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 Зайцева А.А.

Группа ИУ7-52Б

Преподаватели Волкова Л.Л.

Москва — 2021 г.

Оглавление

Введение

Целью данной лабораторной работы является получение навыков оценки трудоемкости алгоритмов на примере трех алгоритмов сортировок.

Под сортировкой понимается упорядочивание элементов по какому-либо признаку. Сортировка относится к важнейшему классу алгоритмов обработки данных и осуществляется большим количеством способов [1].

Алгоритмы сортировки имеют большое практическое применение и часто встречаю там, где речь идет об обработке и хранении больших объемов информации. Некоторые задачи обработки данных решаются проще, если данные заранее упорядочить. Упорядоченные объекты содержатся в телефонных книгах, ведомостях налогов, в библиотеках, в оглавлениях, в словарях [2].

В настоящее время, в связи с постоянно растущими объемами данных, вопрос эффективности сортировки данных не теряет свою актуальность.

В рамках выполнения работы необходимо решить следующие задачи:

1) изучить три алгоритма сортировок: сортировку вставками, сортировку пузырьком и шейкерную сортировку;

2) разработать и реализовать изученные алгоритмы;

3) провести сравнительный анализ трудоёмкости реализаций алгоритмов на основе теоретических расчетов;

4) провести сравнительный анализ процессорного времени выполнения реализаций алгоритмов на основе экспериментальных данных.

1 Аналитическая часть

В данном разделе будут рассмотрены основные идеи трех алгоритмов сортировок - сортировки вставками, сортировки пузырьком и шейкерной сортировки.

1.1 Сортировка вставками

Сортировка вставками — достаточно простой алгоритм. Сортируемый массив можно разделить на две части — отсортированную и неотсортированную. В начале сортировки первый элемент массива считается отсортированным, все остальные — не отсортированные. Начиная со второго элемента массива и заканчивая последним, алгоритм вставляет очередной элемент из неотсортированной части массива в нужную позицию в отсортированной части.

Таким образом, за один шаг сортировки отсортированная часть массива увеличивается на один элемент, а неотсортированная часть массива уменьшается на один элемент. Данный процесс вставки продолжается до тех пор, пока все элементы исходного списка не окажутся в расширяющейся отсортированной части списка [3].

1.2 Сортировка пузырьком

Сортировка пузырьком — один из самых известных алгоритмов сортировки. 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. 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) [1].

1.3 Шейкерная сортировка

Алгоритм шейкерной сортировки является модификацией пузырьковой сортировки по направлению движения. Отличия от нее заключаются в том, что при прохождении части массива, происходит проверка, были ли перестановки. Если их не было, значит эта часть массива уже упорядочена, и она исключается из дальнейшей обработки. Кроме того, при прохождении массива от начала к концу минимальные элементы перемещаются в самое начало, а максимальный элемент сдвигается к концу массива [4].

Вывод

В данном разделе были рассмотрены идеи, лежащие в основе рассматриваемых алгоритмов сортировок - сортировки вставками, сортировки пузырьком и шейкерной сортировки.

2 Конструкторская часть

В данном разделе будут рассмотрены схемы описанных в предыдущем разделе алгоритмов сортировок.

2.1 Схема алгоритма сортировки вставками

На рисунке 2.1 приведена схема алгоритма сортировки вставками.

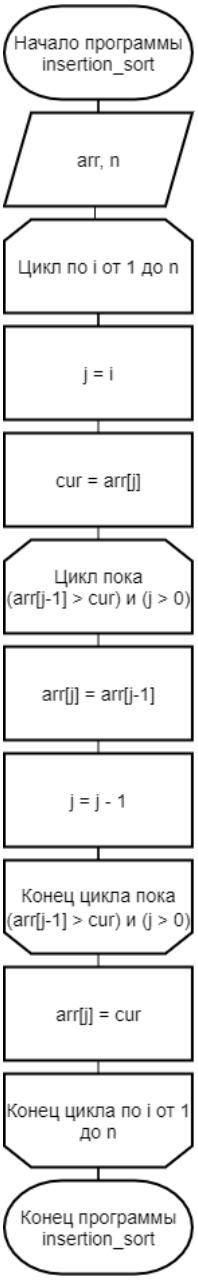

Рисунок 2.1 – Схема алгоритма сортировки вставками

2.2 Схема алгоритма сортировки пузырьком

На рисунке 2.2 приведена схема алгоритма сортировки пузырьком.

