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»

Отчёт по лабораторной работе №3 по дисциплине "Анализ алгоритмов"

Тема Алгоритмы сортировки

Студент Романов А.В.

Группа ИУ7-53Б

Оценка (баллы) _____

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

Москва — 2020 г.

Оглавление

Введение

Одной из важнейших процедур обработки структурированной информации является сортировка. 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;
- 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 сортировки.

Задачи лабораторной работы:

- изучить и реализовать 3 алгоритма сортировки: пузырёк, вставками, быстрая сортировка;
- провести сравнительный анализ трудоёмкости алгоритмов на основе теоретических расчетов и выбранной модели вычислений;
- провести сравнительный анализ алгоритмов на основе экспериментальных данных.

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.

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 Быстрая сортировка

Общая идея алгоритма состоит в следующем:

- выбрать из массива элемент, называемый опорным. Это может быть любой из элементов массива. От выбора опорного элемента не зависит корректность алгоритма, но в отдельных случаях может сильно зависеть его эффективность
- сравнить все остальные элементы с опорным и переставить их в массиве так, чтобы разбить массив на три непрерывных отрезка, следующих друг за другом: «элементы меньшие опорного», «равные» и «большие».
- для отрезков «меньших» и «больших» значений выполнить рекурсивно ту же последовательность операций, если длина отрезка больше единицы.

На практике массив обычно делят не на три, а на две части: например, «меньшие опорного» и «равные и большие»; такой подход в общем случае эффективнее, так как упрощает алгоритм разделения.

1.4 Вывод

В данном разделе были рассмотрены 3 алгоритмы сортировки: пузырьком, вставками и быстрая сортировка.

2 Конструкторская часть

2.1 Схемы алгоритмов

На рисунках 2.1, 2.2 и 2.3 показаны схемы алгоритмов сортировки пузырьком, вставками и быстрой сортировки соответственно.

2.2 Модель вычислений

Для последующего вычисления трудоемкости введём 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

Пусть размер массивов во всех вычислениях обозначается как N.

2.3.1 Алгоритм сортировки пузырьком

Трудоёмкость алгоритма сортировки пузырьком состоит из:

- трудоёмкость сравнения и инкремента внешнего цикла $i \in [1..N)$ (2.4):

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

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

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

- трудоёмкость условия во внутреннем цикле (2.6):

$$f_{if} = 4 + \begin{cases} 0, & \text{в лучшем случае} \\ 9, & \text{в худшем случае} \end{cases} \tag{2.6}$$

Трудоёмкость в **лучшем** случае (2.7):

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

Трудоёмкость в **худшем** случае (2.8):

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

2.3.2 Алгоритм сортировки вставками

Трудоёмкость алгоритма сортировки пузырьком состоит из:

- трудоёмкость сравнения и инкремента внешнего цикла $i \in [1..N)$ (2.9):

$$f_i = 2 + 2(N-1) \tag{2.9}$$

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

$$f_{if} = 4 + \begin{cases} 0, & \text{в лучшем случае} \\ 3(N-1) + \frac{N \cdot (N-1)}{2} \cdot (3 + f_{if}), & \text{в худшем случае} \end{cases} \tag{2.10}$$

- трудоёмкость условия во внутреннем цикле (2.11):

$$f_{if} = 4 + \begin{cases} 0, & \text{в лучшем случае} \\ 9, & \text{в худшем случае} \end{cases} \tag{2.11}$$

Трудоёмкость в **лучшем** случае (2.12):

$$f_{best} = 13N - 10 \approx 13N = O(N) \tag{2.12}$$

Трудоёмкость в **худшем** случае (2.13):

$$f_{worst} = 4.5N^2 + 10N - 13 \approx 4N^2 = O(N^2) \tag{2.13}$$

2.4 Вывод

На основе теоретических данных, полученных из аналитического раздела, были построены схемы трёх алгоритмов сортировки. Оценены их тредёмкости в лучшем и худшем случаях.

