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 №4 по курсу "Анализ алгоритмов"

Тема Параллельное программирование

Студент Зайцева А.А.

Группа ИУ7-52Б

Преподаватели Волкова Л.Л.

Москва — 2021 г.

Оглавление

Введение

Существуют различные способы написания программ, одним из которых является использование параллельных вычислений. Параллельные вычисления – способ организации компьютерных вычислений, при котором программы разрабатываются как набор взаимодействующих вычислительных процессов, работающих параллельно (одновременно) [1].

Основная цель параллельных вычислений – уменьшение времени решения задачи. Многие необходимые для нужд практики задачи требуется решать в реальном времени или для их решения требуется очень большой объем вычислений. Таким трудоемким алгоритмом является, например, алгоритм трассировки лучей - один из методов построения реалистичных сцен с учетом теней, эффектов отражения, преломленияи т.д..

Целью данной работы является изучение организации параллельных вычислений на базе алгоритма трассировки лучей.

В рамках выполнения работы необходимо решить следующие задачи:

1) изучить основы параллельных вычислений;

2) изучить алгоритм трассировки лучей;

3) разработать алгоритм трассировки лучей;

4) реализовать последовательную и параллельную трассировку лучей;

5) провести сравнительный анализ времени работы реализаций.

1 Аналитическая часть

В данном разделе будет приведена теория, необходимая для разработки и реализации последовательного и параллельного алгоритма трассировки лучей.

1.1 Алгоритм трасссировки ллучей

Алгоритм трассировки лучей работает в пространстве изображений и имеет 2 подхода: прямой и обратный.

Основная идея алгоритма прямой трассировки лучей состоит в том, что наблюдатель видит объекты благодаря световым лучам, которые испускает некоторый источник и которые падают на объект, отражаются, преломляются или проходят через него и в результате достигают зрителя. Если проследить за лучами, то становится понятно, что среди них лишь малая часть дойдет до наблюдателя, что показано на рисунке 1.1 слева, а значит большая часть вычислений произведена напрасно.

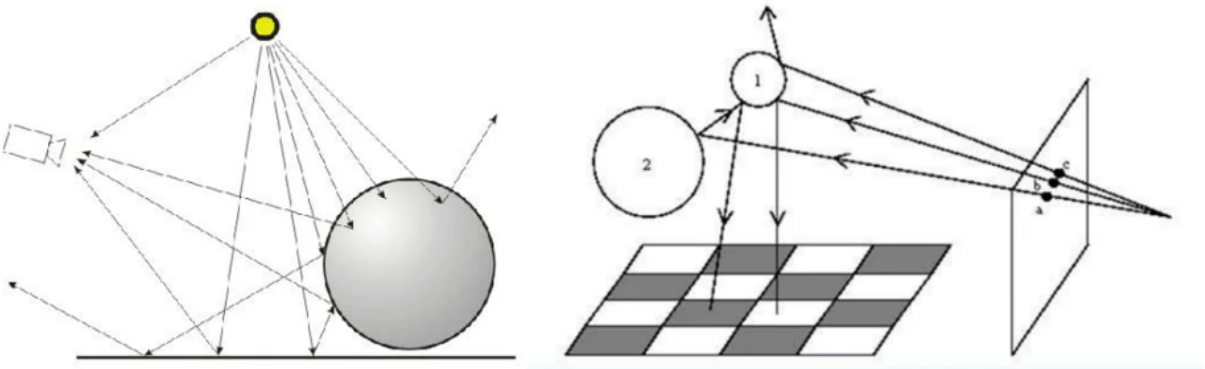

Рисунок 1.1 – Прямая и обратная трассировки лучей

Заменой данному алгоритму служит метод обратной трассировки лучей, который отслеживает лучи в обратном направлении. В ходе работы испускаются лучи от наблюдателя, как показано на рисунке 1.1 справа, и ищутся пересечения луча и всех объектов сцены.

Каждый раз, когда луч пересекает некоторую поверхность, из точки пересечения испускаются новые лучи - отраженный и преломленный. Пути

этих лучей отслеживаются по всей модели, и если лучи пересекают другие поверхности, то снова испускаются лучи.

