Анализ Алгоритмов

Лабораторная работа №3

«Сравнение алгоритмов сортировок»

Юмаев Артур Русланович

ИУ7-55

Преподаватель: Волкова Л.Л.

Оглавление

[Введение 3](#_Toc23456546)

[Постановка задачи 4](#_Toc23456547)

[1. Аналитическая часть 5](#_Toc23456548)

[Сортировка пузырьком 5](#_Toc23456549)

[Сортировка вставками 5](#_Toc23456550)

[Быстрая сортировка 6](#_Toc23456551)

[2. Конструкторская часть 7](#_Toc23456552)

[Оценка трудоемкости алгоритмов 7](#_Toc23456553)

[Сортировка пузырьком 7](#_Toc23456554)

[3. Технологическая часть 9](#_Toc23456555)

[4. Исследовательская часть 11](#_Toc23456556)

[Заключение 14](#_Toc23456557)

# Введение

Сортировка - преобразование последовательности элементов в неубывающую (или невозрастающую) последовательность. Последовательность элементов называют неубывающей, если для любых i и j, таких что выполняется неравенство . Для строго возрастающей последовательности неравенство принимает вид . Аналогичным образом определяется невозрастающая и убывающая последовательности.

Обычно данные представлены в виде массива из N элементов, где элементы - это чаще всего числа или строки. Для этого существует множество алгоритмов, которые с помощью замены значений элементов исходного массива приводят его к отсортированному виду.

Для лучшего понимания того, в каких случаях нужно применить тот или иной алгоритм необходимо знать, что понимают под показателем сложности алгоритма. Речь идет о том, как зависит число операций, которые нужно произвести согласно алгоритму от объема данных, в нашем случае от количества элеменов массива N. Сложность задачи может быть логарифмической, линейной, квадратичной, экспонициальной и т.д., где для решения задачи необходимо выполнение операций соответственно. Например, квадратичный порядок сложности означает, что задача может использовать операций, а может и , здесь коэффициент перед не имеет значения: важен порядок, важно знать во сколько раз программа будет работать дольше, если число N увеличится вдвое, втрое или в 10 раз. В нашем случае независимо от этого коэффициента получим, что программа будет выполняться соответственно в 4, 9 и 100 раз дольше. Наилучшие универсальные алгоритмы сортировки имеют порядок сложности .

# Постановка задачи

Рассмотреть и реализовать три алгоритма сортировки: пузырьковый, вставками, быстрая. Оценить трудоёмкость одного из алгоритмов. Дать асимптотическую оценку двум остальным. Сравнить эффективность алгоритмов между собой путём проведения замеров времени на лучшем, худшем и среднем случаях. Сделать выводы.

# Аналитическая часть

В данном разделе будет приведено описание алгоритмов сортировок. Будут рассмотрены 3 алгоритма: самый базовый алгоритм сортировки пузырьком, алгоритм сортировки вставками и быстрая сортировка.

## Сортировка пузырьком

Алгоритм совершает 𝑠𝑖𝑧𝑒 − 1 проходов по массиву, в каждом проходе совершая перестановки соседних элементов, если их порядок не удовлетворяет критерию сортировки. По окончании всех проходов все элементы в массиве будут гарантированно упорядочены. Алгоритм считается одним из самых простых и неэффективных.

## Сортировка вставками

Проходится по массиву, вставляя каждый элемент в нужное место в отсортированный массив, находящийся местоположением до текущего обрабатываемого элемента. Алгоритм анализирует элементы от начала до конца массива и делает так, что каждый элемент при обработке смещается к началу массива до удовлетворения условия сортировки.

* Худшее асимптотическое время:
* Лучшее асимптотическое время:
* Среднее асимптотическое время:

## Быстрая сортировка

Алгоритм работает следующим образом: последовательность элементов в массиве разделяется на две составлящие опорным элементом, с которым ведётся сравнение. Далее массив отсортировывается последовательными сравнениями с опорным элементом так, чтобы взаимоотношение опорного элемента с частями слева и справа от него удовлетворяло условию.

* Худшее асимптотическое время:
* Лучшее асимптотическое время:
* Среднее асимптотическое время:

# 2. Конструкторская часть

## Оценка трудоемкости алгоритмов

Используется C-подобная модель оценки трудоёмкости.

