объёма входа для лучшего и худшего случаев.

3. Провести контрольные прогоны программы массивов случайных чисел при  $n = 100, 1000, 10000, 100000$  и  $1000000$  элементов с вычислением времени выполнения  $T(n)$  – (в миллисекундах/секундах). Полученные результаты свести в сводную таблицу 1.

4. Провести эмпирическую оценку вычислительной сложности алгоритма, для чего предусмотреть в программе подсчет фактического количества критических операции  $T_n$  как сумму сравнении  $C_n$  и перемещении  $M_n$ . Полученные результаты вставить в сводную таблицу 1.

5. Построить график функции роста  $T_n$  этого алгоритма от размера массива  $n$ .

6. Определить ёмкостную сложность алгоритма.

7. Сделать вывод об эмпирической вычислительной сложности алгоритма на основе скорости роста функции роста.

Таблица 1 – Сводная таблица результатов

$n$	$T(n)$	$T_T = C + M$	$T_n = C_n + M_n$
100			

1000			
10000			
100000			
1000000			

## 1.2 УСЛОВИЕ ЗАДАНИЯ 2

Оценить вычислительную сложность алгоритма простой сортировки в наихудшем и наилучшем случаях.

1. Провести дополнительные прогоны программы на массивах при  $n = 100, 1000, 10000, 100000$  и  $1000000$  элементов, отсортированных:

- Строго в убывающем порядке значений, результаты представить в сводной таблице по формату Таблицы 1;

- Строго в возрастающем порядке значений, результаты представить в сводной таблице по формату Таблицы 1;

2. Сделать вывод о зависимости (или независимости) алгоритма сортировки от исходной упорядоченности массива.

## 1.3 УСЛОВИЕ ЗАДАНИЯ 3

Сравнить эффективность алгоритмов простых сортировок.

1. Выполнить разработку и программную реализацию второго алгоритма (Простая вставка – Insertion Sort).

2. Аналогично заданиям 1 и 2 сформировать таблицы с результатами эмпирического исследования второго алгоритма в среднем, лучшем и худшем случаях в соответствии с форматом Таблицы 1 (на тех же массивах, что и в заданиях 1 и 2).

3. Определить ёмкостную сложность алгоритма от  $n$ .

4. На одном сравнительном графике отобразить функции  $T_n(n)$  двух алгоритмов сортировки в худшем случае.

5. Аналогично на другом общем графике отобразить функции  $T_n(n)$  двух алгоритмов сортировки для лучшего случая.

6. Выполнить сравнительный анализ полученных результатов для двух алгоритмов.

## РЕШЕНИЕ ЗАДАНИЯ 1

### 2.1 ОПИСАНИЕ И РЕАЛИЗАЦИЯ ПЕРВОГО АЛГОРИТМА

Сортировка простого обмена («Exchange Sort») — это простой алгоритм сортировки, состоящий в повторяющихся проходах по сортируемому массиву. На каждой итерации последовательно сравниваются соседние элементы, и, если порядок в паре неверный, то элементы меняют местами.

Инвариант цикла: после выполнения «i-й» операции все элементы массива, находящиеся в «верхней» части массива будут отсортированы. Размерность массива конечна, следовательно конечен и цикл.

Алгоритм реализован на языке C++. (См. Рисунок 1).

```
void algn2(int* x, int n) {  
    for (int i = 0; i < n; i++) {  
        for (int j = 0; j < n - 1; j++) {  
            if (x[j] > x[j + 1]) {  
                swap(x[j], x[j + 1]);  
            }  
        }  
    }  
}
```

Рисунок 1 — Реализация алгоритма «Пузырьком» на C++

### 2.2 АНАЛИЗ ЭФФЕКТИВНОСТИ ПЕРВОГО АЛГОРИТМА

Лучший случай: массив изначально отсортирован по возрастанию. Функция роста:  $T(n) = n - 1$ .

Средний случай: массив заполнен случайными числами. Функция роста:  $T(n) = 0,5n(n-1)$ .