В каждой точке, где луч пересекает поверхность, рисуется луч тени из точки пересечения к каждому источнику света. Если этот луч пересекает другую поверхность перед тем, как достигнуть источника света, то на ту поверхность, с которой был послан луч, падает тень с поверхности, блокирующей свет [2].

1.2 Параллельная реализация алгоритма трассировки лучей

Поскольку алгоритм обратной трассировки лучей обрабатывает каждый пиксель экрана независимо, можно использовать параллельные вычисления для уменьшения времени его работы, разбив экран на некоторые части. Наиболее часто используется горизонтальное или вертикальное разбиение (в данной работе используется второе) [3].

Вывод

В данном разделе были рассмотрены идеи и материалы, необходимые для разработки и реализации последовательного и параллельного алгоритма трассировки лучей.

2 Конструкторская часть

В данном разделе будет приведеная схема последедовательного алгоритма обратной трассировки лучей, а также схемы главного и рабочего потока для параллельной реализации этого алгоритма.

2.1 Схема последовательного алгоритма трассировки лучей

На рисунке 2.1 приведена схема последовательного алгоритма трассировки лучей.

2.2 Схема параллельного алгоритма трассировки лучей

На рисунке 2.2 приведена схема главного потока для параллельной реализации алгоритма трассировки лучей, который разбивает экран на вертикальные участки.

На рисунке 2.3 приведена схема рабочего потока для параллельной реализации алгоритма трассировки лучей.

Вывод

Были разработаны схемы базового и параллельного алгоритмов обратной трассировки лучей.

Рисунок 2.1 – Схема последовательного алгоритма трассировки лучей

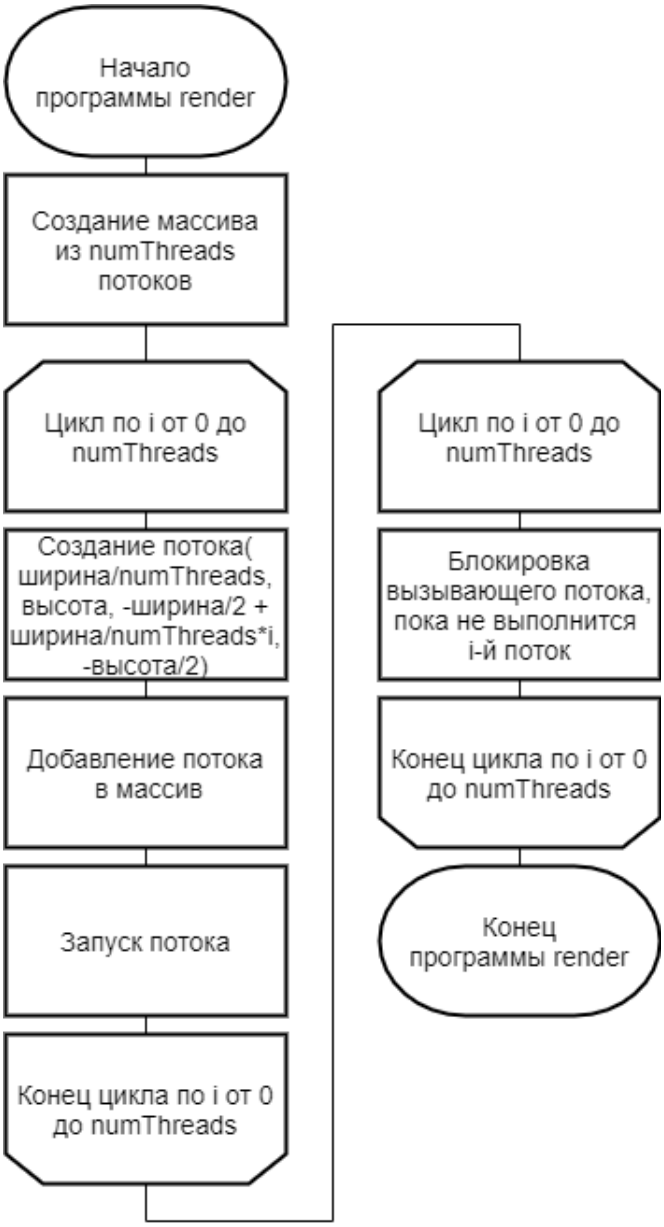

Рисунок 2.2 – Схема главного потока

Рисунок 2.3 – Схема рабочего потока

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 каждой реализацией процессорное время.