Трудоёмкость операций:

1. +, −, =, + =, − =, <, > ==, ++
2. \*, /, %
3. [...](*доступ к элементу массива*)

Трудоемкость смены ячеек памяти местами будем считать 9, так как производится 3 обращения к памяти.

### Сортировка пузырьком

1. Внешний цикл:

|  |
| --- |
| for i in range(1, N) |

* Инициализация пемеренной для цикла: 1
* Количество сравненией:
* Количество инкрементов циклической переменной:
* Количество выполнений цикла:
* Подсчет суммы аргументов следующего цикла:
* Итого:

1. Второй внутренний цикл:

|  |
| --- |
| for j in range(1, N - i + 1) |

* Инициализация переменной для цикла: 1
* Кол-во выполнений цикла + сравнений + инкрементов:
* Итого:

1. Конструкция сравнения:

|  |
| --- |
| if a[j] < a[j - 1]:  tmp = a[j]  a[j] = a[j - 1]  a[j - 1] = tmp |

* Лучший случай:
  + Обращений к памяти: 2 \* 3
  + Арифмметических операций: 2 \* 1
  + Итого: 8 (не заходит в if)
* Худший случай:
  + Сравнение из лучшего случая: 8
  + Обращения к памяти: 4 \* 3 = 12
  + Загрузка в пямять: 3 \* 1 = 3
  + Разность: 2 \* 1 = 2
  + Итого: 8 + 12 + 3 + 2 = 25

Итоговая сложность (худший случай – элементы отсортированы в обратном порядке):

Итоговая сложность (лучший случай – элементы отсортированы):

Итоговая сложность (средний случай – половина отсортирована):

В среднем разность в лучшем случае алгоритм делает свою работу в 3 раза быстрее, чем в худшем случае.

# 3. Технологическая часть

В данном разделе будут приведены листинги для каждого из алгоритмов на языке Python.

Листинг 1 Сортировка пузырьком

|  |
| --- |
| def bubble\_sort(a):  N = len(a)    for i in range(1, N):  for j in range(1, N - i + 1):  if a[j] < a[j - 1]:  a[j], a[j - 1] = a[j - 1], a[j]  return a |

Листинг 2 Алгоритм сортировки вставками

|  |
| --- |
| def insert\_sort(a):  size = len(a)  for c in range(1, size):  tmp = a[c]  i = c - 1  while (i >= 0 and a[i] > tmp):  a[i + 1] = a[i]  a[i] = tmp  i -= 1  return a |

Листинг 3 Алгоритм быстрой сортировки

|  |
| --- |
| def quick\_sort(a):  "User-friendly wrapper"  def qs\_subroutine(a, left, right):  “Main quicksort algorithm”  i = left  j = right  pivot = a[(left + right) // 2]  while (i <= j):  while (a[i] < pivot):  i += 1  while (a[j] > pivot):  j -= 1  if (i <= j):  a[i], a[j] = a[j], a[i]  i += 1  j -= 1  if (left < j):  qs\_subroutine(a, left, j)  if (i < right):  qs\_subroutine(a, i, right)  qs\_subroutine(a, 0, len(a) - 1)  return a |

# 4. Исследовательская часть

В данном разделе будет приведено сравнение алгоритмов по скорости работе на примере массивов размерностей от 100 до 3000 с шагом 200. Также будут разные виды сравнения: сравнения на отсортированных массивах, сравнения на массивах отсортированных в обратном порядке и на массивах неотсортированных

Замер времени проводился с помощью библиотеки time в Python 3.7 и метода process\_time().

