Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

Лабораторная работа №3
по дисциплине "Анализ Алгоритмов"

Тема Алгоритмы сортировки

Студент Рядинский К. В.

Группа ИУ7-53Б

Преподаватель Волкова Л. Л.

Москва

2021 г.

СОДЕРЖАНИЕ

Введение

Целью данной лабораторной работы является изучить и реализовать алгоритмы сортировки, оценить их трудоемкость.

Алгоритмы сортировки имеют широкое практическое применение. Сортировки используются в большом спектре задач, включая обработку коммерческих, сейсмических, космических данных. Часто сортировка является просто вспомогательной операцией для упорядочивания данных, упрощения последующих алгебраических действий над данными.

Сортировка применяется во многих областях программирования, например, базы данных или математические программы. Упорядоченные объекты содержатся в телефонных книгах, ведомостях налогов, в библиотеках, в оглавлениях, в словарях.

Во многих вычислительных системах на сортировку уходит больше половины машинного времени. Исходя из этого, можно заключить, что либо сортировка имеет много важных применений, либо ею часто пользуются без нужды, либо применяются в основном неэффективные алгоритмы сортировки.

В настоящее время, в связи с экспоненциально возросшими объемами данных, вопрос эффективной сортировки данных снова стал актуальным. В настоящее время в сети Интернет можно найти результаты производительности алгоритмов сортировки для ряда ведущих центров данных. При этом используются различные критерии оценки эффективности.

Задачи лабораторной работы:

1. Рассмотреть и изучить сортировки пузырьком, вставками и выбором.
2. Расчитать их трудоемкость.
3. Реализовать каждую из этих сортировок.
4. Сравнить их временные характеристики экспериментально.
5. На основании проделанной работы сделать выводы.

1 Аналитическая часть

В данной главе будут рассмотрены алгоритмы сортировки.

1.1 Сортировка пузырьком

Алгоритм состоит из повторяющихся проходов по сортируемому массиву. За каждый проход элементы последовательно сравниваются попарно и, если порядок в паре неверный (возрастание, в случае сортировки по убыванию, и наоборот), выполняется обмен элементов. Проходы по массиву повторяются $N - 1$ раз, но есть модифицированная версия, где если окажется, что обмены больше не нужны, значит проходы прекращаются. При каждом проходе алгоритма по внутреннему циклу очередной наибольший элемент массива ставится на свое место в конце массива рядом с предыдущим «наибольшим элементом», а наименьший элемент массива перемещается на одну позицию к началу массива.

1.2 Сортировка вставками

Сортировка вставками — алгоритм сортировки, в котором элементы входной последовательности просматриваются по одному, и каждый новый поступивший элемент размещается в подходящее место среди ранее упорядоченных элементов.

В начальный момент отсортированная последовательность пуста. На каждом шаге алгоритма выбирается один из элементов входных данных и помещается на нужную позицию в уже отсортированной последовательности до тех пор, пока набор входных данных не будет исчерпан. В любой момент времени в отсортированной последовательности элементы удовлетворяют требованиям к выходным данным алгоритма.

1.3 Сортировка выбором

Шаги алгоритмы:

1. Находится номер минимального значения в текущем списке.

2. Производится обмен этого значения со значением первой неотсортированной позиции (обмен не нужен, если минимальный элемент уже находится на данной позиции).
3. Сортируется хвост списка, исключиая при этом из рассмотрения уже отсортированные элементы.

Для реализации устойчивости алгоритма необходимо в пункте 2. минимальный элемент непосредственно вставлять в первую неотсортированную позицию, не меняя порядок остальных элементов.

1.3 Вывод

В данной работе стоит задача реализации 3 алгоритмов сортировки, а именно: обычный алгоритм пузырька, вставками и выбором. Необходимо оценить трудоем- кость алгоритмов и проверить ее экспериментально.

2 Конструкторская часть

В данном разделе будут представлены схемы алгоритмов и рассчитана их трудоемкость, а также требования к программному обеспечению (далее – ПО).