3.2 Выбор средств реализации

В качестве языка программирования для реализации данной лабораторной работы был выбран язык Python [4]. 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 [5].

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 [6].

3.3 Листинги кода

В листингах 3.1 - 3.3 представлены реализации рассматриваемых алгоритмов.

Листинг 3.1 – Реализация алгоритма сортировки вставками

```
1 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.

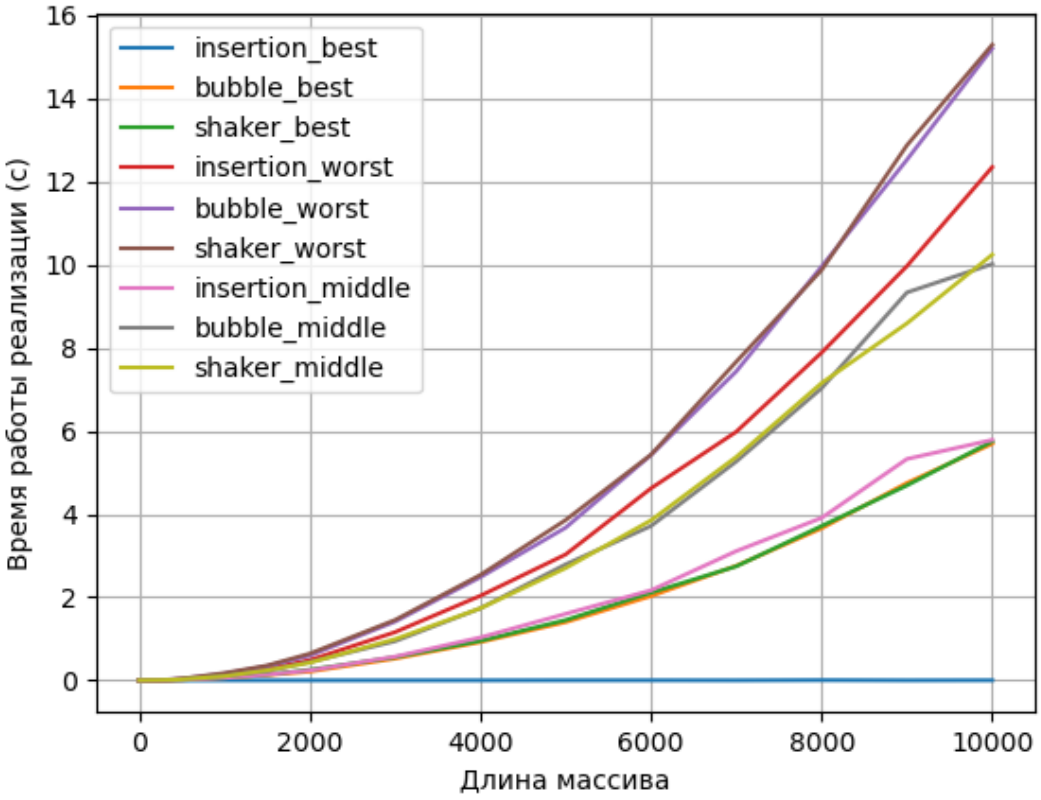

Рисунок 4.2 – Сравнение времени работы реализаций всех алгоритмов на всех массивах

4.4 Сравнение трудоемкости реализаций алгоритмов

Введем модель вычисления трудоемкости для оценки алгоритмов:

- базовые операции стоимостью 1: -, –, -=, +, ++, +=, ==, !=, <, >, <=, >=, », «, [];
- базовые операции стоимостью 2: *, *=, /, /=, %, %=;
- оценка трудоемкости цикла: Fц = Finit + Fсравн + N * (Fтела + Finc + Fсравн), где Finit, Fсравн, Fтела, Finc - трудоемкости инициализации, проверки условия цикла, инкрементирования и тела цикла, соответственно, а N - количество итераций;