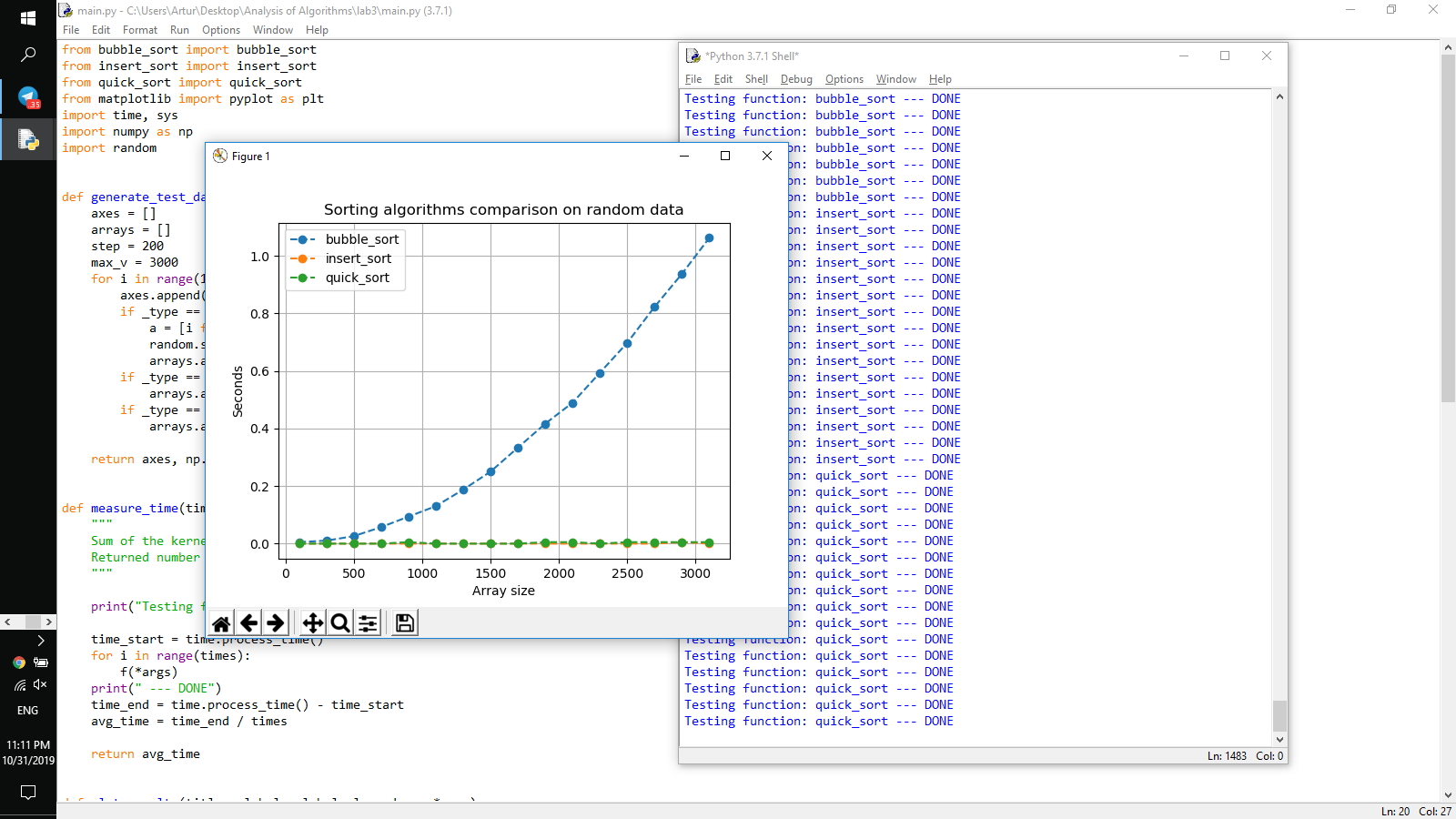


Рисунок 1 - Сравнение алгоритмов по скорости работы на несортированных данных