2.1 Требования к ПО

Требования к вводу

1. На вход подается массив сравнимых элементов.
2. На выходе — отсортированный массив в заданном порядке, сортировка производится на месте, то есть сортируется тот же самый массив и он же возвращается.
3. Алгоритм сортирует целочисленный тип данных t $-2^{64} < t < 2^{64} - 1$.

.

2.2 Трудоемкость алгоритмов

В данном разделе будет введена модель вычислений и рассмотрены трудоемкости алгоритмов сортировки выбором, вставки и пузырьком.

2.2.1 Модель вычислений

Для последующего вычисления трудоемкости необходимо ввести модель вычислений [1].

1. $+, -, /, \%, =, \neq, <, >, \leq, \geq, [], *$ — трудоемкость 1.
2. Трудоемкость оператора выбора *if* условие *then A else B* рассчитывается, как:

$$f_{if} = f_{\text{условия}} + \begin{cases} f_A & \text{если условие выполняется,} \\ f_B & \text{иначе.} \end{cases} \quad (1)$$

3. Трудоемкость цикла расчитывается, как:

$$f_{for} = f_{\text{инициализации}} + f_{\text{сравнения}} + N(f_{\text{тела}} + f_{\text{инициализации}} + f_{\text{сравнения}}). \quad (2)$$

4. Трудоемкость вызова функции равна 0.

2.3 Расчет трудоемкости

В данном разделе будут расчитаны трудоемкости алгоритмов методом пузырька, вставками и выбором.

2.3.1 Вычисление трудоёмкости алгоритма сортировки пузырьком

Лучший случай: массив отсортирован, следовательно не произошло ни одного обмена.

Трудоемкость:

$$T(N) = 2 + \frac{N \cdot (N-1)}{2} \cdot 3 = 1.5N^2 - 1.5N + 2 \tag{3}$$

Худший случай: массив отсортирован в обратном порядке, в каждом случае происходил обмен.

Трудоемкость:

$$T(N) = 2 + \frac{N \cdot (N-1)}{2} \cdot 8 = 4N^2 - 4N + 2 \tag{4}$$

2.3.2 Вычисление трудоёмкости алгоритма сортировки вставками

Лучший случай: массив отсортирован, все внутренние циклы состоят всего из одной итерации.

Трудоемкость:

$$T(N) = 2 + 6N \tag{5}$$

Худший случай: массив отсортирован в обратном порядке. Каждый новый элемент сравнивается со всеми в отсортированной последовательности. Все внутренние циклы будут состоять из j итераций.

Трудоемкость.

$$T(N) = \frac{N \cdot (N-1)}{2} \cdot 10 + N + 2 = 5N^2 - 4N + 2 \quad (6)$$

2.3.3 Вычисление трудоёмкости алгоритма сортировки выбором

Лучший случай: массив отсортирован.

Трудоемкость:

$$T(N) = 2 + \frac{N \cdot (N-1)}{2} \cdot 8 + 4N = 4N^2 + 2 \quad (7)$$

Худший случай: массив отсортирован в обратном порядке, в каждом случае происходил обмен.

Трудоемкость:

$$T(N) = \frac{N \cdot (N-1)}{2} \cdot 5 + 5 \cdot N + 3 = 2.5N^2 + 2.5N + 3 \quad (8)$$

2.4 Схемы алгоритмов

На рисунке 1 изображена схема реализации алгоритма соритровки методом пузырька.

На рисунке 2 изображена схема реализации алгоритма соритровки методом вставок.

На рисунке 3 изображена схема реализации алгоритма соритровки методом выбора.

2.5 Вывод

По аналитическим расчетам можно сделать вывод, что в лучшем случае быстрее всего отработает алгоритм вставками. А в худшем случае быстрее всего отработает алгоритм пузырька.

