Государственное образовательное учреждение высшего профессионального образования
“Московский государственный технический университет имени Н.Э.Баумана”

Дисциплина: Анализ алгоритмов

Лабораторная работа 3

# Алгоритмы сортировки массива.

Студент группы ИУ7-55,
Аминов Тимур Саидович

2019 г.

# Содержание

# Введение

Цель лабораторной работы: изучение алгоритмов сортировки массивов, сравнительный анализ времени работы данных алгоритмов, анализ трудоемкости алгоритмов.
Задачи работы:
1) реализация следующих алгоритмов сортировки массивов – сортировка пузырьком, пузырьком с барьером и шейкер-сортировка;
2) оценка трудоемкости алгоритмов;
3) анализ времени работы программы;
4) сравнительный анализ работы алгоритмов для массивов размера от 100 до 1500 элементов.

# 1 Аналитическая часть

В данной части будут рассмотрены теоретические основы алгоритмов и приведена модель вычислений для оценок трудоемкости.

## 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 или до тех пор, пока на очередном проходе не окажется, что обмены больше не нужны, что означает — массив отсортирован. При каждом проходе алгоритма по внутреннему циклу, очередной наибольший элемент массива ставится на своё место в конце массива рядом с предыдущим «наибольшим элементом», а наименьший элемент перемещается на одну позицию к началу массива («всплывает» до нужной позиции, как пузырёк в воде — отсюда и название алгоритма).
Сложность алгоритма - $O(n^2)$[1].

## 1.2 Сортировка пузырьком с барьером

Модификация алгоритма пузырьком, одной из проблем которого является проход по уже отсортированному массиву. Второй проблемой пузырьковой сортировки является то, что она будет проходит по отсортированной области. Обе эти проблемы решаются введением барьера - местом последнего обмена. Проход по массиву осуществляется не до последнего элемента, а до барьерного, то есть мы проходим только по неотсортированной части массива.
Сложность алгоритма - $O(n^2)$[1].

## 1.3 Шейкер-сортировка

Еще одной проблемой пузырька является то, что при движении от начала массива к концу максимальный элемент становится на последнюю позицию, а минимальный элемент сдвигается только на одну позицию влево. Эта проблема решается поочередными проходами от начала к концу и от конца к началу, постоянно сокращая "рабочую область"массива.
Сложность алгоритма будет вычислена в разделе 3.4.

## 1.4 Модель вычислений

В рамках данной работы используется следующая модель вычислений:

1. Базовые операции имеют трудоемкость 1 (<, >, =, <=, =>, ==, +, -, *, /, %, &, +=, -=, *=, /=, [ ]);
2. оператор if, else if имеет трудоемкость $F_{if} = F_{body} + F_{chek}$, $F_{body}$ - трудоемкость операций тела оператора, $F_{chek}$ - трудоемкость проверки условия;
3. оператор else имеет трудоемкость $F_{body}$;
4. оператор for имеет трудоемкость $F_{for} = 2 + N \cdot (F_{body} + F_{chek})$, где $F_{body}$ – трудоемкость операций в теле цикла.

## 1.5 Вывод

В данном разделе была поверхностно рассмотрена сортировка пузырьком, а также три модификации, ее оптимизирующие: введение флага, введение барьера для сортируемой области массива и чередование проходов по массиву - от начала к концу и от конца к началу.

# 2 Конструкторская часть

В данном разделе будут рассмотрены схемы алгоритмов, рассматриваемых в ходе лабораторной работы.