Худший случай: массив отсортирован по убыванию. Функция роста:  $T(n) = n(n-1)$ .

В таблице 2 показаны теоретические и практические результаты вычисления количества совершённых операций и времени работы. На рис. 2

показана графическая визуализация зависимости количества операций от длины массива.

Таблица 2 – Сводная таблица результатов для алгоритма «Exchange Sort»

<b>n</b>	<b>T(n), мс</b>	<b><math>T_n = C_n + M_n</math></b>
100	0,009	3500
1000	0,437	35000
10000	49,820	350000
100000	10394,5	3500000
1000000	Невозможно измерить	Невозможно измерить

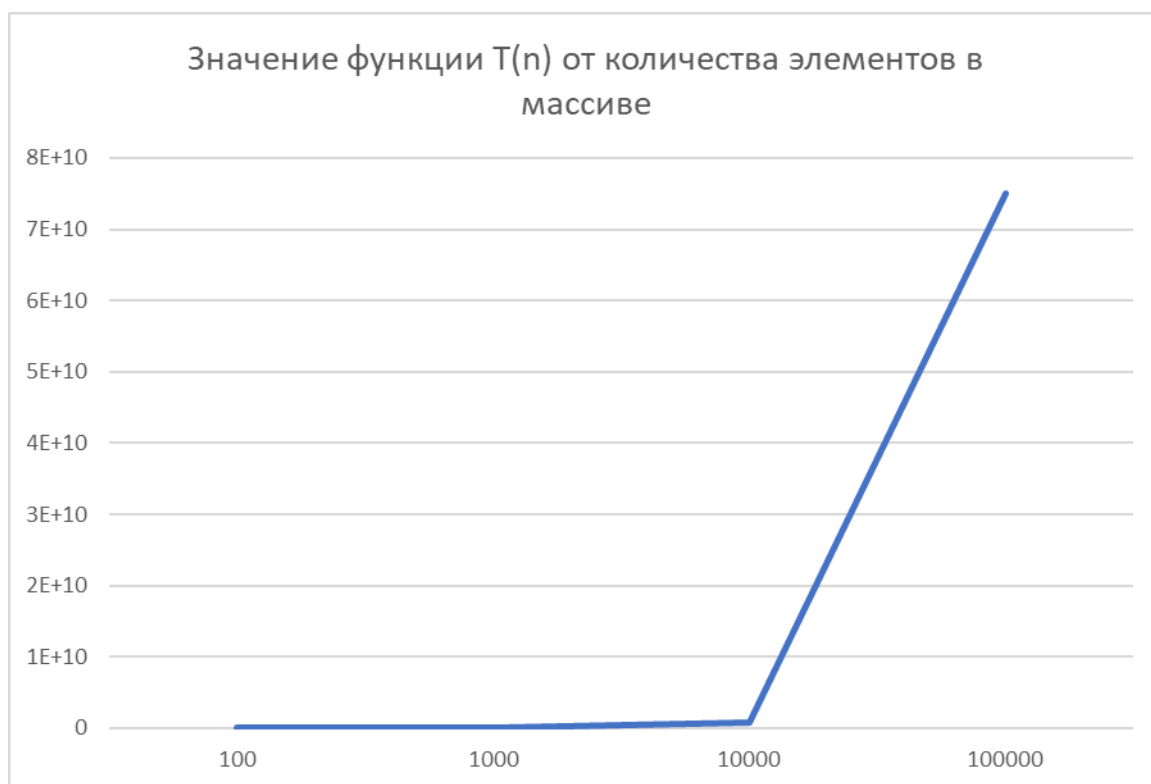


Рисунок 2 – Функция роста сортировки «пузырьком»

Ёмкостная сложность:  $O(n)$ , так как программе необходимо хранить лишь сам массив из «N» элементов.