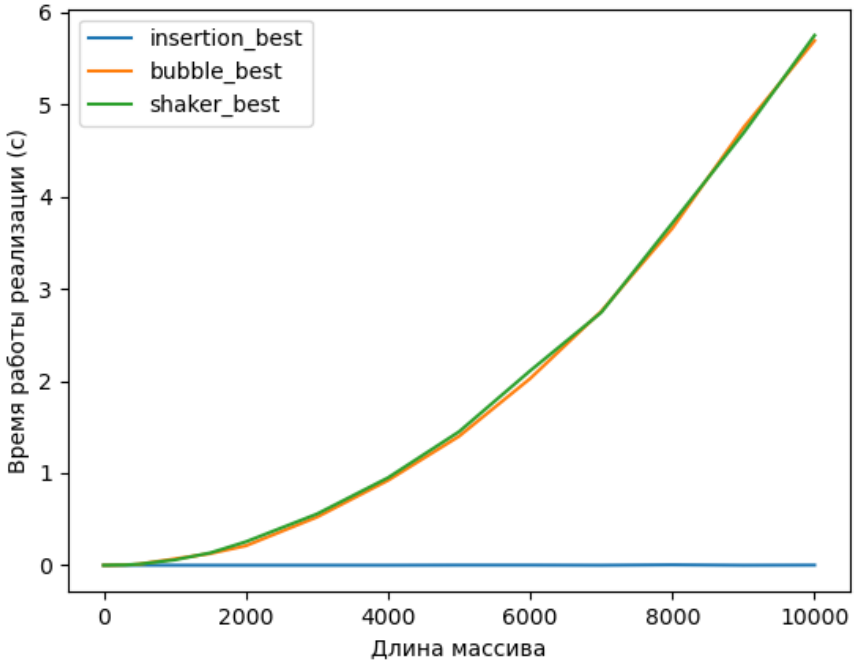

Рисунок 4.3 – Сравнение времени работы реализаций всех алгоритмов на упорядоченных массивах

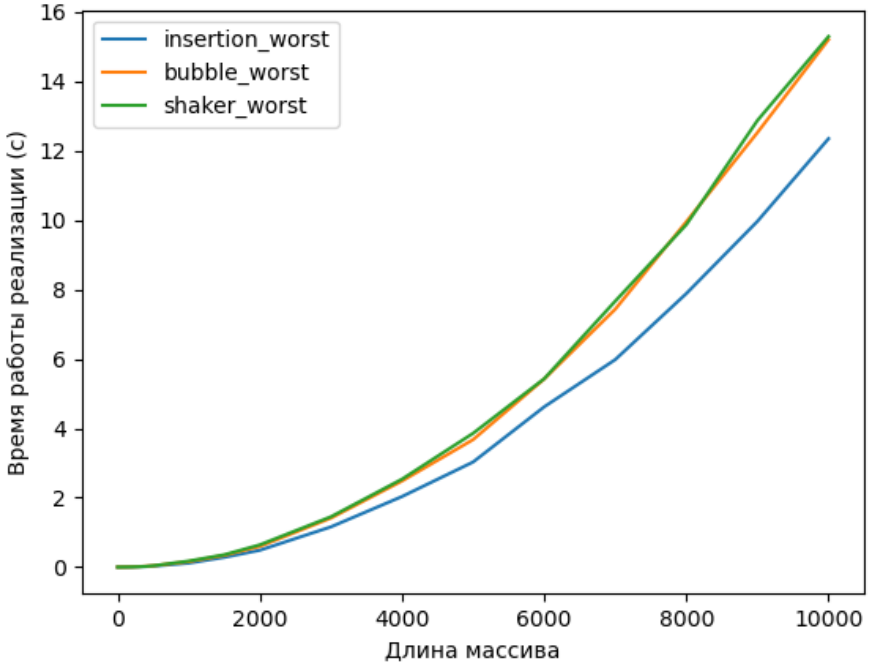

Рисунок 4.4 – Сравнение времени работы реализаций всех алгоритмов на упорядоченных в обратном порядке массивах

- стоимость условного операторoа: Fуо = Fсравн +

$$+ \left[\begin{array}{l} min(f1, f2) \\ max(f1, f2) \end{array} \right., \tag{4.1}$$

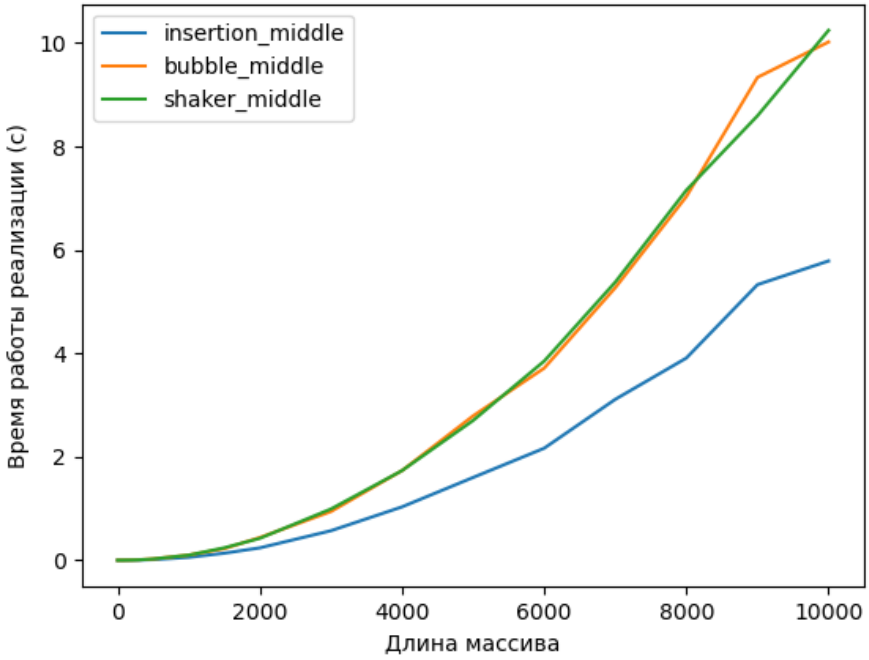

Рисунок 4.5 – Сравнение времени работы реализаций всех алгоритмов на произвольных массивах

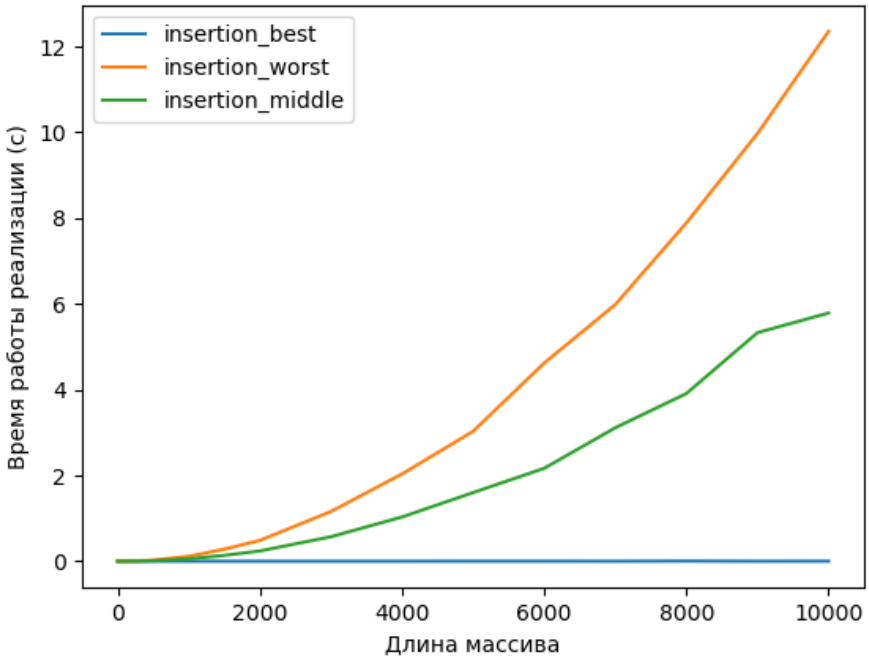

Рисунок 4.6 – Сравнение времени работы реализации алгоритма сортировки вставками на всех классах массивов

где Fсравн, f1, f2 - трудоемкости проверки условия, первого блока и второго блока, соответственно.

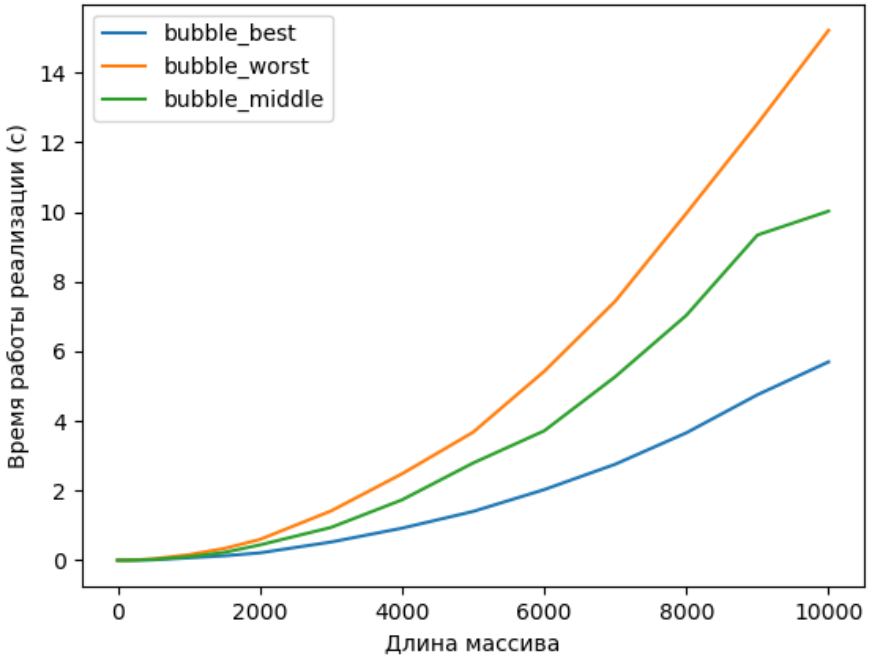