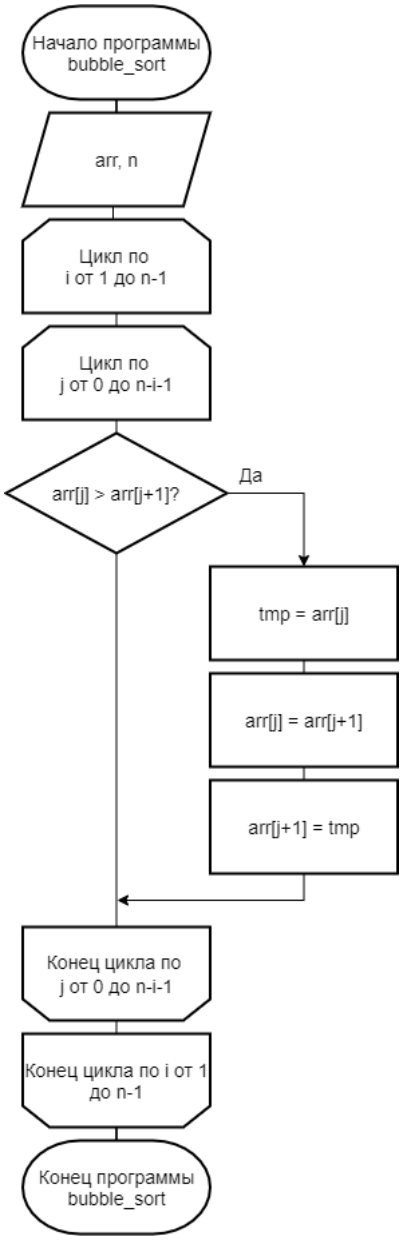

Рисунок 2.2 – Схема алгоритма сортировки пузырьком

2.3 Схема алгоритма шейкерной сортировки

На рисунке 2.3 приведена схема алгоритма шейкерной сортировки.