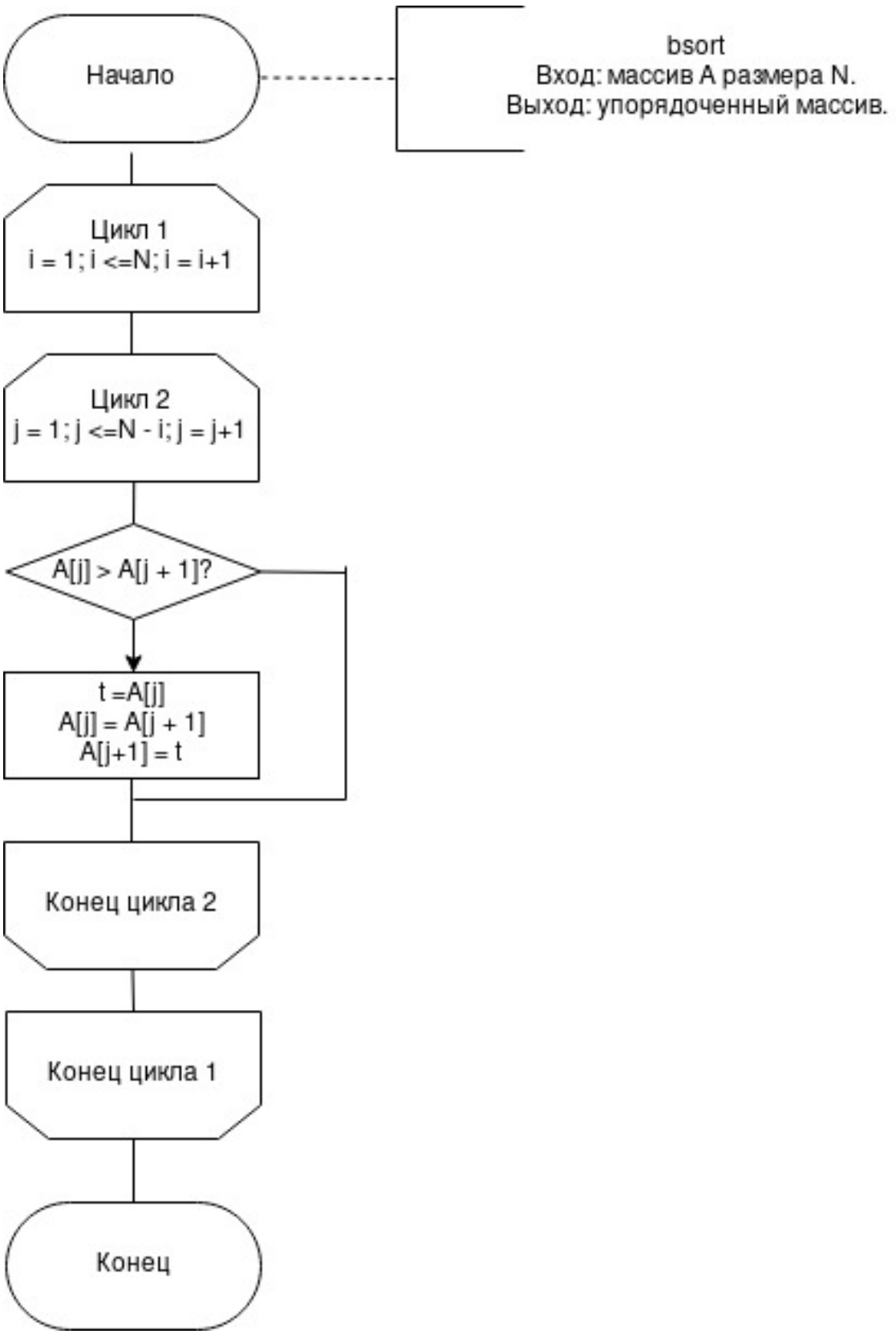

Рис. 2.1: Схема сортировки пузырьком

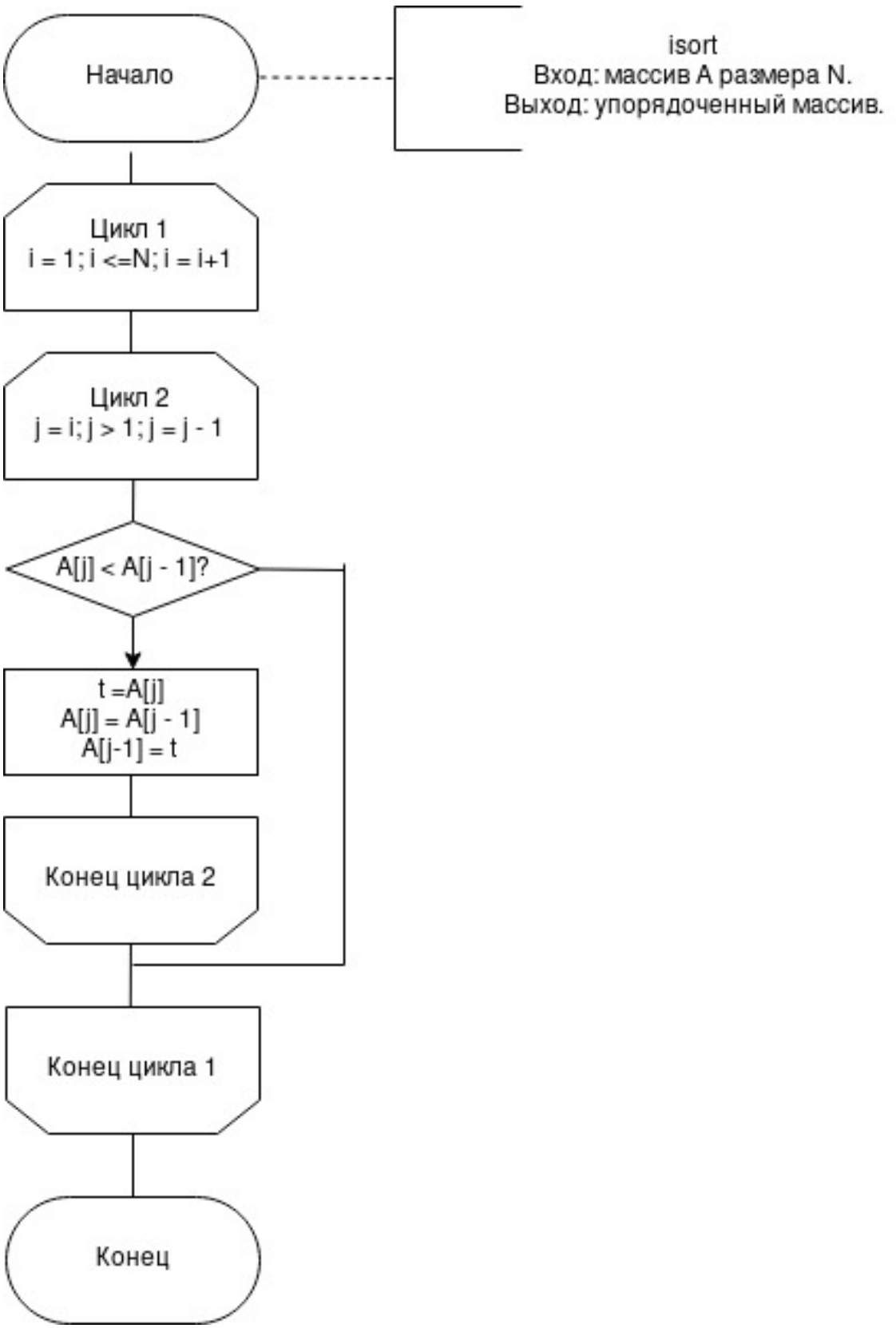

Рис. 2.2: Схема сортировки вставками

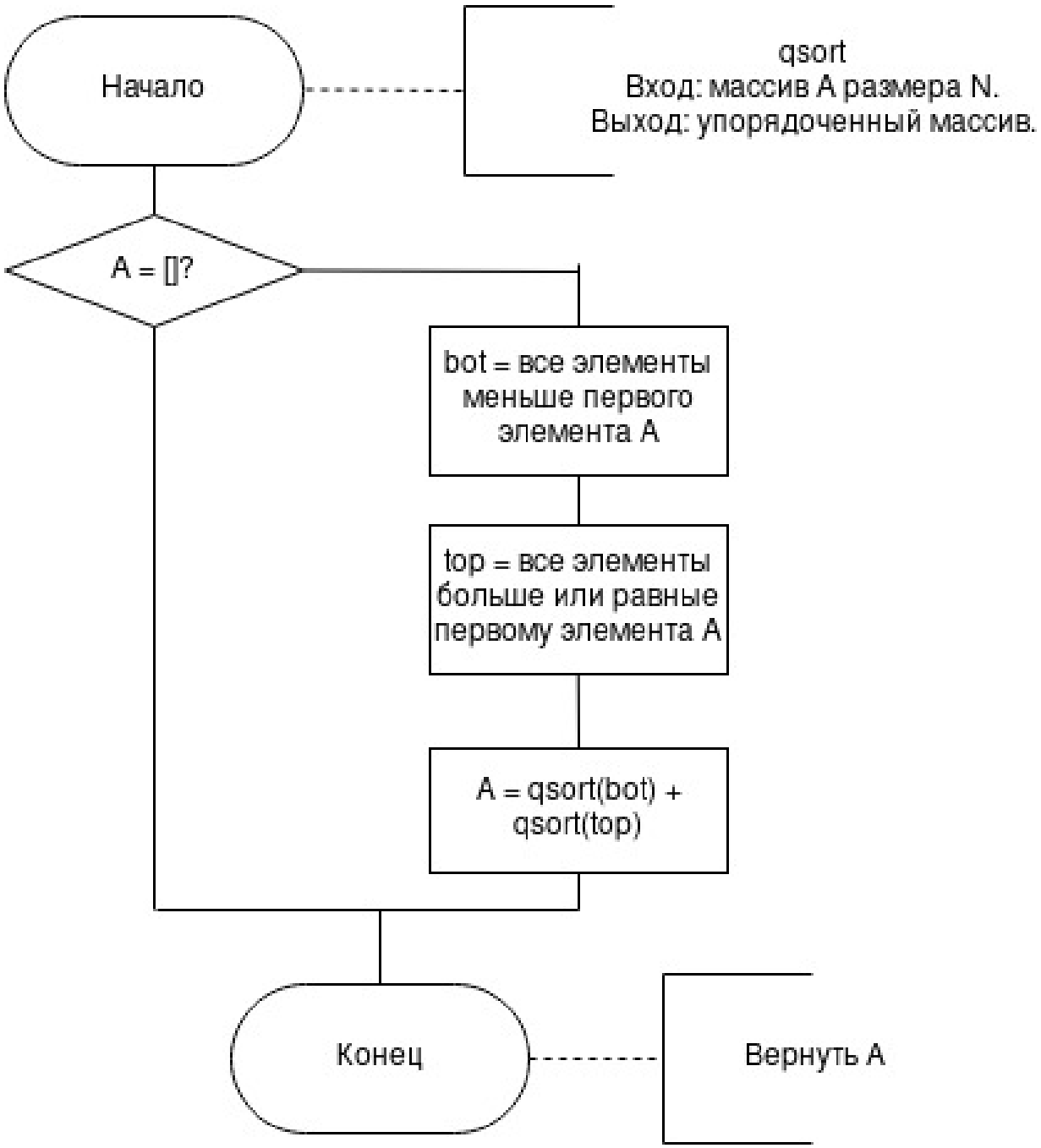

Рис. 2.3: Схема быстрой сортировки

3 Технологическая часть

В данном разделе приведены средства реализации и листинги кода.

3.1 Требование к ПО

К программе предъявляется ряд требований:

- на вход ПО получает массив сравнимых элементов;
- на выходе – тот же массив, но отсортированный в заданном порядке.

3.2 Средства реализации

Для реализации ПО я выбрал язык программирования OCaml [1]. Данный выбор обусловлен моим желанием расширить свои знания в области применения данного язкыа программирования.

3.3 Реализация алгоритмов

В листингах 3.1 - 3.3 приведена реализация трёх алгоритмов сортировки.

