

МИНОБРНАУКИ РОССИИ

*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: Боргачев Т.М.

Группа: ИНБО-10-23

Москва – 2024

# СОДЕРЖАНИЕ

[СОДЕРЖАНИЕ 2](#_Toc160051020)

[1 ФОРМУЛИРОВКА ЗАДАЧИ 3](#_Toc160051021)

[1.1 Цель 3](#_Toc160051022)

[1.2 Задание 1 3](#_Toc160051023)

[1.3 Задание 2 4](#_Toc160051024)

[1.4 Задание 3 4](#_Toc160051025)

[2 ПРОЕКТИРОВАНИЕ И РЕАЛИЗАЦИЯ 5](#_Toc160051026)

[2.1 Задание 1 5](#_Toc160051027)

[2.1.1 Реализация алгоритма в виде функции 5](#_Toc160051028)

[2.1.3 Вычислительная сложность алгоритма 7](#_Toc160051034)

[2.1.4 Ёмкостная сложность алгоритма 8](#_Toc160051035)

[2.2 Задание 2 8](#_Toc160051036)

[2.2.1 Оценка вычислительной сложности алгоритма простой сортировки в наихудшем и наилучшем случаях 8](#_Toc160051037)

[2.3 Задание 3 11](#_Toc160051038)

[2.3.1 Разработка алгоритма 11](#_Toc160051039)

[2.3.2 Определение функции роста алгоритма 11](#_Toc160051040)

[2.3.3 Ёмкостная сложность алгоритма 12](#_Toc160051041)

[2.3.4 Эмпирическое исследование второго алгоритма 12](#_Toc160051042)

[2.3.5 Графики функции роста T(n) двух алгоритмов сортировки 13](#_Toc160051043)

[3 ВЫВОДЫ 14](#_Toc160051044)

[4 ИНФОРМАЦИОННЫЕ ИСТОЧНИКИ 15](#_Toc160051045)

# 1 ФОРМУЛИРОВКА ЗАДАЧИ

## Цель

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

## Задание 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 согласно варианту индивидуального задания (2): 1, 2 задание – алгоритм простого обмена («пузырек», Exchange sort); 3 задание – алгоритм простой вставки (Insertion Sort). Провести тестирование программы на исходном массиве n=10.

2. Используя теоретический подход, определить для алгоритма:

a. Что будет ситуациями лучшего, среднего и худшего случаев.

b. 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).

4. Провести эмпирическую оценку вычислительной сложности алгоритма, для чего предусмотреть в программе подсчет фактического количества критических операций как сумму сравнений и перемещений .

5. Построить график функции роста 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.

1.3 Задание 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:

a. строго в убывающем порядке значений.

b. строго в возрастающем порядке значений.

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

## 1.4 Задание 3

Сравнить эффективность алгоритмов простых сортировок

1. Выполнить разработку и программную реализацию второго алгоритма согласно индивидуальному варианту (2).

2. Аналогично заданиям 1 и 2 сформировать таблицы с результатами эмпирического исследования второго алгоритма в среднем, лучшем и худшем случаях (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 (n) двух алгоритмов сортировки в худшем случае.

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

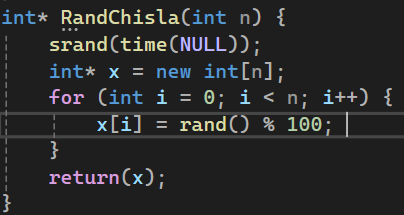
6. Выполнить сравнительный анализ полученных результатов для двух алгоритмов.

# ПРОЕКТИРОВАНИЕ И РЕАЛИЗАЦИЯ

## Задание 1

### 2.1.1 Реализация алгоритма в виде функции

Напишем отдельно функции для заполнения массива случайными значениями рис. 1 и вывода массива на экран рис. 2.



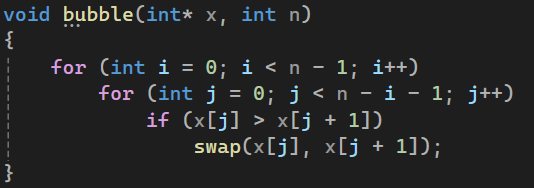
# 

Рисунок 1 – Функция заполнения массива случайными значениями

# 

# 

Рисунок 2 – Функция вывода массива на экран

Функция, реализующая сортировку пузырьком представлена на рис. 3.