## 2.1 Схемы алгоритмов

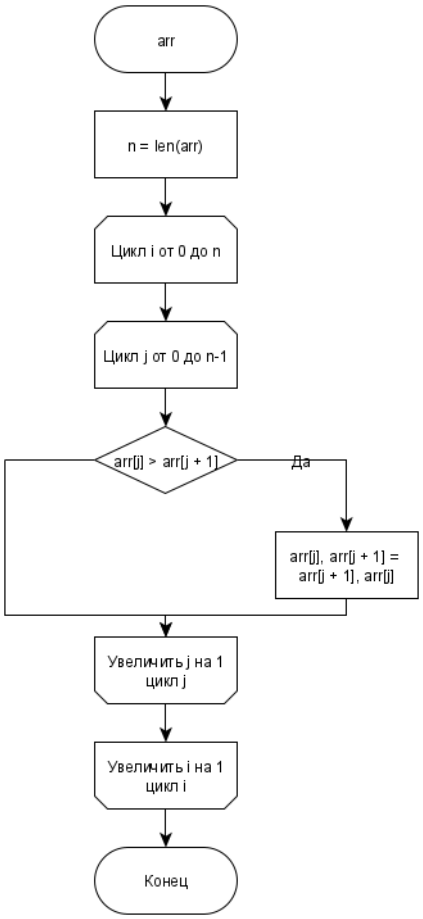


Рис. 1: Схема алгоритма сортировки пузырьком

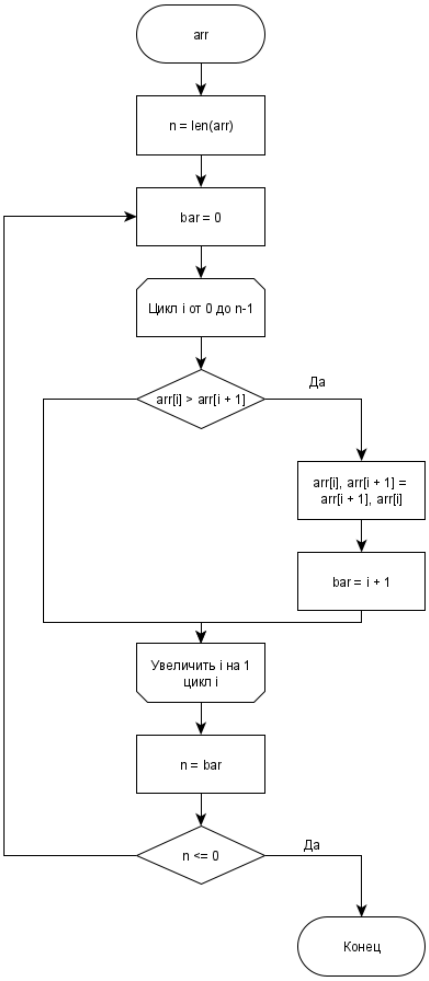


Рис. 2: Схема алгоритма сортировки пузырьком с барьером

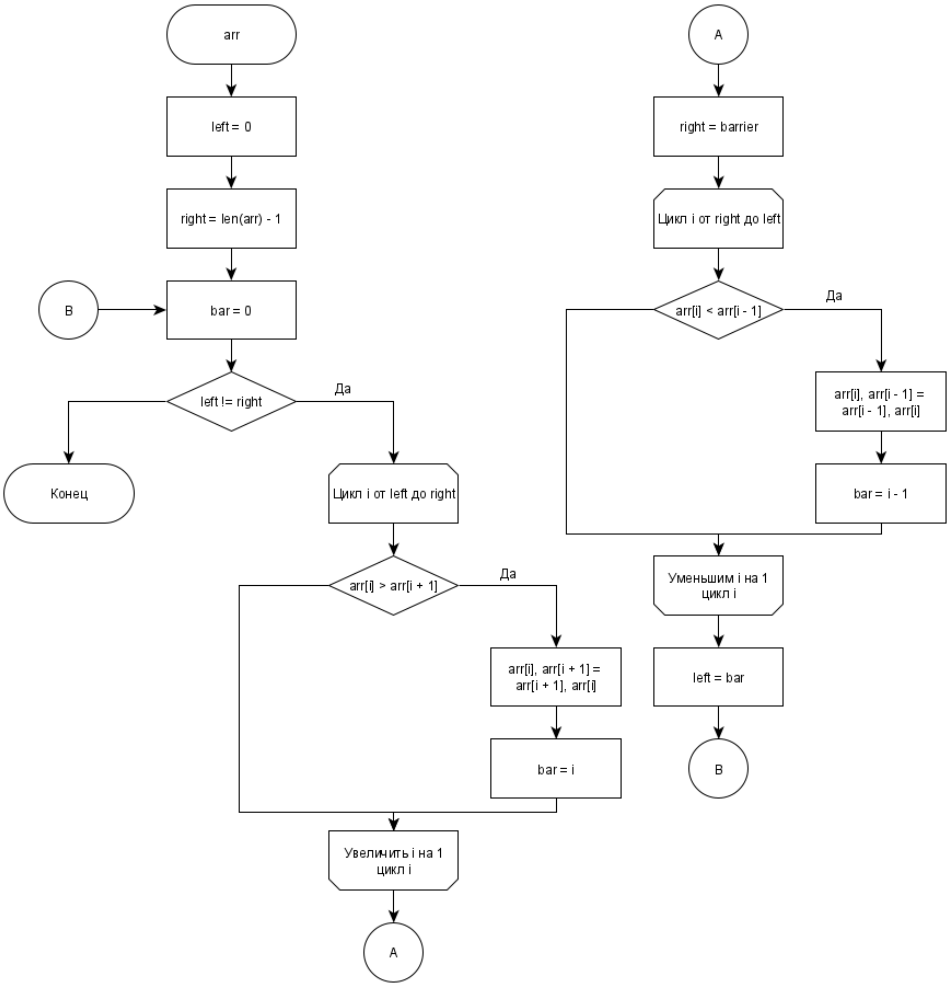


Рис. 3: Схема алгоритма сортировки пузырьком с барьером

## 2.2 Оценки трудоемкости

**Трудоемкость обычного алгоритма пузырьком.**
В лучшем случае - $O(n^2)$(отсортированный массив)[1]
В худшем случае - $O(n^2)$(отсортированный в обратную сторону массив)[1]
**Трудоемкость алгоритма пузырьком с барьером.**
В лучшем случае - $O(n)$(отсортированный массив)[1]
В худшем случае - $O(n^2)$(отсортированный в обратную сторону массив)[1]
**Трудоемкость шейкер-сортировки.**
В лучшем случае:
Посчитаем количество итераций. Для внешнего цикла *dowhile* это 1, для внутреннего $for_1$ - это n - 1, причем, так как массив отсортированный, условие if(arr[i] $>$ arr[i + 1]) не выполнится ни разу, из-за чего после выхода из $for_1$ left $==$ barrier $==$ right. Поэтому в тело цикла $for_2$ мы не зайдем ни разу, лишь один раз отработав условие. Итоговые вычисления представлены в таблице 1:

Таблица 1. Трудоемкость шейкер-сортировки в лучшем случае.

| Трудоемкость | Оценка |
|---|---|
| $F$ | $4 + F_{dowhile}$ |
| $F_{dowhile}$ | $5 + F_{for1} + F_{for2}$ |
| $F_{for1}$ | $1 + (n-1)(2+4) = 6n - 5$ |
| $F_{for2}$ | 2 |
| $F_{best}$ | $6n + 6 \approx O(n)$ |

В худшем случае:
Посчитаем количество итераций. Для внешнего цикла *dowhile* это n/2(так как за одну итерацию pleft и pright будут инкрементированы, то есть "сближены"на 2, начальное "расстояние"между ними n - 1). Для

внутреннего $for_1$ суммарное количество итераций - это сумма последовательности n - 1, n - 3, ... , 1. Эта сумма равна $S_1 = \frac{n-1+1}{2} \cdot \frac{n}{2} = \frac{n^2}{4}$ , причем, так как массив отсортированный в обратном порядке, условие if(arr[i] $>$ arr[i + 1]) выполнится при каждой итерации $for_1$. Для внутреннего $for_2$ суммарное количество итераций - это сумма последовательности n - 2, n - 4, ... , 0 . Эта сумма равна $S_1 = \frac{n-2}{2} \cdot \frac{n}{2} = \frac{n^2-2n}{4}$, в этом цикле также условие if(arr[i] $<$ arr[i - 1]) будет всегда верно. Итоговые вычисления представлены в таблице 2:

Таблица 2. Трудоемкость шейкер-сортировки в лучшем случае.

| Трудоемкость | Оценка |
|---|---|
| $F$ | $4 + F_{dowhile}$ |
| $F_{dowhile}$ | $2.5n + F_{for1} + F_{for2}$ |
| $F_{for1}$ | $\frac{n}{2} + 16\frac{n^2}{4} = 0.5n + 4n^2$ |
| $F_{for2}$ | $\frac{n}{2} + 17\frac{n^2-2n}{4} = -8n + 4.25n^2$ |
| $F_{worst}$ | $4 - 5n + 8.25n^2 \approx O(n^2)$ |

Итого по таблицам 1 и 2 мы можем сделать вывод, что трудоемкость алгоритма шейкер-сортировки равна трудоемкости сортировки пузырьком с барьером($O(n)$ в лучшем случае и $O(n^2)$ - в худшем).

## 2.3 Вывод

В данном разделе были рассмотрены схемы алгоритмов сортировки пузырьком, пузырьком с барьером и шейкер-сортировки, приведенные на рисунках 1-3.

# 3 Технологическая часть

В данной части будет рассмотрена трудоемкость каждого из рассматриваемых в данной лабораторной работе алгоритма, а также будут сформулированы требования к программному обеспечению.

## 3.1 Требования к программному обеспечению

Входные данные - массив, его длина. Выходные данные - отсортированный массив.


Рис. 4: IDEF0-диаграмма, описывающая сортировку массива

## 3.2 Средства реализации

Программа была написана на языке python версии 3.8. Программа корректно работает с пустыми и неправильно введенными данными. Для замеров времени была использована библиотека timeit.

## 3.3 Листинг кода

В листингах 1-3 представлена реализация заявленных алгоритмов.

Листинг 1: Алгоритм сортировки пузырьком

```
    def bubble_sort(arr):
        n = len(arr)

        for i in range(n):
            for j in range(n - 1):
                if arr[j] > arr[j + 1]:
                    arr[j], arr[j + 1] = arr[j + 1], arr[j]

        return arr
```

Листинг 2: Алгоритм сортировки пузырьком с барьером

```
    def bubble_barrier_sort(arr):
        n = len(arr)

        while True:
            bar = 0
            for i in range(n - 1):
                if arr[i] > arr[i + 1]:
                    arr[i], arr[i + 1] = arr[i + 1], arr[i]
                    bar = i + 1
            n = bar
            if n <= 0:
                break

        return arr
```

Листинг 3: Алгоритм шейкер-сортировки

```
    def shaker_sort(arr):
        left = 0
```

```
        right = len(arr) - 1
        bar = 0

        while left != right:
            for i in range(left, right, 1):
                if arr[i] > arr[i + 1]:
                    arr[i], arr[i + 1] = arr[i + 1], arr[i]
                    bar = i
            right = bar

            for i in range(right, left, -1):
                if arr[i] < arr[i - 1]:
                    arr[i], arr[i - 1] = arr[i - 1], arr[i]
                    bar = i - 1
            left = bar

        return arr
```

## 3.4 Вывод

В данном разделе были даны реализации сортировки пузырьком, пузырьком с барьером и шейкер-сортировки, а также был проведен анализ трудоемкости шейкер-сортировки.

# 4 Экспериментальная часть

В данной части будут даны примеры работы всех сортировок и рассмотрен эксперимент, изучающий затрачиваемое ими время. Все расчеты проводились на 64-битной Windows 10, на x64 процессоре Intel Core i7 с 16гб оперативной памяти.

