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»

Отчет по лабораторной работе №3 по курсу "Анализ алгоритмов"

Тема Алгоритмы сортировки

Студент Пересторонин П.Г.

Группа ИУ7-53Б

Преподаватели Волкова Л.Л., Строганов Ю.В.

Москва — 2020 г.

Оглавление

Введение

Одной из важнейших процедур обработки структурированной информации является сортировка [1]. Сортировкой называют процесс перегруппировки заданной последовательности (кортежа) объектов в некотором определенном порядке. Определенный порядок (например, упорядочение в алфавитном порядке, по возрастанию или убыванию количественных характеристик, по классам, типам и.т.п.) в последовательности объектов необходимо для удобства работы с этим объектом . В частности, одной из целей сортировки является облегчение последующего поиска элементов в отсортированном множестве.

Алгоритмы сортировки используются практически в любой программной системе. Целью алгоритмов сортировки является упорядочение последовательности элементов данных. Поиск элемента в последовательности отсортированных данных занимает время, пропорциональное логарифму количеству элементов в последовательности, а поиск элемента в последовательности не отсортированных данных занимает время, пропорциональное количеству элементов в последовательности, то есть намного больше. Существует множество различных методов сортировки данных. Однако любой алгоритм сортировки можно разбить на три основные части:

- сравнение, определяющее упорядоченность пары элементов;
- перестановка, меняющая местами пару элементов;
- собственно сортирующий алгоритм, который осуществляет сравнение и перестановку элементов данных до тех пор, пока все эти элементы не будут упорядочены.

Важнейшей характеристикой любого алгоритма сортировки является скорость его работы, которая определяется функциональной зависимостью среднего времени сортировки последовательностей элементов данных, заданной длины, от этой длины. Время сортировки будет пропорционально количеству сравнений и перестановки элементов данных в процессе их сортировки.

Как уже было сказано, в любой сфере, использующей какое-либо программное обеспечение, с большой долей вероятности используются сорти-

ровки. К примеру, на сайте [2] можно найти результаты производительности алгоритмов сортировки для ряда ведущих центров данных. При этом используются различные критерии оценки эффективности.

Задачи лабораторной работы:

- изучить и реализовать 3 алгоритма сортировки: пузырёк, вставками, выбором;
- провести сравнительный анализ трудоёмкости алгоритмов на основе теоретических расчетов и выбранной модели вычислений;
- провести сравнительный анализ алгоритмов на основе экспериментальных данных;
- подготовить отчета по лабораторной работе.

1 Аналитическая часть

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, 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 (“всплывает” до нужной позиции, как пузырёк в воде – отсюда и название алгоритма).

1.2 Сортировка вставками

Сортировка вставками — алгоритм сортировки, котором элементы входной последовательности просматриваются по одному, и каждый новый поступивший элемент размещается в подходящее место среди ранее упорядоченных элементов [1].

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 алгоритма:

1. находим номер минимального значения в текущем массиве;

2. производим обмен этого значения со значением первой неотсортированной позиции (обмен не нужен, если минимальный элемент уже находится на данной позиции);

3. теперь сортируем “хвост” массива, исключив 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, что может привести к резкому увеличению числа обменов.

Вывод

В данной работе стоит задача реализации 3 алгоритмов сортировки, а именно: пузырьком, вставками и выбором. Необходимо оценить теоретическую оценку алгоритмов и проверить ее экспериментально.

2 Конструкторская часть

2.1 Разработка алгоритмов

На рисунках 2.1, 2.2 и 2.3 представлены схемы алгоритмов сортировки пузырьком, выбором и вставками соответственно.