# 

Рисунок 3 – Реализация алгоритма в виде функции

Результаты тестирования алгоритма на массиве со случайными элементами размера 10 представлены на рис. 4.

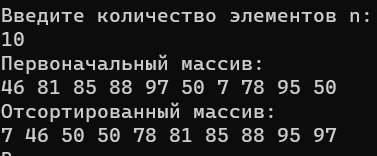


Рисунок 4 – Результаты тестирования алгоритма

Алгоритм работает корректно.

### 2.1.2 Определение функции роста алгоритма

Лучший случай для алгоритма: в массиве только один элемент. Средний случай для алгоритма: массив заполнен случайно. Худший случай: элементы массива расположены по убыванию. Посчитаем количество операций для лучшего и худшего случаев с помощью табл. 1.

Таблица 1 – Подсчет количества операций

|  |  |  |
| --- | --- | --- |
| Оператор | Кол-во выполнений оператора в строке | |
| в лучшем случае | в худшем случае |
| for (int i=0; i<n-1; i++) { | 1 | n |
| for (int j=0; j<n-i-1; j++) { | 0 | () + 1 |
| if (x[j]>x[j+1]) { | 0 | () |
| swap(x[j], x[j+1]) } | 0 | () |
| } |  |  |
| } |  |  |

Таким образом, функция роста алгоритма T(n) в лучшем случае будет равна 1, а в худшем случае T(n) = n + 1 + (3) = n + 1 + (3\*n\*(n-1)/2) = n + 1 + 1.5n^2 – 1.5n =1.5n^2 - 0.5n + 1.

### 2.1.3 Вычислительная сложность алгоритма

Для проведения эмпирической оценкой вычислительной сложности алгоритма заполним и будем использовать табл. 2.

Таблица 2 – Время выполнения алгоритма и количество операций

|  |  |  |
| --- | --- | --- |
| **n** | **T(n), мс** | **=+** |
| 100 | 1 | 9373 |
| 1000 | 2 | 1012653 |
| 10000 | 216 | 99673661 |
| 100000 | 28571 | 1380270839 |
| 1000000 | 2984380 | 15302708390 |

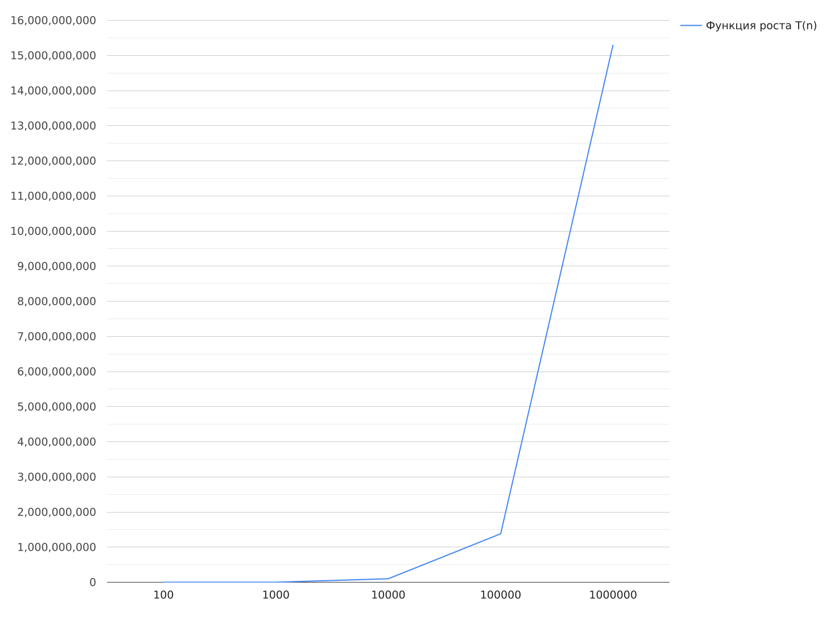
График зависимости функции роста T(n) от значений n представлен на рис. 5.