## **2.3 ВЫВОД ОБ ЭМПИРИЧЕСКОЙ ВЫЧИСЛИТЕЛЬНОЙ СЛОЖНОСТИ АЛГОРИТМА НА ОСНОВЕ СКОРОСТИ РОСТА ФУНКЦИИ**

Сортировка простым обменом наиболее эффективна когда массив уже частично отсортирован и когда элементов массива не много. Судя по полученным результатам и графику функции роста, зависимость количества операций от длины массива – квадратичная.



## РЕШЕНИЕ ЗАДАНИЯ 2

Проверим работу алгоритма сортировки вставками на лучшем ( $T_1$ ) и худшем ( $T_2$ ) случае.

В таблице 2 описаны полученные результаты алгоритма сортировки вставками на лучшем и худшем случаях.

Таблица 2 – Результаты работы алгоритма на разных входных данных

<b>n</b>	<b><math>T_1(n)</math>, мс</b>	<b><math>T_2(n)</math>, мс</b>	<b><math>T_2=C_2+M_2</math></b>	<b><math>T_1=C_1+M_1</math></b>
100	0,00002	0,003	9900	99
1000	0,0001	0,427	999000	999
10000	0,001	54,628	99990000	9999
100000	0,014	5260,59	9999900000	99999
1000000	0,302	537812	999999000000	999999

По результатам таблицы 2 можно сделать вывод, что чем меньше массив отсортирован, тем больше времени будет затрачено на сортировку данным алгоритмом.

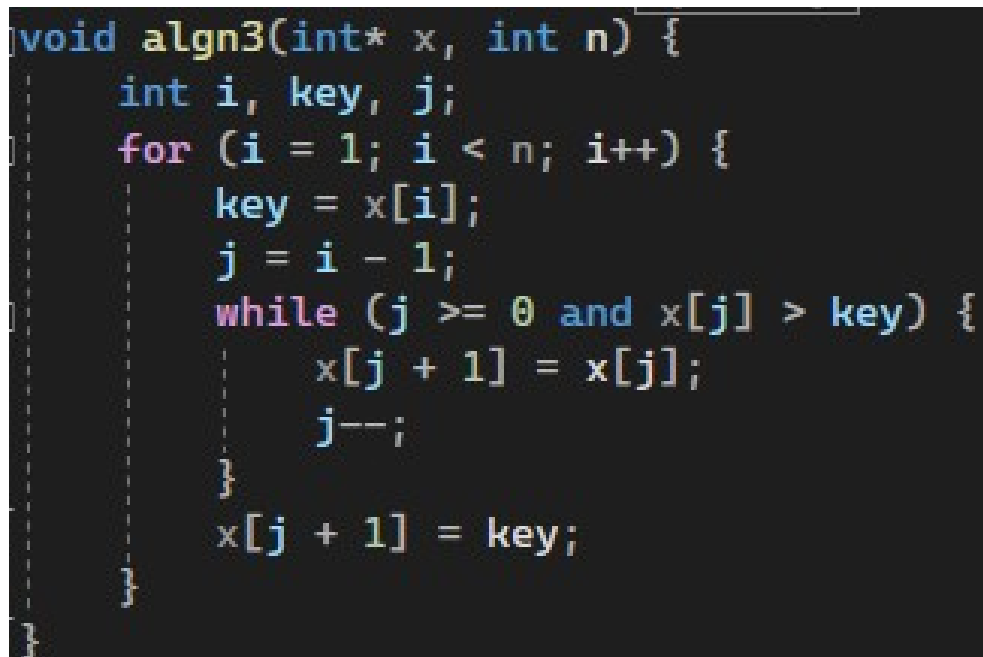
## РЕШЕНИЕ ЗАДАНИЯ 3

### 3.1 ОПИСАНИЕ И РЕАЛИЗАЦИЯ ВТОРОГО АЛГОРИТМА

Сортировка вставками («Insertion Sort») — это простой алгоритм сортировки. Суть его заключается в том, что на каждом шаге алгоритма мы берем один из элементов массива, находим позицию для вставки и вставляем.