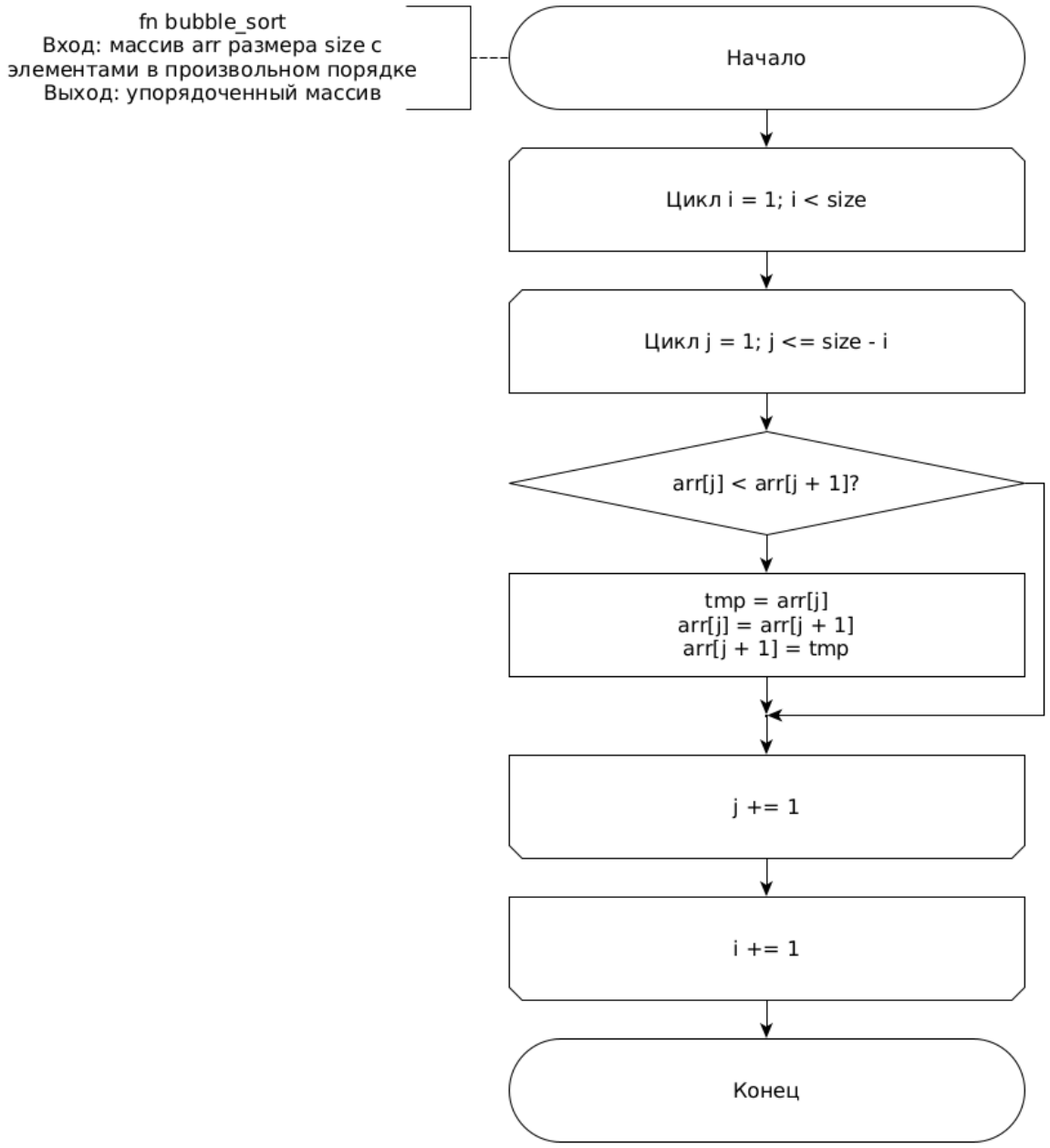

Рис. 2.1: Схема алгоритма сортировки пузырьком

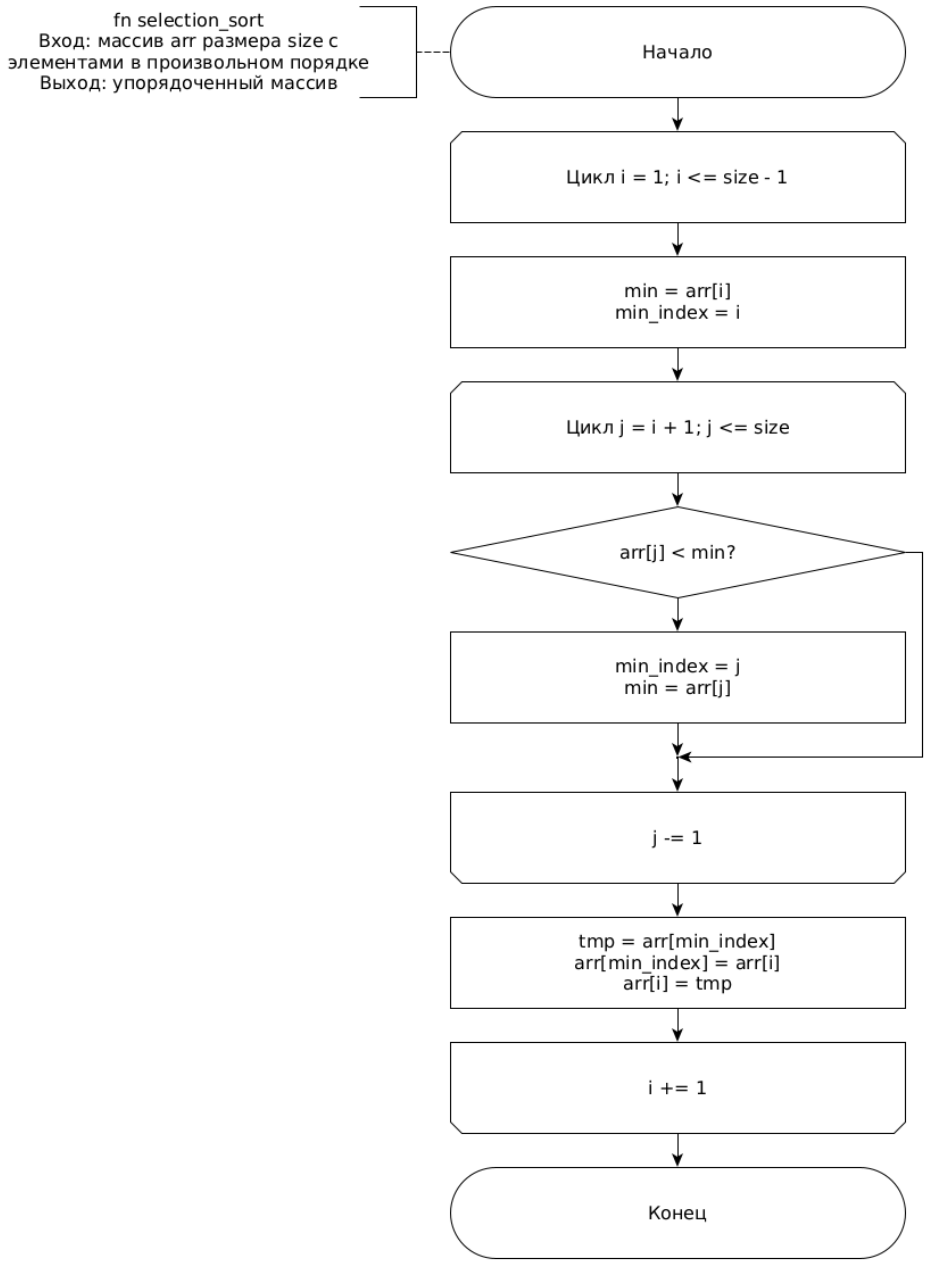

Рис. 2.2: Схема алгоритма сортировки выбором

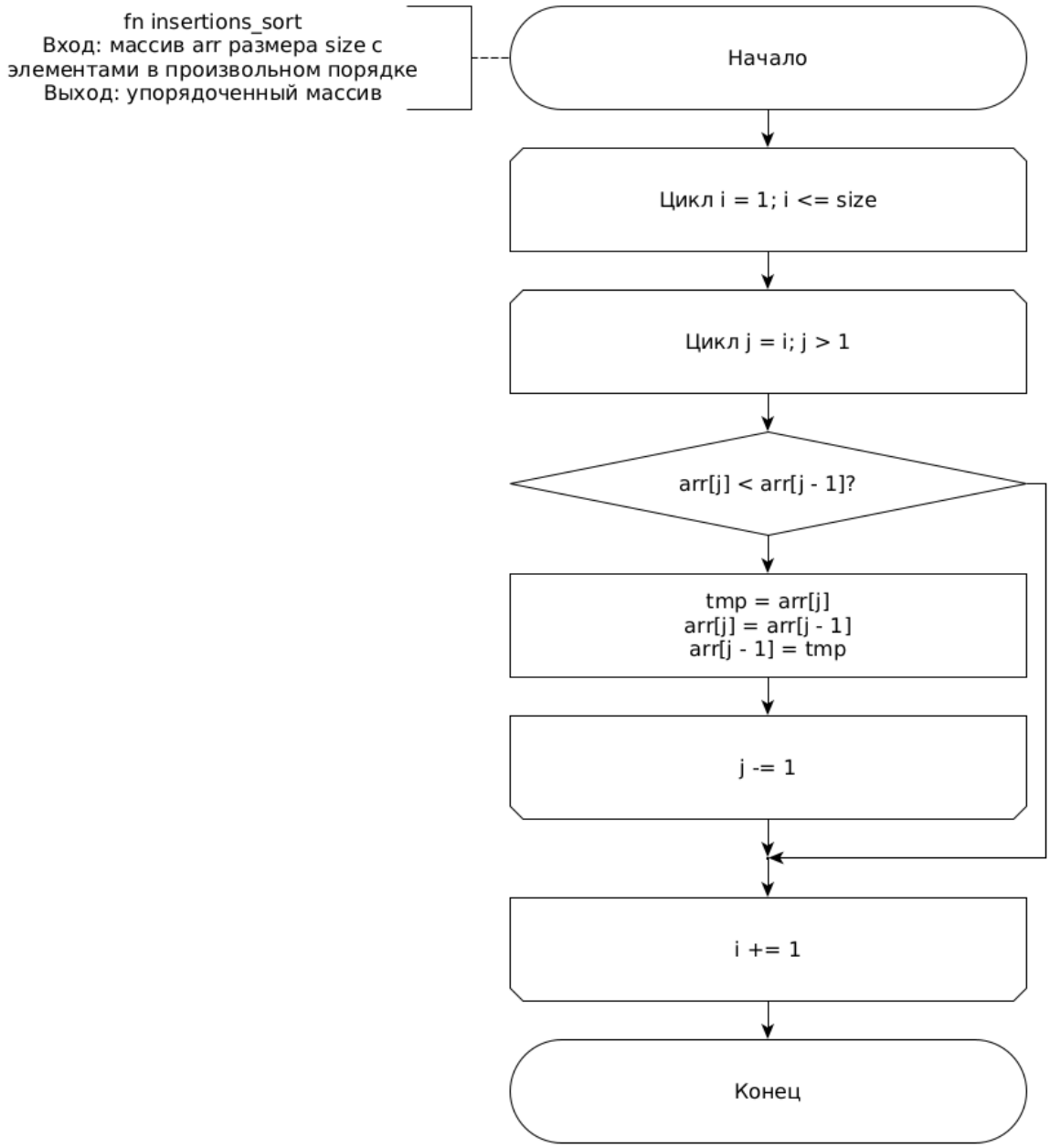

Рис. 2.3: Схема алгоритма сортировки вставками

2.2 Модель вычислений

Для последующего вычисления трудоемкости необходимо ввести модель вычислений:

1. операции из списка (2.1) имеют трудоемкость 1;

$$+, -, /, \%, ==, !=, <, >, <=, >=, [], ++, -- \tag{2.1}$$

2. трудоемкость оператора выбора `if условие then A else B` рассчитывается, как (2.2);

$$f_{if} = f_{\text{условия}} + \begin{cases} f_A, & \text{если условие выполняется,} \\ f_B, & \text{иначе.} \end{cases} \tag{2.2}$$

3. трудоемкость цикла рассчитывается, как (2.3);

$$f_{for} = f_{\text{инициализации}} + f_{\text{сравнения}} + N(f_{\text{тела}} + f_{\text{инкремента}} + f_{\text{сравнения}}) \tag{2.3}$$

4. трудоемкость вызова функции равна 0.