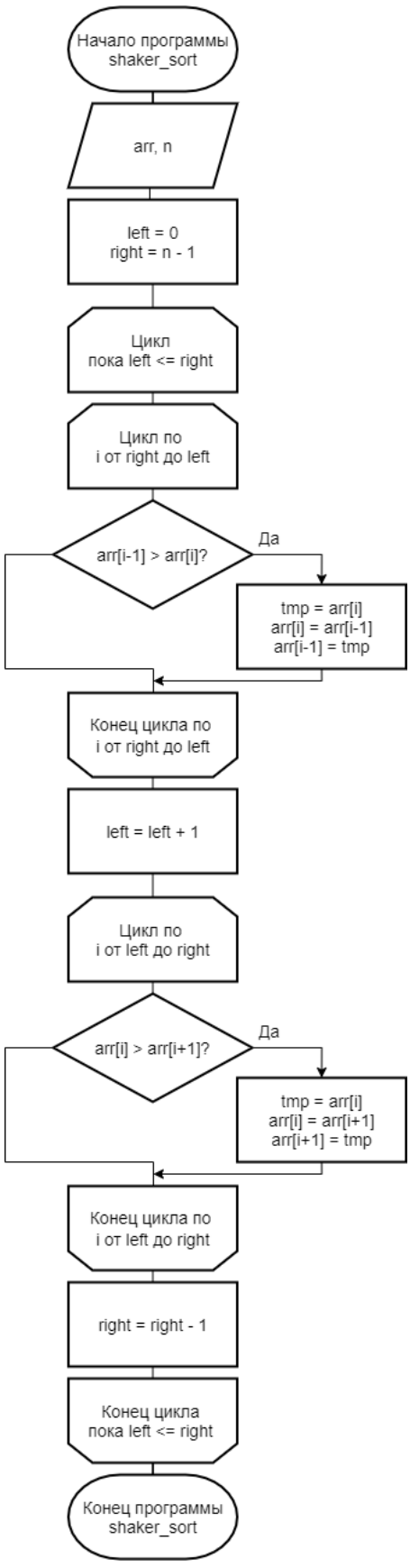

Рисунок 2.3 – Схема алгоритма шейкерной сортировки

Вывод

Были разработаны схемы трех алгоритмов сортировки.

3 Технологическая часть

В данном разделе производится выбор средств реализации, а также приводятся требования к программному обеспечению (ПО), листинги реализованных алгоритмов и тесты для программы.

3.1 Требования к ПО

На вход программе подается массив целых чисел, а на выходе должен быть получен масссив, отсортированный с помощью каждого реализованного алгоритма сортировки: вставками, пузырьком и шейкерной. Также необходимо вывести затраченное каждым алгоритмом процессорное время.

3.2 Выбор средств реализации

В качестве языка программирования для реализации данной лабораторной работы был выбран язык Python [5]. Он позволяет быстро реализовывать различные алгоритмы без выделения большого времени на проектирование сруктуры программы и выбор типов данных.

Кроме того, в Python есть библиотека time, которая предоставляет функцию process_time для замера процессорного времени [6].

В качестве среды разработки выбран PyCharm. Он является кроссплатформенным, а также предоставляет удобный и функциональнаый отладчик и средства для рефакторинга кода, что позволяет быстро находить и исправлять ошибки [7].

3.3 Листинги кода

В листингах 3.1 - **??** представлены реализации рассматриваемых алгоритмов.

Листинг 3.1 – Реализация алгоритма сортировки вставками

```
def insertion_sort(arr, n):
  for i in range(1, n):
    j = i
    cur = arr[j]
    while arr[j - 1] > cur and j > 0:
      arr[j] = arr[j - 1]
      j -= 1
    arr[j] = cur
  return arr
```

Листинг 3.2 – Реализация алгоритма сортировки пузырьком

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

Листинг 3.3 – Реализация алгоритма шейкерной сортировки

```

def shaker_sort(arr, n):
  left = 0
  right = n - 1
  while left <= right:
    for i in range(right, left, -1):
      if arr[i - 1] > arr[i]:
        arr[i - 1], arr[i] = arr[i], arr[i - 1]
    left += 1
    for i in range(left, right, 1):
      if arr[i] > arr[i + 1]:
        arr[i + 1], arr[i] = arr[i], arr[i + 1]
    right -= 1

  return arr
```

3.4 Тестирование

В таблице 3.1 приведены функциональные тесты для реализаций алгоритмов сортировок. Все тесты пройдены успешно каждой реализацией.

Таблица 3.1 – Тесты

Исходный массив	Ожидаемый результат
1	1
4, 4, 4, 4	4, 4, 4, 4
-1, 1, 2, 2, 10	-1, 1, 2, 2, 10
9, 8, 0	0, 8, 9
4, 5, 2, -1, 5, 6	-1, 2, 4, 5, 5, 6

Вывод

Был производен выбор средств реализации, реализованы и протестированы алгоритмы сортировок (вставками, пузырьком и шейкером), а также приведены требования к ПО.

4 Исследовательская часть

4.1 Пример работы

На рисунке 4.1 приведен пример работы программы.

```
Введите массив целых чисел (в одну строку, через пробелы):
 -1 9 83 193 1 3 5 -45 6

Пузырьковая сортировка:
До:
-1 9 83 193 1 3 5 -45 6
Ответ:
-45 -1 1 3 5 6 9 83 193
Время: 0.0

Шейкерная сортировка:
До:
-1 9 83 193 1 3 5 -45 6
Ответ:
-45 -1 1 3 5 6 9 83 193
Время: 0.0

Сортировка вставками:
До:
-1 9 83 193 1 3 5 -45 6
Ответ:
-45 -1 1 3 5 6 9 83 193
Время: 0.0
```

Рисунок 4.1 – Пример работы программы

4.2 Технические характеристики

Технические характеристики устройства, на котором выполнялось тестирование:

- операционная система: Windows 10;
- оперативная память: 16 Гб;
- процессор: Intel® Core™ i5-8259U.