Листинг 3.1: Функция сортировки массива пузырьком

```
let rec bsort arr =
  let rec _bsort = function
    | x1 :: x2 :: xs when x1 > x2 -> x2 :: _bsort (x1 :: xs)
    | x1 :: x2 :: xs -> x1 :: _bsort (x2 :: xs)
    | arr -> arr
  in
  let maybe_sorted = _bsort arr in
    if maybe_sorted = arr then arr
  else bsort maybe_sorted
;;
```

Листинг 3.2: Функция сортировки массива вставками

```
let rec isort arr =
  let rec insert k = function
    | x :: xs when k < x -> k :: x :: xs
```

```
    | x :: xs -> x :: (insert k xs)
    | arr -> [k]
  in
    match arr with
    | [] -> []
    | x :: xs -> insert x (isort xs)
;;
```

Листинг 3.3: Функция быстрой сортировки

```
let rec qsort = function
    | [] -> []
    | x :: xs -> let bot, top = List.partition (fun a -> a < x) xs
                 in qsort bot @ (x :: qsort top)
;;
```

3.4 Тестовые данные

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

Входной массив	Результат	Ожидаемый результат
$[15, 25, 35, 45, 55]$	$[15, 25, 35, 45, 55]$	$[15, 25, 35, 45, 55]$
$[55, 45, 35, 25, 15]$	$[15, 25, 35, 45, 55]$	$[15, 25, 35, 45, 55]$
$[-10, -20, -30, -25, -50]$	$[-50, -30, -25, -20, -10]$	$[-50, -30, -25, -20, -10]$
$[40, -10, 20, -30, 75]$	$[-30, -10, 20, 40, 75]$	$[-30, -10, 20, 40, 75]$
$[100]$	$[100]$	$[100]$
$[-20]$	$[-20]$	$[-20]$
Пустой массив	Пустой массив	Пустой массив

Таблица 3.1: Тестирование функций

3.5 Вывод

В данном разделе были разработаны исходные коды трёх алгоритмов сортировки: пузырьком, вставками и быстрая сортировка.