Рисунок 4.7 – Сравнение времени работы реализации алгоритма сортировки пузырьком на всех классах массивов

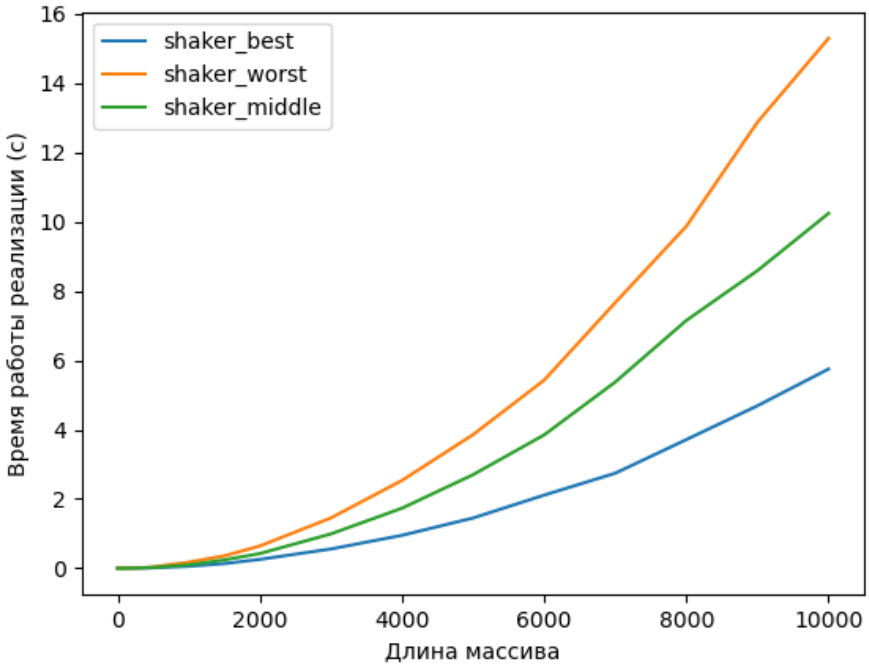

Рисунок 4.8 – Сравнение времени работы реализации алгоритма шейкерной сортировки на всех классах массивов

Оценим трудоемкость реализаций алгоритмов сортировки.

Рассмотрим трудоемкость реализации алгоритма сортировки вставка-

ми.

Лучший случай (отсортированный массив): алгоритм ни разу не войдет во внутренний цикл. Трудоемкость: $f(n) = 2 + (n-1)(2+1+2+4+2) = 11n - 9 = O(n)$.

Худший случай (отсортированный в обратном порядке массив): алгоритм будет полностью (пока не дойдет до начала массива) проходить внутренний цикл каждый раз . Трудоемкость: $f(n) = 2 + (n-1)(2+1+2+4+2+\frac{n}{2}(4+4+1)) = \frac{9n^2}{2} + \frac{13n}{2} - 9 = O(n^2)$.

Рассмотрим трудоемкость алгоритма сортировки пузырьком.

Лучший случай (отсортированный массив): не будет произведено ни одного обмена. Трудоемкость: $f(n) = 3 + (n-1)(3+\frac{n}{2}4) = 2n^2 + n = O(n^2)$.