Во время тестирования ноутбук был включен в сеть питания и нагружен только встроенными приложениями окружения и системой тестирования.

4.3 Сравнение времени выполнения реализаций алгоритмов

Все реализации алгоритмов сравнивались на трех видах массивов: упорядоченный (лучший случай - все реализации выдают наименьшее время работы), упорядоченный в обратном порядке (худший случай - все реализации выдают наибольшее время работы), случайно сгенерированный массив (произвольный случай). При этом для каждого класса массивов создавались массивы длиной от 0 до 10000 с шагом, увеличивающимся по мере роста длины массива.

Так как сортировки выполняются достаточно быстро, а замеры времени имеют некоторую погрешность, они для каждых класса и длины массивы и каждой реализации алгоритма сортировки выполнялись 10 раз, а затем вычислялось среднее время работы.

На рисунке 4.2 приведены результаты сравнения времени работы всех реализаций на всех массивах.

На рисунках 4.3 - 4.5 приведены результаты сравнения времени работы всех реализаций на отдельных классах массивов.

На рисунках 4.6 - 4.8 приведены результаты сравнения времени работы каждой реализации на всех классах массивов.

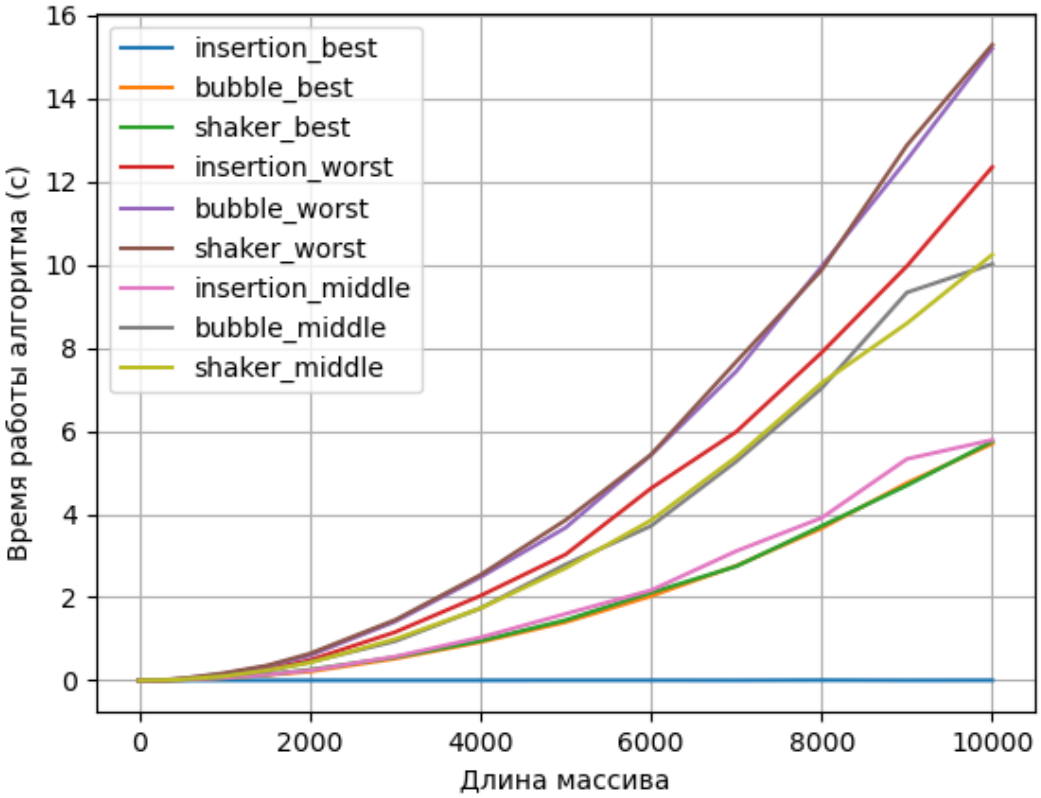

Рисунок 4.2 – Сравнение времени работы реализаций всех алгоритмов на всех массивах