2.3 Трудоёмкость алгоритмов

Обозначим во всех последующих вычислениях размер массивов как N.

2.3.1 Алгоритм сортировки пузырьком

Трудоёмкость алгоритма сортировки пузырьком состоит из:

- Трудоёмкость сравнения и инкремента внешнего цикла по $i \in [1..N)$, которая равна (2.4):

$$f_{outer} = 2 + 2(N-1) \tag{2.4}$$

- Суммарная трудоёмкость внутренних циклов, количество итераций которых меняется в промежутке $[1..N-1]$, которая равна (2.5):

$$f_{inner} = 3(N-1) + \frac{N \cdot (N-1)}{2} \cdot (3 + f_{if}) \tag{2.5}$$

- Трудоёмкость условия во внутреннем цикле, которая равна (2.6):

$$f_{if} = 4 + \begin{cases} 0, & \text{л.с.} \\ 9, & \text{х.с.} \end{cases} \tag{2.6}$$

Трудоёмкость в лучшем случае (2.7):

$$f_{best} = -3 + \frac{3}{2}N + \frac{7}{2}N^2 \approx \frac{7}{2}N^2 = O(N^2) \tag{2.7}$$

Трудоёмкость в худшем случае (2.8):

$$f_{worst} = -3 - 8N + 8N^2 \approx 8N^2 = O(N^2) \tag{2.8}$$

2.3.2 Алгоритм сортировки выбором

Трудоёмкость алгоритма сортировки выбором состоит из:

- Трудоёмкость сравнения, инкремента внешнего цикла, а также зависимых только от него операций, по $i \in [1..N)$, которая равна (2.9):

$$f_{outer} = 2 + 12(N - 1) \tag{2.9}$$

- Суммарная трудоёмкость внутренних циклов, количество итераций которых меняется в промежутке $[1..N - 1]$, которая равна (2.10):

$$f_{inner} = 2(N - 1) + \frac{N \cdot (N - 1)}{2} \cdot f_{if} \tag{2.10}$$

- Трудоёмкость условия во внутреннем цикле, которая равна (2.11):

$$f_{if} = 3 + \begin{cases} 0, & \text{л.с.} \\ 3, & \text{х.с.} \end{cases} \tag{2.11}$$

Трудоёмкость в лучшем случае (2.12):

$$f_{best} = -12 + 12.5N + \frac{3}{2}N^2 \approx \frac{3}{2}N = O(N^2) \tag{2.12}$$

Трудоёмкость в худшем случае (2.13):

$$f_{worst} = -12 + 11N + 3N^2 \approx 3N^2 = O(N^2) \tag{2.13}$$

2.3.3 Алгоритм сортировки вставками

Трудоёмкость сортировки вставками может быть рассчитана таким же образом, что и трудоёмкости алгоритмов выше. Асимптотическая трудоёмкость алгоритма сортировки вставками $O(N^2)$ в худшем случае и $O(N)$ — в лучшем.

Вывод

Были разработаны схемы всех трех алгоритмов сортировки. Для каждого из них были рассчитаны и оценены лучшие и худшие случаи.

3 Технологическая часть

В данном разделе приведены средства реализации и листинг кода.

3.1 Требования к ПО

К программе предъявляется ряд требований:

- на вход подаётся массив сравнимых элементов;
- на выходе — тот же массив, но отсортированный в порядке, заданным функцией сравнения.

3.2 Средства реализации

В качестве языка программирования для реализации данной лабораторной работы был выбран современный компилируемый ЯП Rust [3]. Данный выбор обусловлен моим желанием расширить свои знания в области применения данного языка, а также тем, что данный язык предоставляет широкие возможности для написания тестов [4].