## 4.1 Примеры работы программы

Листинг 4: Пример работы 1

```
15 8 9 13 4 -5 17 22 3
bubble_sort: -5 3 4 8 9 13 15 17 22

bubble_barrier_sort: -5 3 4 8 9 13 15 17 22

shaker_sort: -5 3 4 8 9 13 15 17 22
```

В данном примере все алгоритмы дали верный результат.

Листинг 5: Пример работы 2

```
-1 1 2 3 4
bubble_sort: -1 1 2 3 4

bubble_barrier_sort: -1 1 2 3 4

shaker_sort: -1 1 2 3 4
```

В данном примере все алгоритмы дали верный результат.

Листинг 6: Пример работы 3

```
10 9 8 7 6 5 4 3 2 1
bubblesort: 1 2 3 4 5 6 7 8 9 10

bubblebar: 1 2 3 4 5 6 7 8 9 10

shakersort: 1 2 3 4 5 6 7 8 9 10
```

Во всех трех случаях все сортировки дали верный результат, что подтвердило их правильность.

## 4.2 Постановка эксперимента

В данном эксперименте будет проверено время на сортировку массивов длиной от 100 до 1500 с шагом 100. Для каждого значения длины будет сгенерировано 3 массива: с произвольными значениями, отсортированный и отсортированный в обратном порядке. Замеры будут проводиться по 10 раз для сокращения влияния случайных факторов на итоговый результат.

## 4.3 Результаты эксперимента

На рисунках 5-10 представлены сравнение времени сортировок на массивах различной длины, на осях абсцисс отложена длина массива, на осях ординат - затрачиваемое время(в секундах).

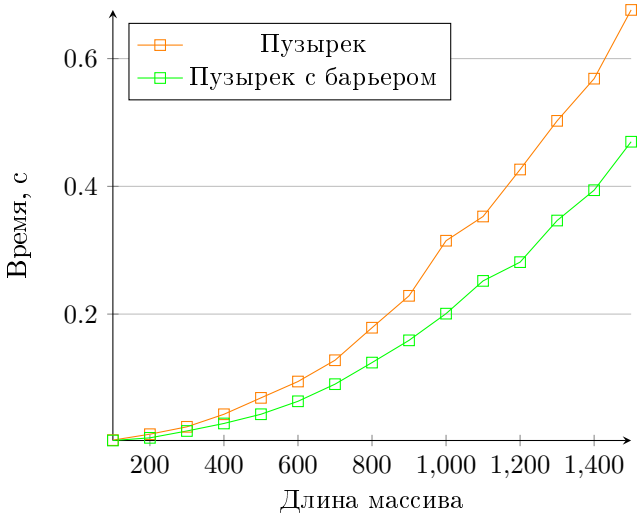


Рис. 5: сравнение времени на сортировку произвольно заполненного массива пузырьком и пузырьком с барьером.

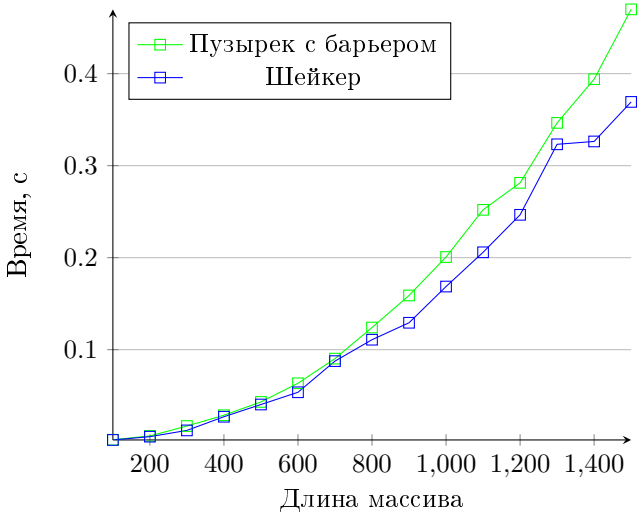


Рис. 6: сравнение времени на сортировку произвольно заполненного массива пузырьком с барьером и шейкер-сортировкой.