Вывод

Реализации алгоритмов нахождения расстояния Дамерау — Левенштейна по времени выполнения сопоставимы с реализациями алгоритмом нахождения расстояния Левенштейна, хотя и немного уступают вторым в связи с дополнительной проверкой, позволяющей находить ошибки пользователя, связанные с неверным порядком букв. Однако эта операция зачастую позволяет найти более короткое расстояние между строками.

Рекурсивные реализации алгоритмов нахождения расстояний Левенштейна и Дамерау-Левенштейна, не использующие кеширование, работают на порядок дольше итеративных реализаций. При применении кеширования они требуют меньше времени, однако все равно уступают по производительности итеративным алгоритмам, особенно при большой длине строк.

Но по расходу памяти итеративные реализации проигрывают рекурсивным: максимальный размер используемой памяти в них пропорционален

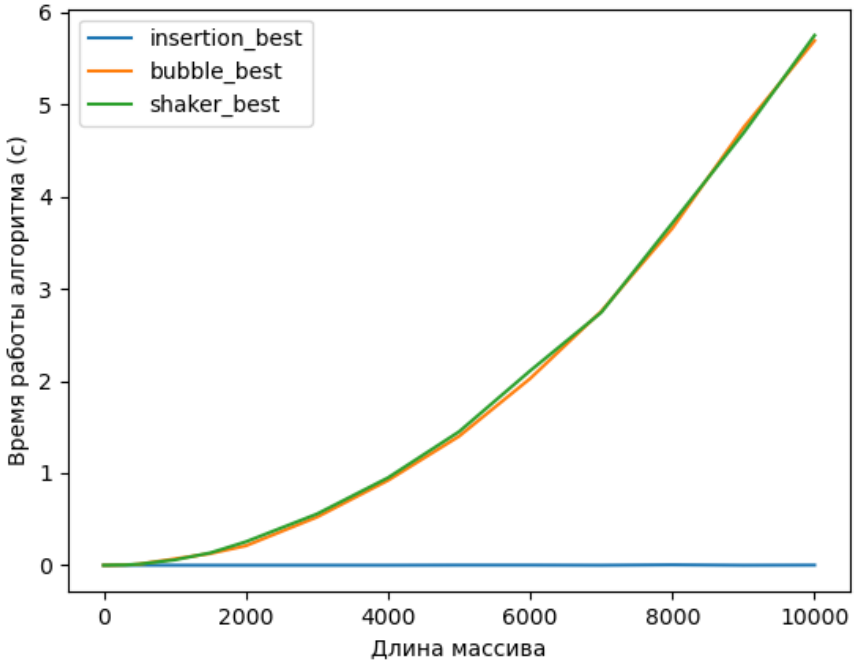

Рисунок 4.3 – Сравнение времени работы реализаций всех алгоритмов на упорядоченных массивах