3.3 Листинг кода

В листингах 3.1 – 3.3 приведены листинги алгоритма сортировки пузырьком, вставками и выбором соответственно. В листингах 3.4 – 3.5 приведены примеры реализации тестов и бенчмарков.

```rust
pub fn bubble_sort<T: Ord>(arr: &mut [T]) {
    if arr.len() == 0 {
        return;
    }

    for i in 0..(arr.len() - 1) {
        for j in 0..(arr.len() - 1 - i) {
            if arr[j] > arr[j + 1] {
                arr.swap(j, j + 1);
            }
```

```
        }
    }
}
```

Листинг 3.1: Алгоритм сортировки пузырьком

```rust
pub fn insertion_sort<T: Ord>(arr: &mut [T]) {
    for i in 1..arr.len() {
        for j in (1..(i + 1)).rev() {
            if arr[j] < arr[j - 1] {
                arr.swap(j, j - 1);
            } else {
                break;
            }
        }
    }
}
```

Листинг 3.2: Алгоритм сортировки вставками

```rust
pub fn selection_sort<T: Ord + Clone>(arr: &mut [T]) {
    if arr.len() == 0 {
        return;
    }

    for i in 0..(arr.len() - 1) {
        let mut min_elem = arr[i].clone();
        let mut min_index = i;

        for j in (i + 1)..arr.len() {
            if arr[j] < min_elem {
                min_elem = arr[j].clone();
                min_index = j;
            }
        }

        arr.swap(i, min_index);
    }
}
```

Листинг 3.3: Алгоритм сортировки выбором

```
    arr.sort();
    for i in 0..SORTS_ARRAY.len() {
        SORTS_ARRAY[i](&mut arrays[i]);
        assert_eq!(arrays[i], arr, "{}", SORTS_DESCRIPTIONS[i]);
    }
}

```

```
8
9  #[test]
10 fn check_random() {
```

Листинг 3.4: Пример реализации теста

```
1     let mut arrays: Vec<_> = (0..SORTS_ARRAY.len()).map(|_| arr.clone()).collect();
2     arr.sort();
3     for i in 0..SORTS_ARRAY.len() {
4         SORTS_ARRAY[i](&mut arrays[i]);
5         assert_eq!(arrays[i], arr, "{}", &SORTS_DESCRIPTIONS[i]);
6     }
```

Листинг 3.5: Пример реализации бенчмарка

3.4 Тестирование функций

В таблице 3.1 приведены тесты для функций, реализующих алгоритмы сортировки. Тесты пройдены успешно.

Входной массив	Результат	Ожидаемый результат
[1,2,3,4]	[1,2,3,4]	[1,2,3,4]
[3,2,1]	[1,2,3]	[1,2,3]
[5,6,2,4, − 2]	[−2,2,4,5,6]	[−2,2,4,5,6]
[4]	[4]	[4]
[]	[]	[]

Таблица 3.1: Тестирование функций

Вывод

Правильный выбор инструментов разработки позволил эффективно реализовать алгоритмы, настроить модульное тестирование и выполнить исследовательский раздел лабораторной работы.

4 Исследовательская часть

4.1 Технические характеристики

- Операционная система: Manjaro [5] Linux [6] x86_64.
- Память: 8 GiB.
- Процессор: Intel® Core™ i7-8550U[7].

Тестирование проводилось на ноутбуке, включенном в сеть электропитания. Во время тестирования ноутбук был нагружен только встроенными приложениями окружения, окружением, а также непосредственно системой тестирования.

4.2 Время выполнения алгоритмов

Результаты замеров приведены в таблицах 4.1, 4.2 и 4.3. На рисунках 4.1, 4.2 и 4.3 приведены графики зависимостей времени работы алгоритмов сортировки от размеров массивов на отсортированных, обратно отсортированных и случайных данных.