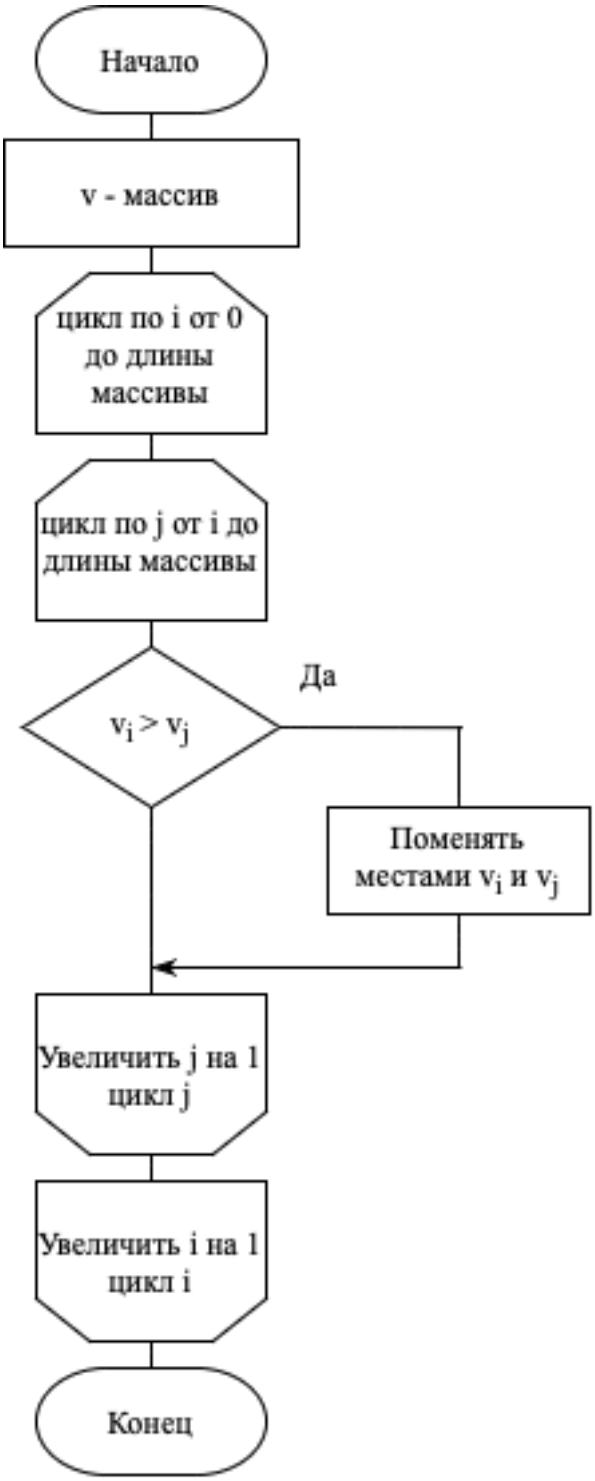

Рис. 1 — Схема алгоритма сортировки пузырьком

Рис. 2 — Схема алгоритма сортировки вставками

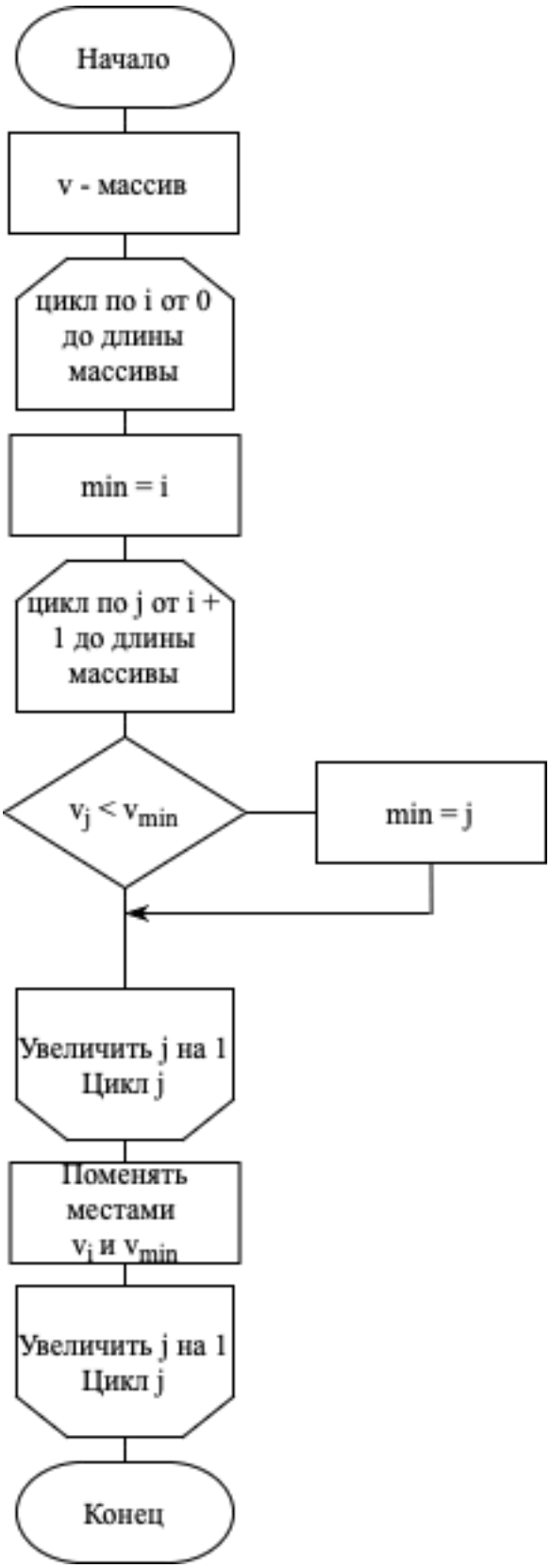

Рис. 3 — Схема алгоритма сортировки выбором

3 Технологическая часть

В данном разделе будет представлен выбор языка программирования и листинги реализаций алгоритмов.

3.1 Выбор языка программирования

Для реализации программ я выбрал язык программирования Rust [2], так как этот язык предоставляет как низкоуровневые интерфейсы, так и высокоуровневые. Также он является безопасным языком. Среда разработки Visual Studio Code. Также данный язык был выбран потому, что в нем присутствует инструментарий для замера процессорного времени и тестирования.

3.2 Листинги реализации алгоритма

В листингах 1 - 3 представлены реализации трех алгоритмов сортировки.

Листинг 1: Функция сортировки массива методом пузырька

```
pub fn bubble_sort<T: std::cmp::Ord>(v: &mut [T]) {
    for i in 0..v.len() {
        for j in i..v.len() {
            if v[i] > v[j] {
                v.swap(i, j);
            }
        }
    }
}
```

Листинг 2: Функция сортировки массива методом вставок

```
pub fn insertion_sort<T: std::cmp::Ord>(v: &mut [T]) {
    for i in 1..v.len() {
        let mut j = i;
        while j > 0 && v[j] < v[j - 1] {
            v.swap(j, j - 1);
            j = j - 1;
        }
    }
```

```
9 }
```

Листинг 3: Функция сортировки массива методом выбора

```
1  pub fn selection_sort<T: std::cmp::Ord>(v: &mut [T]) {
2      let mut min;
3      for i in 0..v.len() {
4          min = i;
5          for j in (i + 1)..v.len() {
6              if v[j] < v[min] {
7                  min = j;
8              }
9          }
10         v.swap(i, min);
11     }
12 }
```

3.3 Тестирование

Тестирование проводилось встроенной библиотекой модульного тестирования в систему сборки Cargo.

В таблице 1 приведены результаты тестирования.

Таблица 1 — Результаты функционального тестирования.