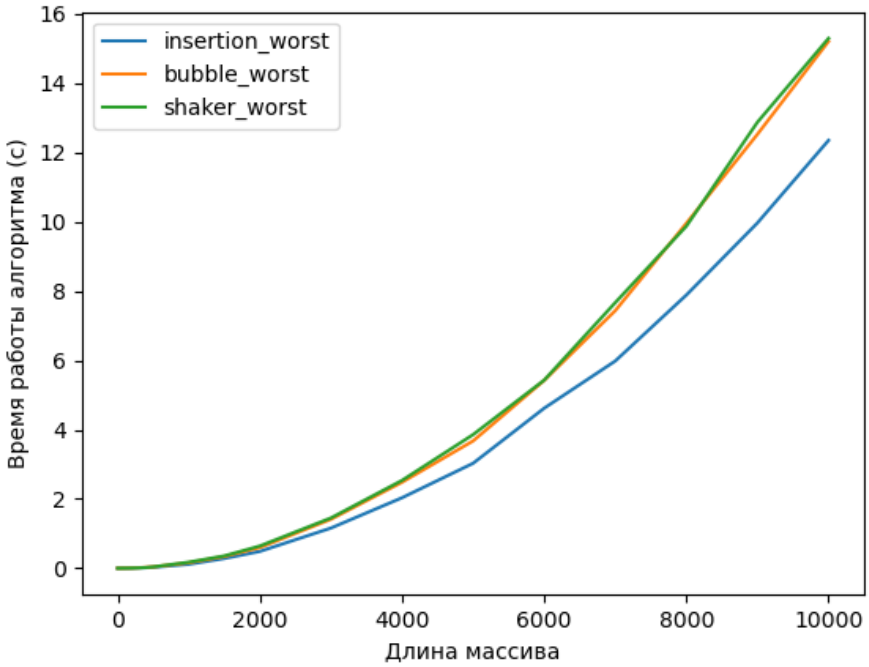

Рисунок 4.4 – Сравнение времени работы реализаций всех алгоритмов на упорядоченных в обратном порядке массивах

произведению длин строк, в то время как в рекурсивных — сумме длин строк.

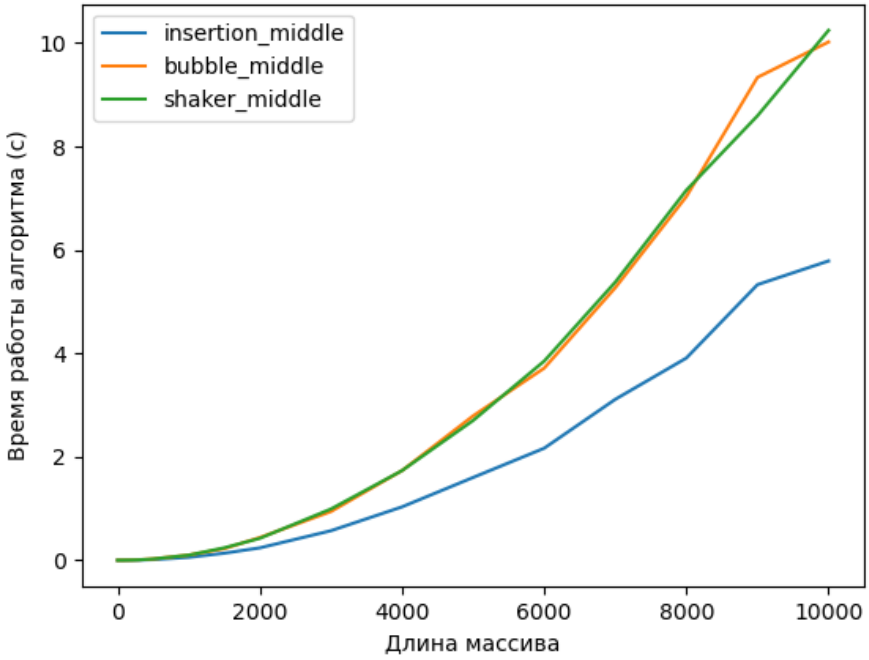

Рисунок 4.5 – Сравнение времени работы реализаций всех алгоритмов на произвольных массивах