Размер	Время сортировки, мс		
	Пузырьком	**Вставками**	**Выбором**
100	7.068	0.250	4.083
200	26.988	0.502	17.110
300	59.298	0.730	23.351
400	107.697	0.958	41.347
500	161.647	1.175	60.030
1000	636.853	2.273	222.302
1500	1369.023	3.234	505.098
2000	2389.259	4.512	851.278
2500	3647.803	5.575	1307.592
5000	14895.849	11.719	5135.223
10000	58307.149	24.069	20307.903

Таблица 4.1: Время работы алгоритмов сортировки на отсортированных данных

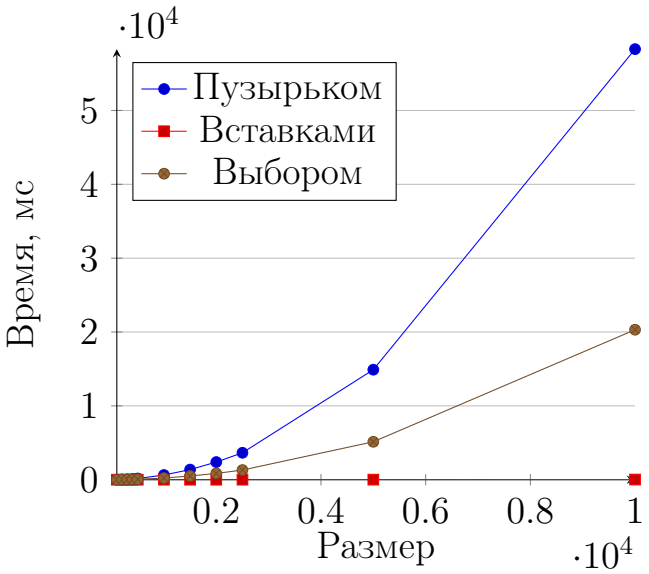

Рис. 4.1: Зависимость времени работы алгоритма сортировки от размера отсортированного массива

Размер	Время сортировки, мс		
	Пузырьком	**Вставками**	**Выбором**
100	8.086	9.157	3.252
200	32.521	35.378	14.212
300	75.489	78.161	31.237
400	135.382	137.685	52.899
500	209.979	213.662	79.600
1000	861.301	845.836	290.952
1500	1859.053	1864.651	633.313
2000	3339.893	3336.791	1063.964
2500	5237.301	5049.810	1640.561
5000	20874.374	19542.268	6438.368
10000	87888.557	80619.583	25833.789

Таблица 4.2: Время работы алгоритмов сортировки на обратно отсортированных данных

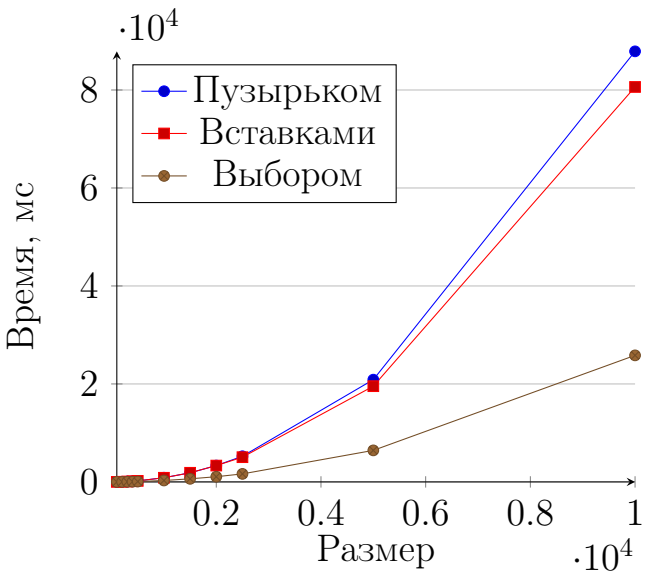

Рис. 4.2: Зависимость времени работы алгоритма сортировки от размера массива, отсортированного в обратном порядке

Размер	Время сортировки, мс		
	Пузырьком	**Вставками**	**Выбором**
100	6.866	4.961	4.083
200	56.883	18.281	17.110
300	129.535	41.556	38.305
400	230.850	70.562	72.523
500	348.290	110.935	111.524
1000	1266.627	410.407	420.069
1500	2632.387	909.836	920.052
2000	4649.903	1524.800	1562.513
2500	7338.153	2584.541	2360.383
5000	30918.510	9127.116	10334.131
10000	167716.113	36373.969	41401.814