Худший случай (отсортированный в обратном порядке массив): замена во внутреннем цикле будет производиться каждый раз. Трудоемкость: $f(n) = 3 + (n-1)(3 + \frac{n}{2}(4+9) = \frac{13n^2}{2} - \frac{7n}{2} = O(n^2)$.

Вывод

Реализация сортировки пузырьком квадратично зависит от количества элементов в массиве (как в лучшем, так и в худшем случае) и работает дольше всех остальных протестированных реализаций. Ее модификация - шейкерная сортировка - показывает схожие результаты, так как является модификацией только лишь по направлению движения.

Реализацией, которая работает меньшее количество времени, чем вышеупомянутые, является реализация алгоритма сортирвки вставками. Для нее в худшем случае время сортировки массива также квадратично зависит от количества элементов в нем, как и для алгоритмов пузырьковой и шейкерной сортировки. Но для лучшего случая время работы реализации алгоритма сортирвки вставками зависит от количества элементов линейно.

Таким образом, реализацией с наименьшим временем выполнения (по экспериментальным данным - во всех случаях) является сортировка вставками.

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) разработаны 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] Параллельные вычисления [Электронный ресурс]. Режим доступа: https://ru.bmstu.wiki/\T2A\CYRP\T2A\cyra\T2A\cyrr\T2A\cyra\T2A\cyrl\T2A\cyrl\T2A\cyre\T2A\cyrl\T2A\cyrsftsn\T2A\cyrn\T2A\cyrery\T2A\cyre_\T2A\cyrv\T2A\cyrery\T2A\cyrch\T2A\cyri\T2A\cyrs\T2A\cyrl\T2A\cyre\T2A\cyrn\T2A\cyri\T2A\cyrya (дата обращения: 05.10.2021).

[2] Роджерс Д. Математические основы машинной графики. М.: Мир, 1989. Т. 512.

[3] Параллельная обработка в алгоритме визуализации с трассировкой лучей. [Электронный ресурс]. Режим доступа: https://cyberleninka.ru/article/n/parallelnaya-obrabotka-v-algoritmah-vizualizatsii-s-trassirovkoy-lu (дата обращения: 05.10.2021).

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

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

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