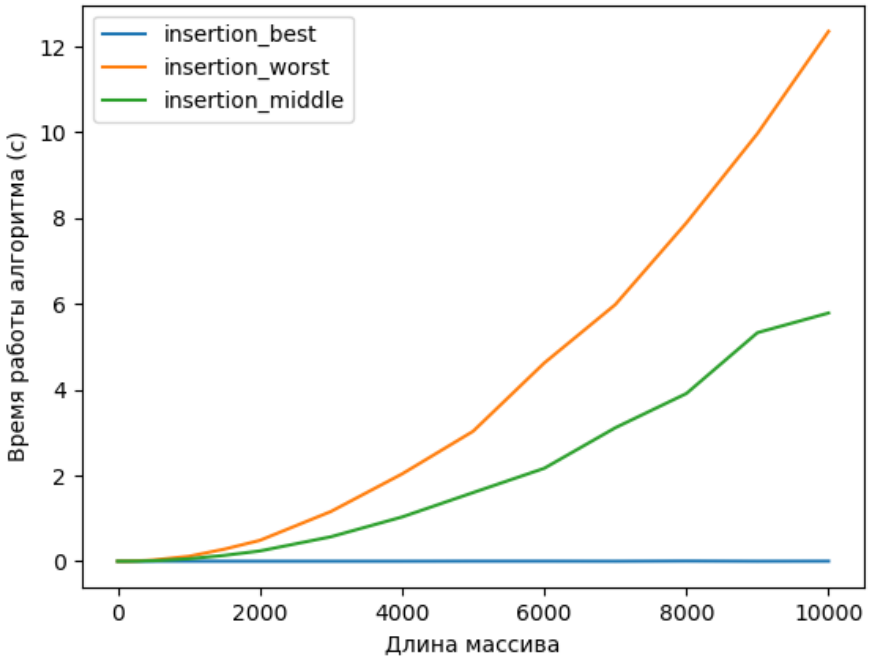

Рисунок 4.6 – Сравнение времени работы реализации алгоритма сортировки вставками на всех классах массивов

Если же применить к итеративным реализациям оптимизацию по памяти, то они будут выигрывать как по пиковой затрачиваемой памяти, так

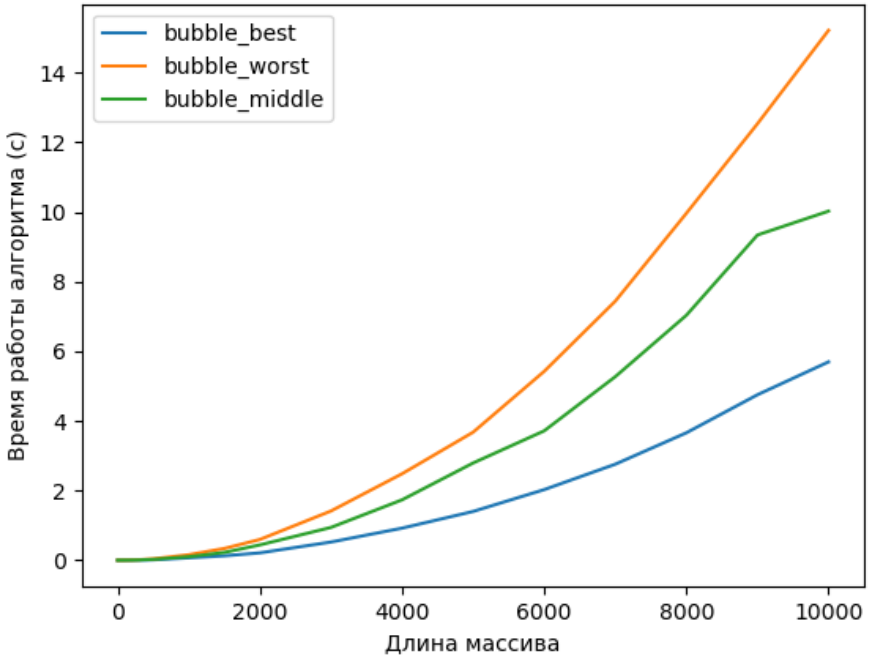

Рисунок 4.7 – Сравнение времени работы реализации алгоритма сортировки пузырьком на всех классах массивов