Инвариант цикла: после выполнения «i-й» операции все элементы массива с позиции 0 до позиции «i» отсортированы. Размерность массива конечна, следовательно конечен и цикл.

Алгоритм реализован на языке C++. (См. Рисунок 3).

A screenshot of a code editor showing the implementation of the Insertion Sort algorithm in C++. The code is as follows:

```
void algn3(int* x, int n) {  
    int i, key, j;  
    for (i = 1; i < n; i++) {  
        key = x[i];  
        j = i - 1;  
        while (j >= 0 and x[j] > key) {  
            x[j + 1] = x[j];  
            j--;  
        }  
        x[j + 1] = key;  
    }  
}
```

Рисунок 3 – Реализация алгоритма сортировки вставками на C++

### 3.2 АНАЛИЗ ЭФФЕКТИВНОСТИ ВТОРОГО АЛГОРИТМА

Лучший случай: массив изначально отсортирован по возрастанию.  
Функция роста:  $T(n) = n - 1$ .

Средний случай: массив заполнен случайными числами. Функция роста:  
 $T(n) = 0,5n(n-1)$ .

Худший случай: массив отсортирован по убыванию. Функция роста:  $T(n) = n(n-1)$ .

В таблице 3 показаны теоретические и практические результаты вычисления количества совершённых операций и времени работы. На рис. 4 показана графическая визуализация зависимости количества операций от длины массива.

Таблица 3 – Сводная таблица результатов для алгоритма «Insertion Sort»

n	T(n)	$T_n = C_n + M_n$
100	0,0016	6040
1000	0,107	545348
10000	10,187	6085127653
100000	1045,23	497743278328
1000000	Невозможно измерить	Невозможно измерить

Ёмкостная сложность:  $O(n)$ , так как программе необходимо хранить лишь сам массив.

### 3.3 ВЫВОД ОБ ЭМПИРИЧЕСКОЙ ВЫЧИСЛИТЕЛЬНОЙ СЛОЖНОСТИ ВТОРОГО АЛГОРИТМА

Судя по полученным результатам и графику функции роста, зависимость количества операций от длины массива – квадратичная.

Проверим работу алгоритма сортировки вставками на лучшем ( $T_1$ ) и худшем ( $T_2$ ) случае. В таблице 4 описаны полученные результаты.

Таблица 4 – Результаты работы алгоритма в лучшем и худшем случае

n	$T_1(n)$ , мс	$T_2(n)$ , мс	$T_2 = C_2 + M_2$	$T_1 = C_1 + M_1$
100	0,0000003	0,0015	9900	99
1000	0,000045	0,208	999000	999
10000	0,004	22,116	99990000	9999
100000	0,049	2146,94	9999900000	99999
1000000	0,53	251878	999999000000	999999

Чем меньше массив отсортирован, тем больше времени будет затрачено на сортировку данным алгоритмом.