Вход	Результат	Ожидаемый результат
1, 2, 3, 4	1, 2, 3, 4	1, 2, 3, 4
4, 3, 2, 1	1, 2, 3, 4	1, 2, 3, 4
1, 3, 2, 4	1, 2, 3, 4	1, 2, 3, 4
пустой	пустой	пустой

3.3 Вывод

В данном разделе были представлены требования к программному обеспечению, выбор языка программирования, листинги реализаций алгоритмов и результаты тестирования.

4 Исследовательская часть

В данном разделе будут представлены замеры времени работы реализаций алгоритмов.

4.1 Сравнительный анализ на основе замеров времени работы алгоритмов

Был проведен замер времени работы каждого из алгоритмов с помощью библиотеки Criterion [3]. Эта библиотека замеряет процессорное время выполнения функции и усредняет его.

Таблица 2 — Результаты времени выполнения алгоритмов сортировок на отсортированном массиве (в секундах)

Размер массива	Пузырек	Вставки	Выбором
10	5,50E-08	3,02E-08	6,20E-08
100	2,18E-06	1,07E-07	7,16E-06
500	4,44E-05	3,83E-07	2,24E-04
1000	1,69E-04	7,57E-07	9,34E-04

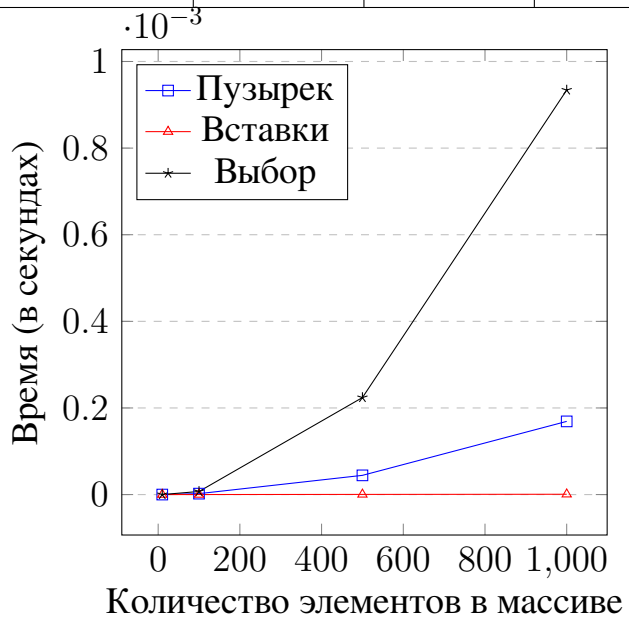

Рис. 4 — График времени выполнения алгоритмов при отсортированном массиве

Таблица 3 — Результаты времени выполнения алгоритмов сортировок на отсортированном в обратном порядке массиве (в секундах)

Размер массива	Пузырек	Вставки	Выбором
10	6,80E-08	8,14E-08	1,22E-07
100	2,54E-06	8,00E-06	7,55E-06
500	5,66E-05	2,01E-04	2,27E-04
1000	2,16E-04	8,25E-04	9,33E-04

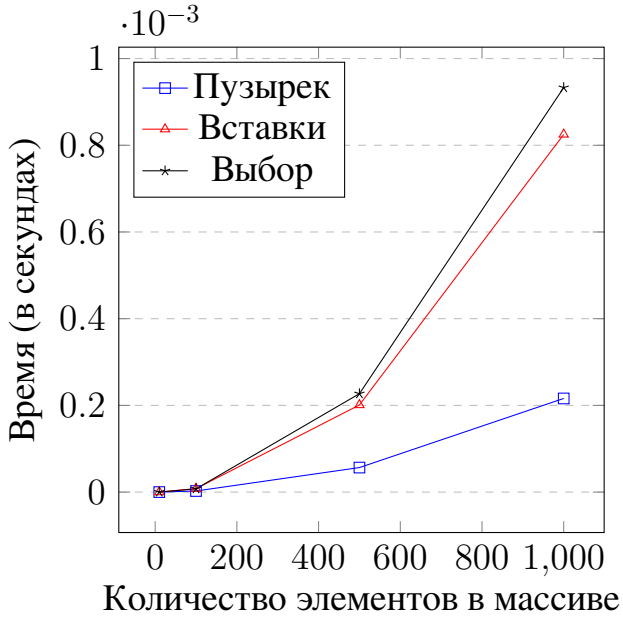

Рис. 5 — График времени выполнения алгоритмов при отсортированном в обратном порядке массиве