Рисунок 5 – График зависимости функции роста от n

### 2.1.4 Ёмкостная сложность алгоритма

Так как не требуется дополнительного массива для решения задачи, то требуется только один массив длиной n (ёмкостная сложность n).

## Задание 2

### 2.2.1 Оценка вычислительной сложности алгоритма простой сортировки в наихудшем и наилучшем случаях

Результаты тестирования алгоритма в наихудшем и наилучшем случаях представлены соответственно в табл. 3 и 4.

Таблица 3 – Тестирование алгоритма в наихудшем случае

|  |  |  |
| --- | --- | --- |
| **n** | **T(n), мс** | **=+** |
| 100 | 1 | 14861 |
| 1000 | 3 | 1489457 |
| 10000 | 300 | 148997699 |
| 100000 | 34609 | 2015051561 |
| 1000000 | 3434910 | 2999992000005 |

Таблица 4 – Тестирование алгоритма в наилучшем случае

|  |  |  |
| --- | --- | --- |
| **n** | **T(n), мс** | **=+** |
| 100 | 1 | 5049 |
| 1000 | 2 | 500499 |
| 10000 | 72 | 50004999 |
| 100000 | 7201 | 705082703 |
| 1000000 | 766918 | 1784293663 |

Код, реализующий сортировку отсортированных массивов представлен на рис. 6.

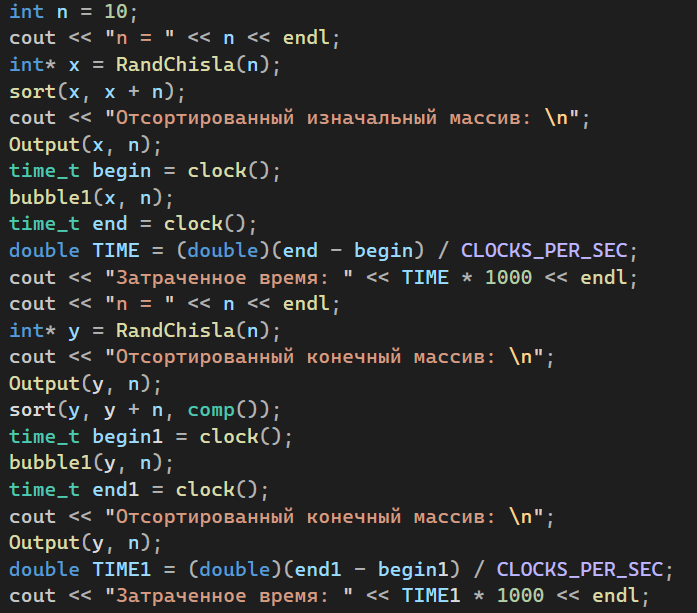


Рисунок 6 – Код сортировки лучшего и худшего случаев

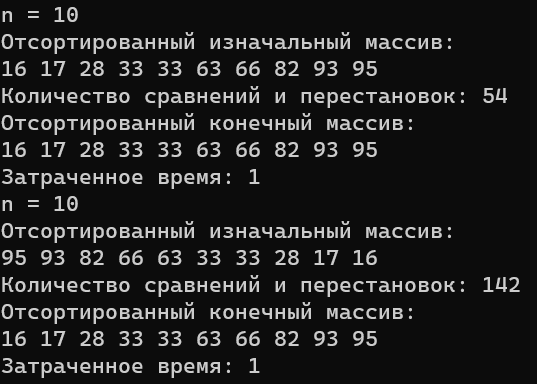
Результат работы кода представлен на рис. 7.

Рисунок 7 – Результат работы алгоритма в лучшем и худшем случаях

Таким образом, можно утверждать, что время работы алгоритма линейно зависит от исходной упорядоченности массива.

## Задание 3

### 2.3.1 Разработка алгоритма

Код на языке программирования C++, реализующей алгоритм сортировки простой вставки представлен на рис. 8.