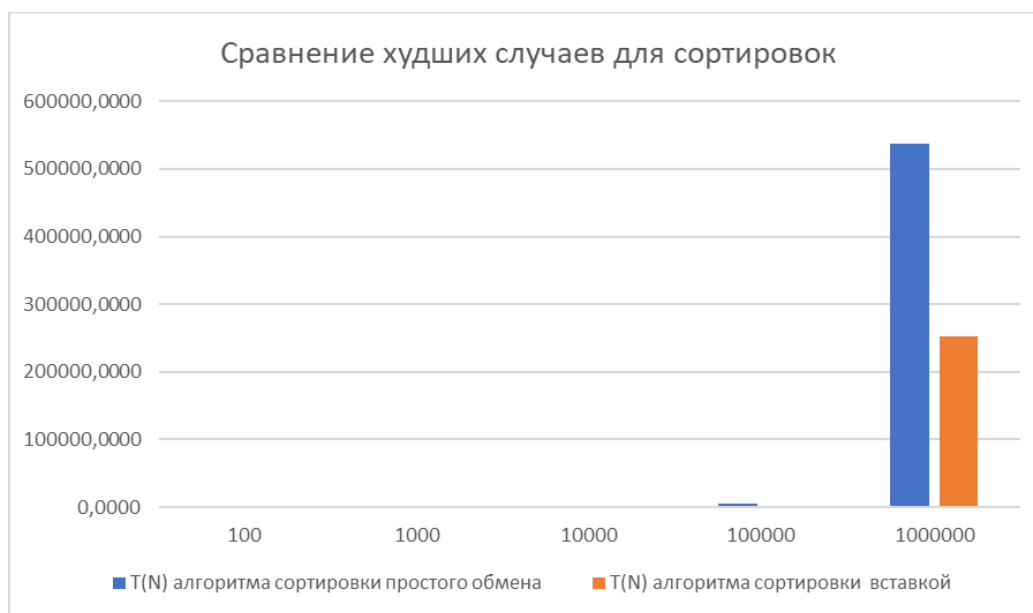


Рисунок 4 – График сравнения худших случаев для сортировок

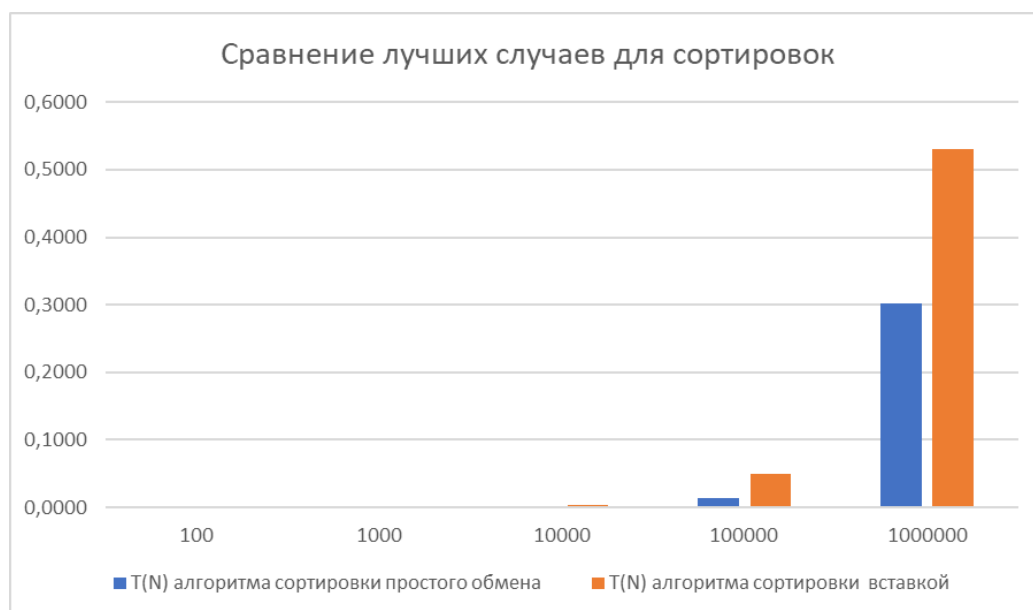


Рисунок 5 – График сравнения лучших случаев для сортировок

## **ВЫВОД**

В процессе выполнения работы мной были освоены навыки определения сложности алгоритма («ёмкостная» и «временная»), реализации алгоритма на языке программирования и сравнение эффективности двух методов сортировок. Основываясь на рис. 4 и таблицах 2 и 4, я могу сделать вывод, что метод сортировки вставками является более эффективным, чем метод сортировки простого обмена.

## **СПИСОК ИНФОРМАЦИОННЫХ ИСТОЧНИКОВ**

1. Структуры данных и проектирование программ : Пер. с англ. / Р. Круз. — М.: БИНОМ. Лаборатория знаний, 2017. — 766 с.
2. Полный справочник по C++ : Пер. с англ. / Г. Шилдт. — М.: ООО "И.Д.Вильямс", 2016. — 796 с.: ил. — Предм. указ.: с. 787-796