4 Исследовательская часть

4.1 Технические характеристики

Ниже приведены технические характеристики устройства, на котором было проведено тестирование ПО:

- Операционная система: Debian [2] Linux [3] 11 «bullseye» 64-bit.
- Оперативная память: 12 GB.
- Процессор: Intel(R) Core(TM) i5-3550 CPU @ 3.30GHz [4].

4.2 Время выполнения алгоритмов

Время выполнения алгоритм замерялось с помощью применения технологии профайлинга [5]. Данный инстрмуент даёт детальное описание количества вызовов и количества времени CPU, занятого каждой функцией.

Таблица 4.1: Таблица времени выполнения сортировок на отсортированных данных (в секундах)

Размер	bsort	isort	qsort
1000	0.01922	2e-05	0.00025
1200	0.02419	7e-05	0.00029
1400	0.03516	6e-05	0.00035
1600	0.04540	0.0001	0.00034
1800	0.06051	6e-05	0.00032
2000	0.06591	9e-05	0.00038

4.3 Вывод

Как и ожидалось в результате оценки трудоемкости алгоритмов, сортировка вставками работает очень быстро на уже отсортированном массиве. Время сортировки вставками на всех трёх видах массивов примерно одинаково. При обратно отсортированном массиве, быстрая

Таблица 4.2: Таблица времени выполнения сортировок на отсортированных данных в обратном порядке (в секундах)

Размер	bsort	isort	qsort
1000	0.02464	0.02351	0.05463
1200	0.03504	0.03444	0.08216
1400	0.04539	0.04633	0.11752
1600	0.07741	0.06945	0.18477
1800	0.08010	0.07510	0.19965
2000	0.08919	0.08319	0.22478

Таблица 4.3: Таблица времени выполнения сортировок на случайных данных (в секундах)

Размер	bsort	isort	qsort
1000	0.01937	0.01129	0.00065
1200	0.02371	0.01426	0.00042
1400	0.03296	0.01619	0.00095
1600	0.04761	0.01842	0.00143
1800	0.06122	0.02028	0.00145
2000	0.06756	0.02129	0.00220

сортировка начинает значительно проигрывать по скорости. При прямом порядке элементов в массиве, сортировка пузьком работает в 115 раз медленее чем сортировка вставками, и в 70 раз медленее чем быстрая.

Заключение

В рамках данной лабораторной работы:

- были изучены и реализованы 3 алгоритма сортировки: пузырёк, вставками, быстрая сортировка;
- был проведён сравнительный анализ трудоёмкости алгоритмов на основе теоретических расчетов и выбранной модели вычислений;
- был проведён сравнительный анализ алгоритмов на основе экспериментальных данных.

На основании анализа трудоемкости алгоритмов в выбранной модели вычислений, было показано, что алгоритм сортировки вставками имеет наименьшую сложность (линейную) в уже отсортированном массиве. В случае обранто отсортированного массива, сортировка вставками и пузырьком имееют квадратическую сложность. На основании замеров времени исполнения алгоритмов, был сделан вывод, что при прямом порядке элементов в массиве, сортировка пузьком работает в 115 раз медленее чем сортировка вставками, и в 70 раз медленее чем быстрая. Так же был доказана выведенная трудоемкость алгоритма сортировки вставками – при уже отсортированном массиве сортировка работает очень быстро. На случайных данных быстрее всех работает алгоритм быстрой сортировки.

Литература

[1] OCaml – industrial strength programming language supporting functional, imperative and object-oriented styles [Электронный ресурс]. Режим доступа: https://ocaml.org/. Дата обращения: 01.10.2020.

[2] Debian – универсальная операционная система [Электронный ресурс]. Режим доступа: https://www.debian.org/. Дата обращения: 20.09.2020.

[3] Linux – Getting Started [Электронный ресурс]. Режим доступа: https://linux.org. Дата обращения: 20.09.2020.

[4] Процессор Intel® Core™ i5-3550 [Электронный ресурс]. Режим доступа: https://ark.intel.com/content/www/ru/ru/ark/products/65516/intel-core-i5-3550-processor-6m-cache-up-to-3-70-ghz.html. Дата обращения: 20.09.2020.

[5] OCaml profiling [Электронный ресурс]. Режим доступа: https://caml.inria.fr/pub/docs/manual-ocaml/profil.html. Дата обращения: 01.10.2020.