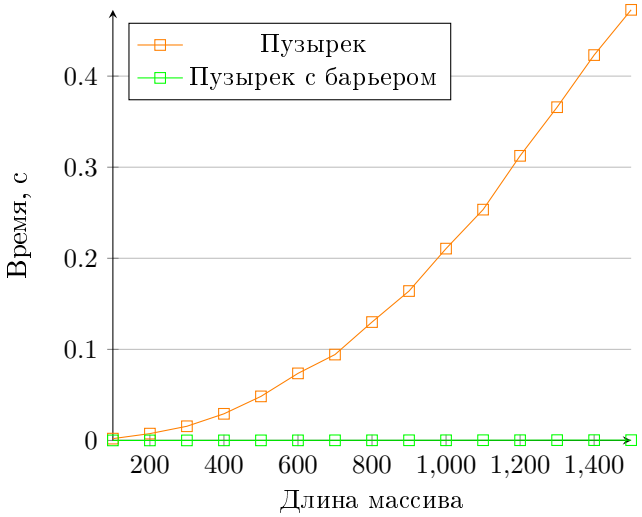


Рис. 7: сравнение времени на сортировку отсортированного массива пузырьком и пузырьком с барьером.

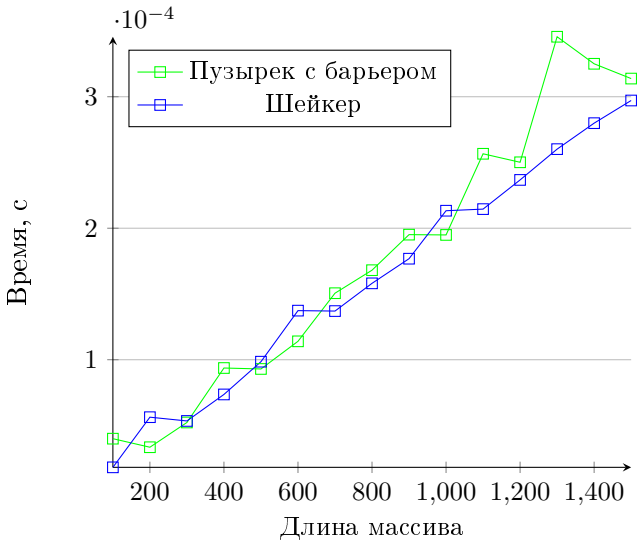


Рис. 8: сравнение времени на сортировку отсортированного массива пузырьком с барьером и шейкер-сортировкой.

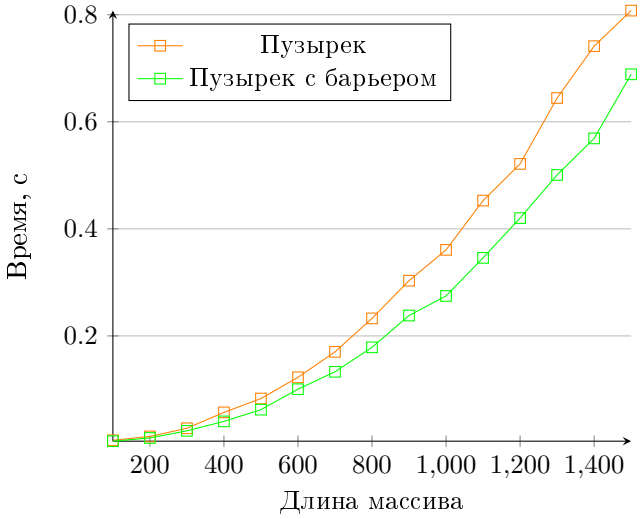


Рис. 9: сравнение времени на сортировку отсортированного в обратном порядке массива пузырьком и пузырьком с барьером.