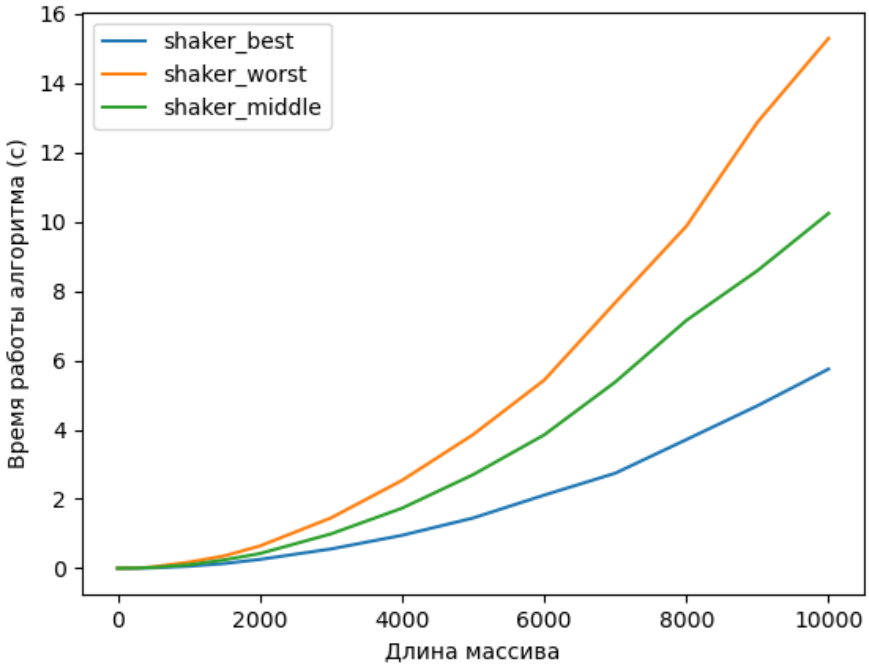

Рисунок 4.8 – Сравнение времени работы реализации алгоритма шейкерной сортировки на всех классах массивов

и по времени выполнения.

Заключение

В результате выполнения лабораторной работы были получены навыки оценки трудоемкости алгоритмов на примере трех алгоритмов сортировок.

В ходе выполнения лабораторной работы были выполнены следующие задачи:

1) изучены три алгоритма сортировок: сортировка вставками, сортировка пузырьком и шейкерная сортировка;

2) разработтаны и реализованы изученные алгоритмы;

3) проведен сравнительный анализ трудоёмкости реализаций алгоритмов на основе теоретических расчетов;

4) проведен сравнительный анализ процессорного времени выполнения реализаций алгоритмов на основе экспериментальных данных.

Литература

[1] Д.В. Шагбазян. Алгоритмы сортировки. Анализ, реализация, применение. Нижний Новгород: Нижегородский госуниверситет, 2019. Т. 42.

[2] Лекция 43: Алгоритмы сортировки массивов. Внутренняя сортировка [Электронный ресурс]. Режим доступа: https://intuit.ru/studies/courses/648/504/lecture/11472 (дата обращения: 13.09.2021).

[3] Сортировка вставками [Электронный ресурс]. Режим доступа: http://cppstudio.com/post/462/ (дата обращения: 13.09.2021).

[4] Алгоритм сортировки перемешиванием [Электронный ресурс]. Режим доступа: https://space-base.ru/library/algoritm/algoritm-sortirovki-peremeshivaniem-shejkernaya-dvunapravlennaya-pu (дата обращения: 13.09.2021).

[5] Лутц Марк. Изучаем Python, том 1, 5-е изд. Пер. с англ. — СПб.: ООО “Диалектика”, 2019. Т. 832.

[6] time — Time access and conversions [Электронный ресурс]. Режим доступа: https://docs.python.org/3/library/time.html (дата обращения: 05.09.2021).

[7] Python и Pycharm [Электронный ресурс]. Режим доступа: https://py-charm.blogspot.com/2017/09/pycharm.html (дата обращения: 05.09.2021).