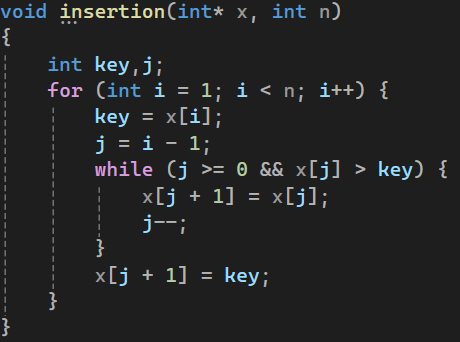


Рисунок 8 – Алгоритм простой вставки

Результат тестирования алгоритма при n=10 представлены на рис.9.

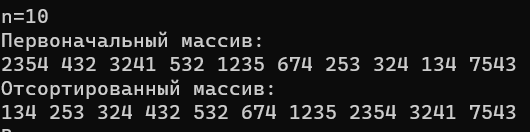


Рисунок 9 – Тест алгоритма

Можно сделать вывод о корректной работе алгоритма.

### 2.3.2 Определение функции роста алгоритма

Рассчитаем функцию роста алгоритма, определив количество выполняемых операций с помощью табл. 5.

Таблица 5 - Количество операций алгоритма

|  |  |  |
| --- | --- | --- |
| Оператор | Кол-во выполнений оператора в строке | |
| в лучшем случае | в худшем случае |
| int key, j; | 2 | 2 |
| for (int i = 1; i < n; i++) { | 1 | n |
| key = x[i]; | 0 | n - 1 |
| Оператор | Кол-во выполнений оператора в строке | |
| в лучшем случае | в худшем случае |
| j = i - 1; | 0 | n - 1 |
| while (j >= 0 && x[j] > key) { | 0 | ( |
| x[j + 1] = x[j]; | 0 | ( |
| j--} | 0 | ( |
| x[j + 1] = key;} | 0 | n-1 |

Таким образом, функция роста в худшем случае T(n) = 4n-1 + (n-2)\*(n-1)/2= 4n – 1 + 0.5n^2 -1.5n + 1 = 0.5n^2 +2.5n, а в лучшем случае T(n) = 3.

### 2.3.3 Ёмкостная сложность алгоритма

Так как не требуется дополнительного массива для решения задачи, то требуется только один массив длиной n (ёмкостная сложность n).

### 2.3.4 Эмпирическое исследование второго алгоритма

Результаты тестирования алгоритма в среднем, наилучшем и наихудшем случаях представлены соответственно в табл. 5, 6 и 7.

Таблица 5 – Тестирование алгоритма в среднем случае

|  |  |  |
| --- | --- | --- |
| **n** | **T(n), мс** | **=+** |
| 100 | 1 | 5366 |
| 1000 | 1 | 499076 |
| 10000 | 43 | 49605128 |
| 100000 | 4198 | 648864620 |
| 1000000 | 428095 | 1228144492 |

Таблица 6 – Тестирование алгоритма в наилучшем случае

|  |  |  |
| --- | --- | --- |
| **n** | **T(n), мс** | **=+** |
| 100 | 1 | 396 |
| 1000 | 1 | 3996 |
| 10000 | 1 | 39996 |
| 100000 | 1 | 399996 |
| **n** | **T(n), мс** | **=+** |
| 1000000 | 3 | 3999996 |

Таблица 7 – Тестирование алгоритма в наихудшем случае

|  |  |  |
| --- | --- | --- |
| **n** | **T(n), мс** | **=+** |
| 100 | 1 | 10202 |
| 1000 | 1 | 993240 |
| 10000 | 80 | 99030152 |
| 100000 | 8425 | 1310373614 |
| 1000000 | 856645 | 250002500000 |