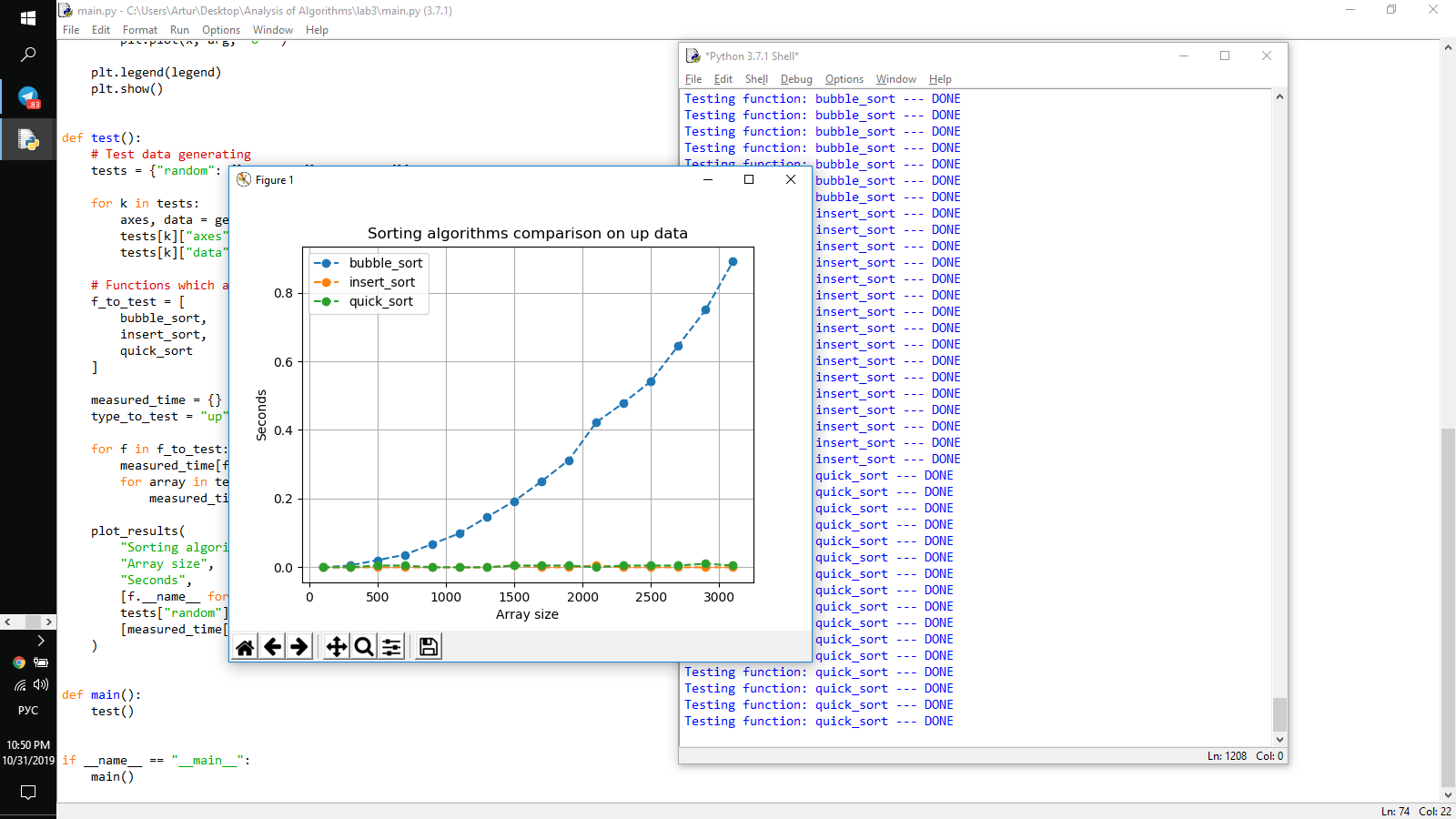


Рисунок 2 - Сравнение алгоритмов по скорости работы на сортированных данных