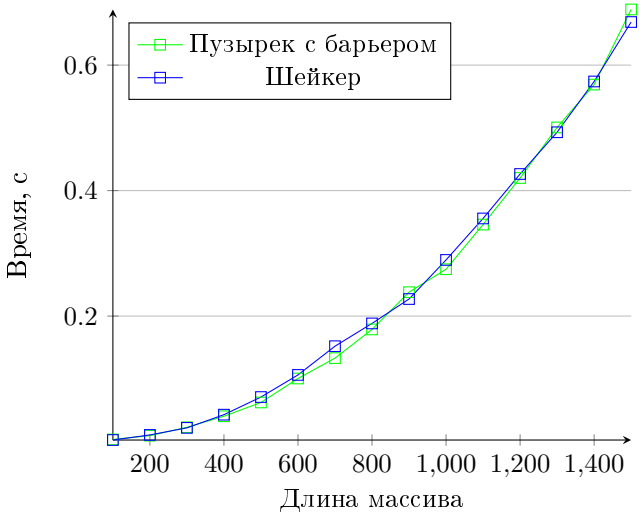


Рис. 10: сравнение времени на сортировку отсортированного в обратном порядке массива пузырьком с барьером и шейкер-сортировкой.

## 4.4 Анализ экспериментальных данных

На рисунках 5, 7 и 9 мы видим, что сортировка пузырьком значительно оптимизируется введением барьера при отсортированном массиве. Однако при неотсортированном массиве и массиве, отсортированном в обрат-

ном порядке прирост скорости прирост скорости не настолько велик.
На рисунке 6 мы видим, что разница между шейкер-сортировкой и пузырьком с барьером слабо заметна в большинстве случаев - шейкер более затратный, похожая картина наблюдается на рисунке 6 - разница между алгоритмами минимальна, в пользу пузырька с барьером. При отсортированном в обратном порядке массиве время работы массивов идентично. В целом такие данные обусловлены тем, что в шейкер-сортировке больше побочных вычислений, и ее оптимизации по сравнению с пузырьком с барьером заметны только в худшем случае.

## 4.5 Вывод

В данном разделе сортировки были протестированы на правильность, также был проведен эксперимент по замеру времени работы каждой из них.

# Заключение

В данной лабораторной работе были рассмотрены различные модификации алгоритма сортировки пузырьком - классический алгоритм, с барьером и шейкер-сортировка. Была выбрана модель вычислений и с помощью нее вычислена трудоемкость каждого из алгоритмов, после чего был поставлен эксперимент, подтвердивший правильность посчитаной трудоемкости. Помимо этого было выявлено, что добавление барьера в сортировку пузырьком на порядки увеличивает скорость работы, в то время как введение двухстороннего прохода по отношению к пузырьку с барьером может сыграть роль только в худшем случае - при обратно отсортированном массиве.

# Список литературы


[1] Кнут Д. Э. 5.2.2 Обменная сортировка // Искусство программирования. Том 3. Сортировка и поиск = The Art of Computer Programming. Volume 3. Sorting and Searching / под ред. В. Т. Тертышного (гл. 5) и И. В. Красикова (гл. 6). — 2-е изд. — Москва: Вильямс, 2007. — Т. 3. — 832 с. — ISBN 5-8459-0082-1.

[2] Макконнелл Дж. Основы современных алгоритмов = Analysis of Algorithms: An Active Learning Approach / Под ред. С. К. Ландо. — М.: Техносфера, 2004. — С. 72-76. — ISBN 5-94836-005-9.

[3] Левитин А. В. Глава 3. Метод грубой силы: Пузырьковая сортировка // Алгоритмы. Введение в разработку и анализ — М.: Вильямс, 2006. — С. 144–146. — 576 с. — ISBN 978-5-8459-0987-9

[4] Язык программирования python // [Электронный ресурс]. Режим доступа: https://www.python.org/ (дата обращения: 10.11.19).

[5] Библиотека для замера времени timeit // [Электронный ресурс]. Режим доступа: https://docs.python.org/2/library/timeit.html (дата обращения: 10.11.19).

[6] Анимация алгоритмов сортировки // [Электронный ресурс]. Режим доступа: http://sorting.at/ (дата обращения: 10.11.19).