Таблица 4.3: Время работы алгоритмов сортировки на случайных данных

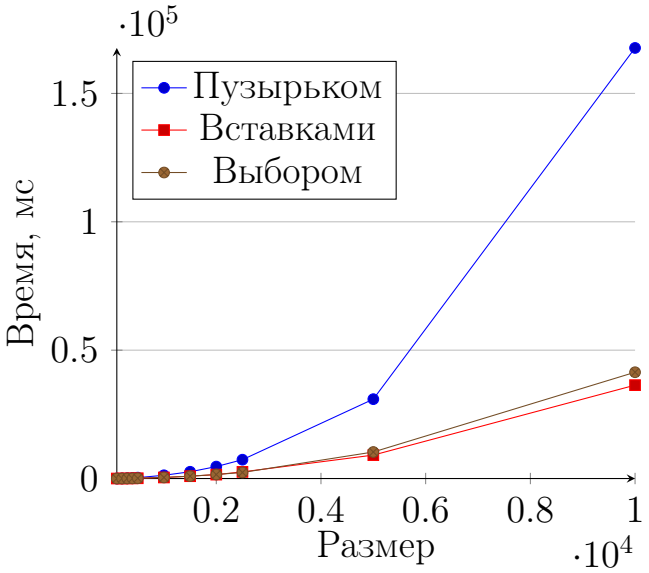

Рис. 4.3: Зависимость времени работы алгоритма сортировки от размера случайного массива

Вывод

Алгоритм сортировки вставками работает лучше остальных двух на случайных числах и уже отсортированных, практический интерес, конечно, представляет лишь первый случай, на котором сортировка вставками почти стабильно быстрее других рассматриваемых алгоритмов. Например, на массиве в 5000 элементов сортировка вставками выигрывает по времени на случайных данных пузырьковую сортировку на 66%, выбором — на 12%.

Заключение

В рамках лабораторной работы:

- были изучены и реализованы 3 алгоритма сортировки: пузырёк, вставками, выбором;
- был проведен сравнительный анализ трудоёмкости алгоритмов на основе теоретических расчетов и выбранной модели вычислений;
- был проведен сравнительный анализ алгоритмов на основе экспериментальных данных;
- был подготовлен отчет по проделанной работе.

Наиболее эффективной оказалась сортировка вставками, выигрывая по времени на случайных данных пузырьковую сортировку на 66%, выбором — на 12%. Так же было отмечено, что сортировка выбором работает всегда за примерно одинаковое время, поэтому в худшем случае (случай, когда элементы массива отсортированы в обратном порядке) обгоняет остальные два алгоритма.

Литература

[1] Кнут Дональд. Сортировка и поиск. Вильямс, 2000. Т. 3 из *Искусство программирования.* с. 834.

[2] Sort Benchmark. URL: `http://sortbenchmark.org/` (дата обращения: 25.11.2019).

[3] Rust Programming Language [Электронный ресурс]. URL: `https://doc.rust-lang.org/std/index.html`. 2017.

[4] Документация по ЯП Rust: бенчмарки [Электронный ресурс]. Режим доступа: `https://doc.rust-lang.org/1.7.0/book/benchmark-tests.html` (дата обращения: 21.09.2020).

[5] Manjaro – enjoy the simplicity [Электронный ресурс]. Режим доступа: `https://manjaro.org/` (дата обращения: 21.09.2020).

[6] Русская информация об ОС Linux [Электронный ресурс]. Режим доступа: `https://www.linux.org.ru/` (дата обращения: 21.09.2020).

[7] Процессор Intel® Core™ i7-8550U [Электронный ресурс]. Режим доступа: `https://ark.intel.com/content/www/ru/ru/ark/products/122589/intel-core-i7-8550u-processor-8m-cache-up-to-4-00-ghz.html` (дата обращения: 21.09.2020).