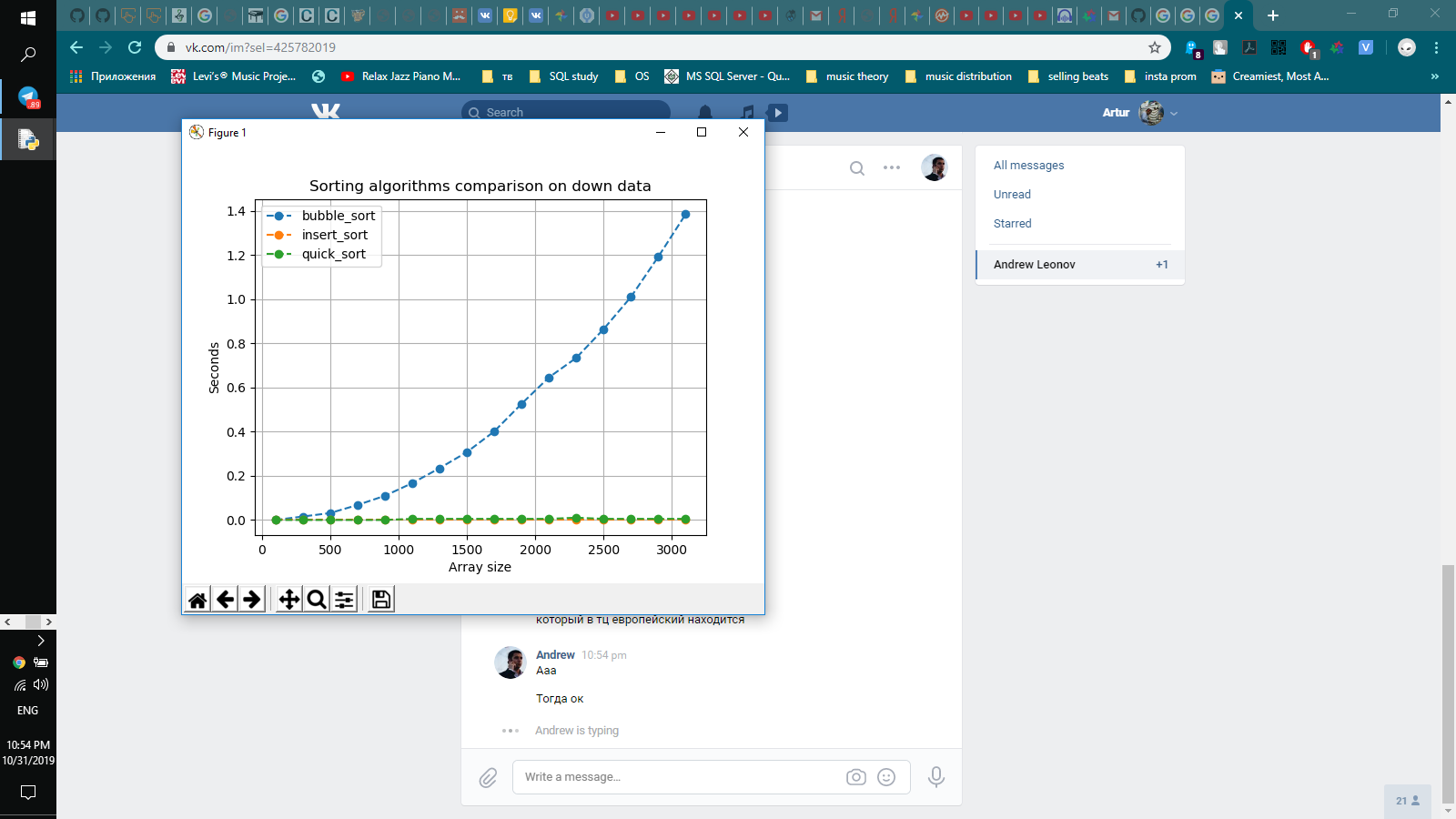


Рисунок 3 - Сравнение алгоритмов по скорости работы на сортированных в обратном порядке данных