### 2.3.5 Графики функции роста T(n) двух алгоритмов сортировки

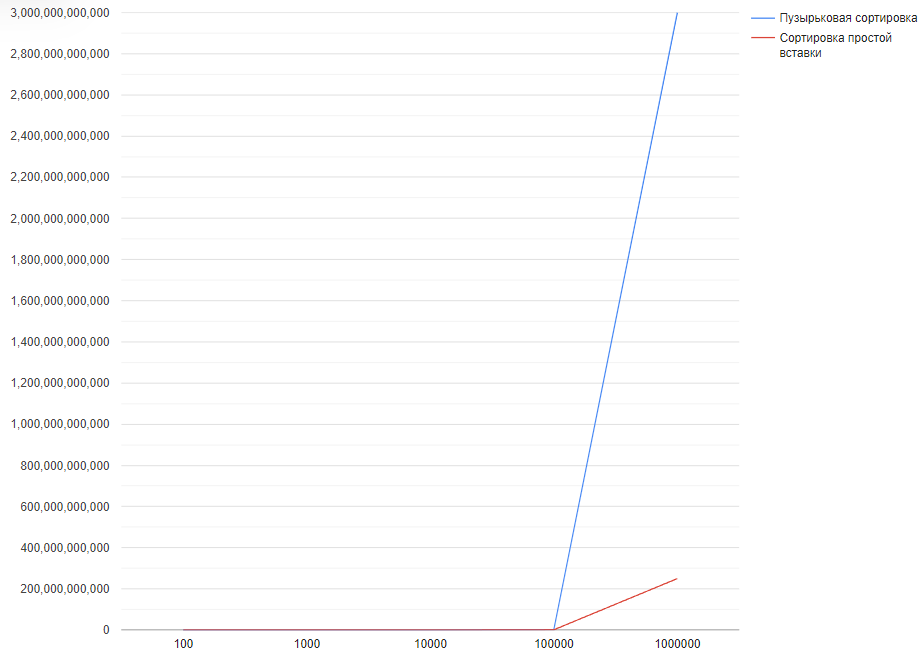
График функций роста алгоритмов в худшем случае представлен на рис. 10.

Рисунок 10 – Графики функций роста алгоритмов в худшем случае

График функций роста алгоритмов в лучшем случае представлен на рис. 11.

# 3 ВЫВОДЫ

Рисунок 11 - Графики функций роста алгоритмов в лучшем случае

На основе скорости роста функции роста первого и второго алгоритма, можно сделать вывод о том, что их эмпирическая вычислительная сложность квадратичная.

Время работы обоих алгоритмов также линейно зависит от того, насколько отсортирован первоначальный массив.

На основе графиков функций роста двух алгоритмов, можно сделать вывод о том, что алгоритм пузырьковой сортировки как в лучшем, так и в худшем случае требует все большей вычислительной мощности с ростом значения n, нежели алгоритм простой вставки.

Таким образом, алгоритм сортировки простой вставки эффективнее алгоритма пузырьковой сортировки.

# 4 ИНФОРМАЦИОННЫЕ ИСТОЧНИКИ

1. Сартаков М.В., ПР-1.1 (Теоретическая сложность алгоритма) М., МИРЭА — Российский технологический университет – 12 с. - URL: <https://online-edu.mirea.ru/pluginfile.php?file=%2F1042738%2Fmod_assign%2Fintroattachment%2F0%2FПР1.1%20%28Теоретическая%20сложность%20алгоритма%29.pdf&amp;forcedownload=1> (дата обращения: 15.02.2024). - Режим доступа: Электронно-облачная система – Cloud MIREA РТУ МИРЭА. - Текст: электронный.
2. Рысин М.Л., Сартаков М.В., Туманова М.Б., Введение в структуры и алгоритмы обработки данных. Ч. 1 - учебное пособие, 2022, МИРЭА – Российский технологический университет. – 2022, 109с. – URL: [file:///C:/Users/borga/Downloads/Рысин%20М.Л.%20и%20др.%20Введение%20в%20структуры%20и%20алгоритмы%20обработки%20данных.%20Ч.%201%20-%20учебное%20пособие,%202022.pdf](C://Users/borga/Downloads/Рысин%20М.Л.%20и%20др.%20Введение%20в%20структуры%20и%20алгоритмы%20обработки%20данных.%20Ч.%201%20-%20учебное%20пособие,%202022.pdf) (дата обращения: 15.02.2024 ). – Режим доступа: Электронно-облачная система – Cloud MIREA РТУ МИРЭА. - Текст: электронный.