Таблица 4 — Результаты времени выполнения алгоритмов сортировок на случайном массиве (в секундах)

Размер массива	Пузырек	Вставки	Выбором
100	2,91E-06	3,75E-06	8,72E-06
1000	3,32E-04	4,11E-04	9,48E-04
2500	2,02E-03	2,55E-03	5,99E-03
5000	8,27E-03	1,02E-02	2,43E-02

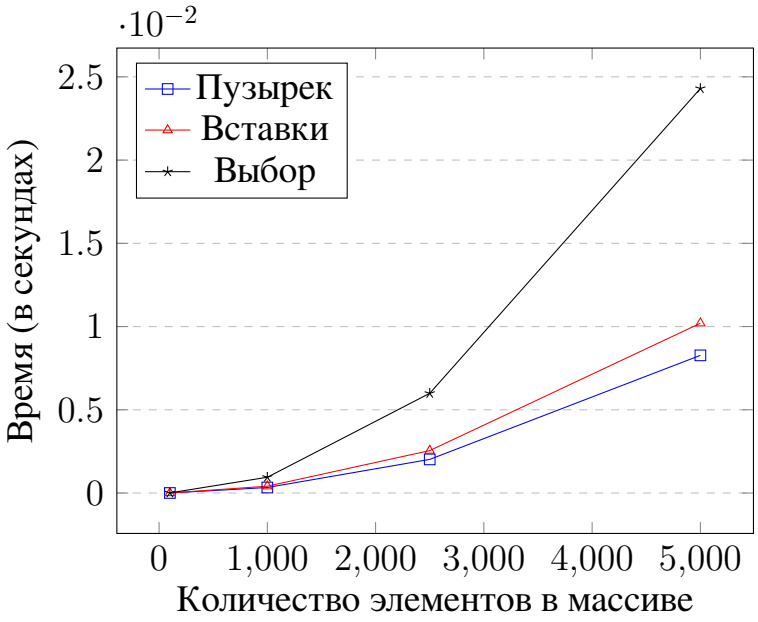

Рис. 6 — График времени выполнения алгоритмов при случайном массиве

4.1 Вывод

По результатам тестирования выявлено, что все рассматриваемые алгоритмы реализованы правильно. Самым быстрым алгоритмом при использовании случайного массива оказался алгоритм сортировки методом пузырька. Метод выбора является самый медленным на случайном наборе данных. Теоретические данные совпали с экспериментальными.

Заключение

В рамках данной лабораторной работы была достигнута цель и выполнены следующие задачи:

— были изучены и реализованы три алгоритма сортировки: методом пузырька, вставок и выбором;

— был проведен сравнительный анализ трудоемкости алгоритмов на основе теоретических расчетов и выбранной модели вычислений;

— был проведен сравнительный анализ алгоритмов на основе экспериментальных данных.

Экспериментально были установлены различия в производительности различных алгоритмов сортировки. Для массивов длины 5000, заполненной случайными данными, метод пузырька быстрее вставок и выбора.

Литература

1. Блэнди Дж., Орендорф Дж. Программирование на языке Rust = Programming Rust. — ДМК Пресс, 2018. — 550 с. — ISBN 978-5-97060-236-2.
2. Criterion.rs - Statistics-driven benchmarking library for Rust [Электронный ресурс] https://github.com/bheisler/criterion.rs (дата обращения: 14 октября 2021 г.)
3. LPDDR4 [Электронный ресурс] https://ru.wikipedia.org/wiki/LPDDR#LPDDR4 (дата обращения: 14 октября 2021 г.)