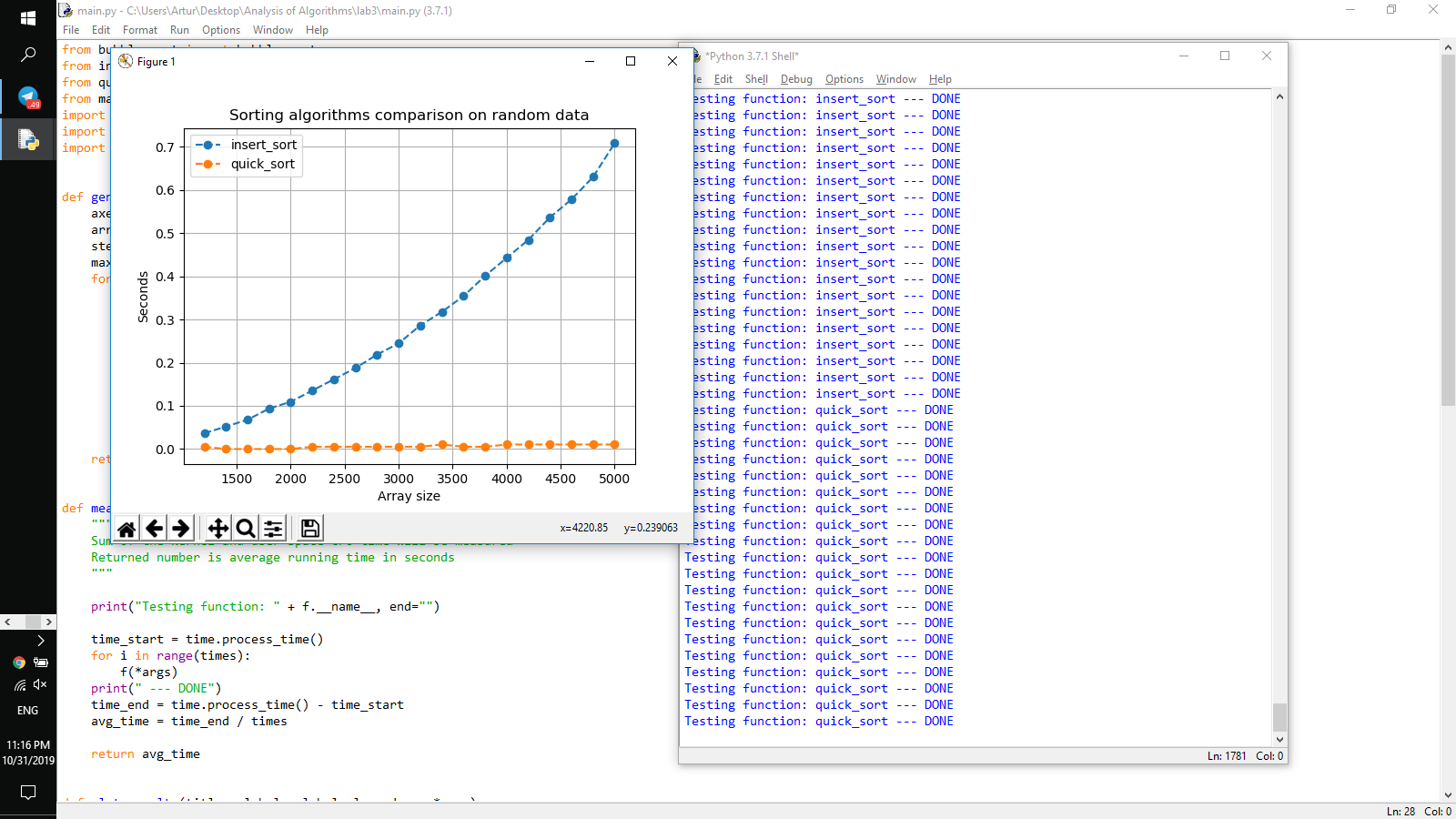


Рисунок 4 - Сравнение алгоритма быстрой сортировки и сортировки вставками на случайных данных

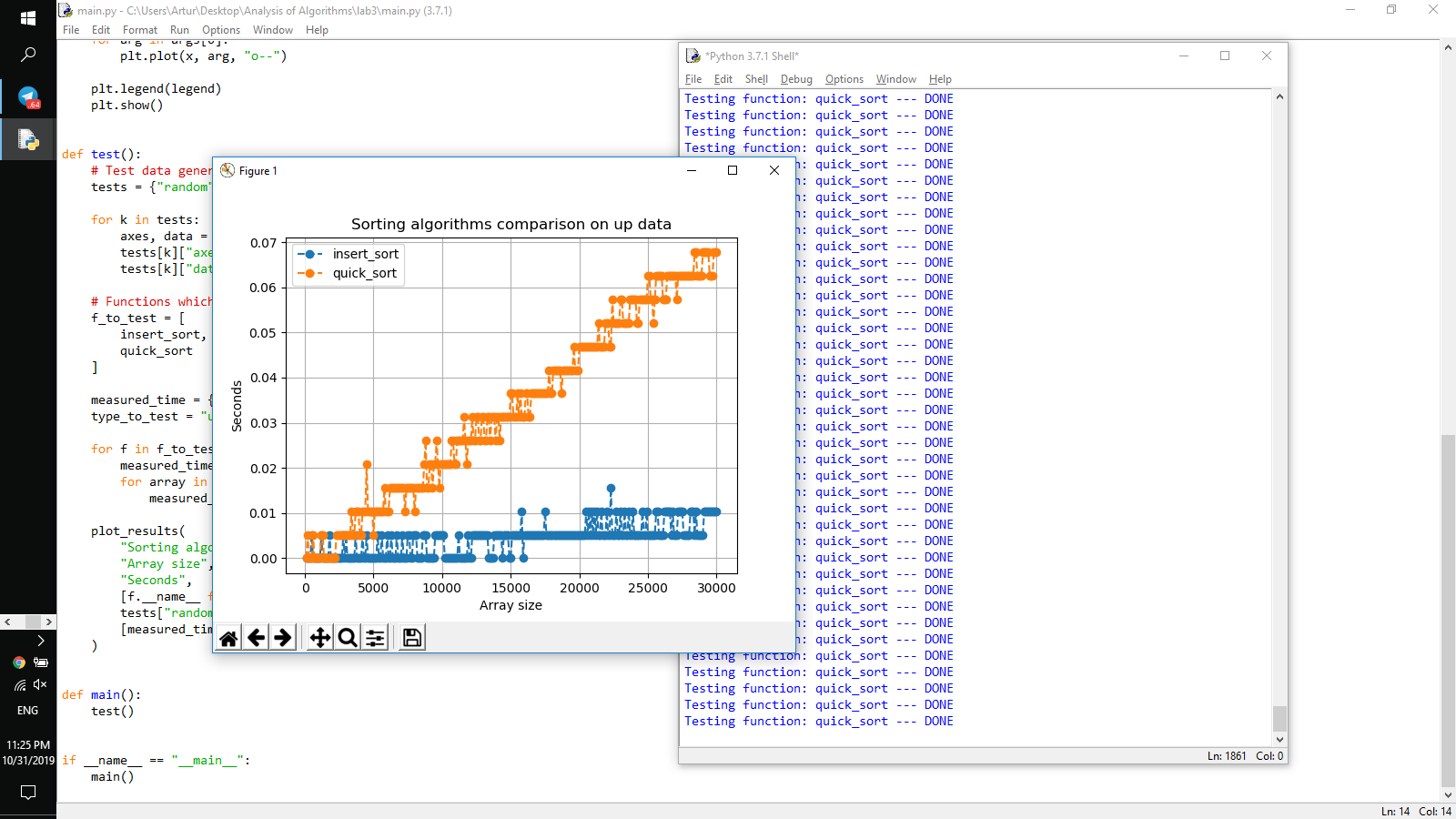


Рисунок 5 - Сравнение скорости алгоритмов быстрой сортировки и сортировки вставками на сортированных данных

Опыт показал, что самым быстрым является алгоритм быстрой сортировки – quick sort. Вторым по быстродействию идет алгоритм сортировки вставками и далее алгоритм пузырька. Алгоритм быстрой сортировки сохраняет почти нулевую скорость работы на размерностях до 5000.

В случае, если часто попадаются отсортированные (почти отсортированные) массивы, выгодно использовать сортировку вставками, которая показывает себя даже лучше, чем быстрая. Пузырёк не рекомендуется использовать, т. к. существуют гораздо более эффективные методы по асимптотике и на практике, что показали эксперименты (вставки быстрее в произвольном случае, несмотря на то, что асимптотика у обоих алгоритмов в этом случае одинаковая). В целом (выбирая среди трёх рассмотренных алгоритмов) рекомендуется использовать быструю сортировку.

# Заключение

Были рассмотренны и реализованы три алгоритма сортировки (пузырьковый, вставками, быстрая). Оценена трудоёмкость одного из алгоритмов. Дана асимптотическая оценка остальным. Алгоритмы были сравнены между собой путём проведения замеров времени на лучшем, худшем и среднем случаях по эффективности. Были сделаны выводы.