Анализ Алгоритмов

Лабораторная работа №3

«Сравнение алгоритмов сортировок»

Юмаев Артур Русланович

ИУ7-55

Преподаватель: Волкова Л.Л.

Оглавление

[Введение 3](#_Toc23870801)

[Постановка задачи 4](#_Toc23870802)

[1. Аналитическая часть 5](#_Toc23870803)

[Сортировка пузырьком 5](#_Toc23870804)

[Сортировка вставками 5](#_Toc23870805)

[Быстрая сортировка 6](#_Toc23870806)

[2. Конструкторская часть 7](#_Toc23870807)

[Оценка трудоемкости алгоритмов 7](#_Toc23870808)

[Сортировка пузырьком 7](#_Toc23870809)

[Схемы алгоритмов сортировки 9](#_Toc23870810)

[3. Технологическая часть 13](#_Toc23870811)

[4. Исследовательская часть 15](#_Toc23870812)

[Заключение 19](#_Toc23870813)

# 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.

# Постановка задачи

Цель: цассмотреть и реализовать три алгоритма сортировки: пузырьковый, вставками, быстрая. Задачи:

* 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 и сделать выводы.

# Аналитическая часть

В данном разделе будет приведено описание алгоритмов сортировок.

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 .

В частности, в данной работе будут рассмотрены 3 алгоритма: самый базовый алгоритм сортировки пузырьком, алгоритм сортировки вставками и быстрая сортировка.

## Сортировка пузырьком

Алгоритм совершает n – 1 проходов по массиву, в каждом проходе совершая перестановки соседних элементов, где n – длина массива. 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. Худший случай для алгоритма – данные отсортированны в обратном порядке. Лучший случай – данные отсортированы.

## 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.

* Худшая производительность по времени: [1];
* Лучшая производительность по времени: ;
* Средняя производительность по времени: ,

где n – длина массива.

## 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.

* Худшая производительность по времени: [2];
* Лучшая производительность по времени: ;
* Средняя производительность по времени: ,

где n – длина массива.

Логарифм в данной формуле появляется в следствии особенностей алгоритма, который похож на поиск в дереве.

# 2. Конструкторская часть

## Оценка трудоемкости алгоритмов

Используется C-подобная модель оценки трудоёмкости.

Трудоёмкость операций:

1. Трудоемкость операций: +, −, =, + =, − =, <, > ==, ++ равна 1.
2. Трудоемкость операций: \*, /, % равна 2.
3. Трудоемкость операции доступа к элементу памяти: [...] равна 3.

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

Цикл будет оцениваться о фактически выполненным операциям из перечня выше.

Условный оператор if будет фактически оценен как сумма стоимости операций в условии и трудоемкости различных ветвей (в лучшем случае и в худшем случае). Стоимость условного перехода из уловия в одну из ветвей решения полагается равной 0.

### 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:
* Сравнение во внутреннем цикле:
* Итого:

1. Второй внутренний цикл:

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

* Кол-во выполнений цикла + сравнений + инкрементов:
* Стоимость сравнения и инкремента:

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):

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

## Схемы алгоритмов сортировки

В данном разделе будут приведены схемы алгоритмов сортировки одномерных массивов.

На рисунке 1 изображена схема алгоритма сортировки пузырьком. Алгоритм является довольно простым, но в то же время неэффективным по времени.

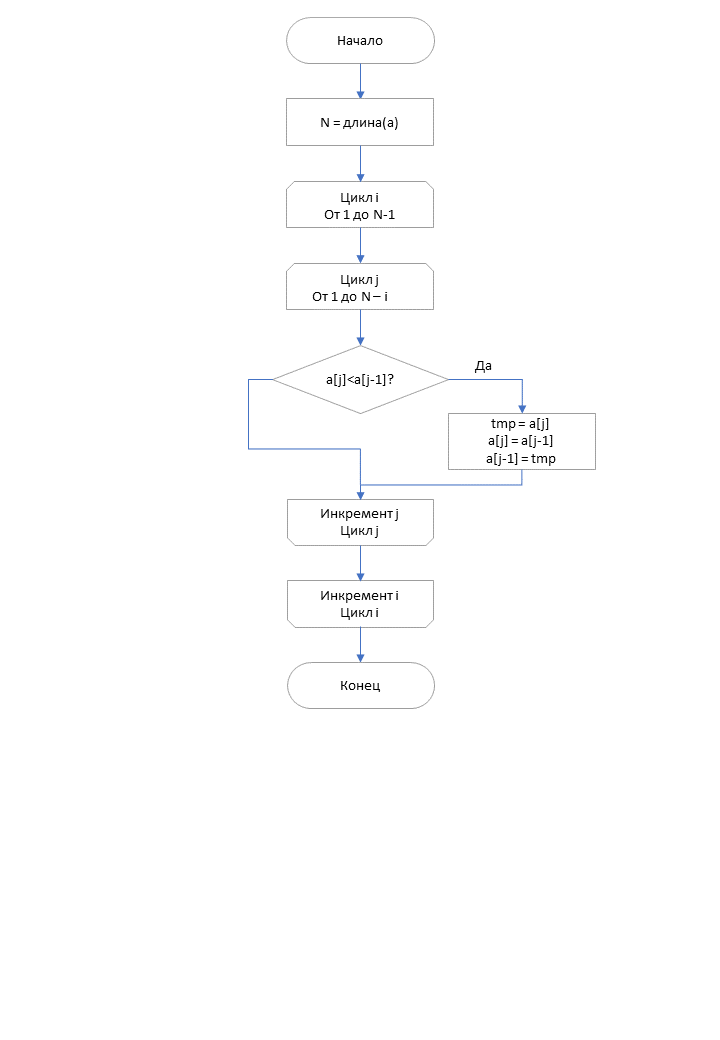


Рисунок 1 - Схема алгоритма сортировки пузырьком

На рисунке 2 изображена схема алгоритма сортировки одномерного массива вставками.

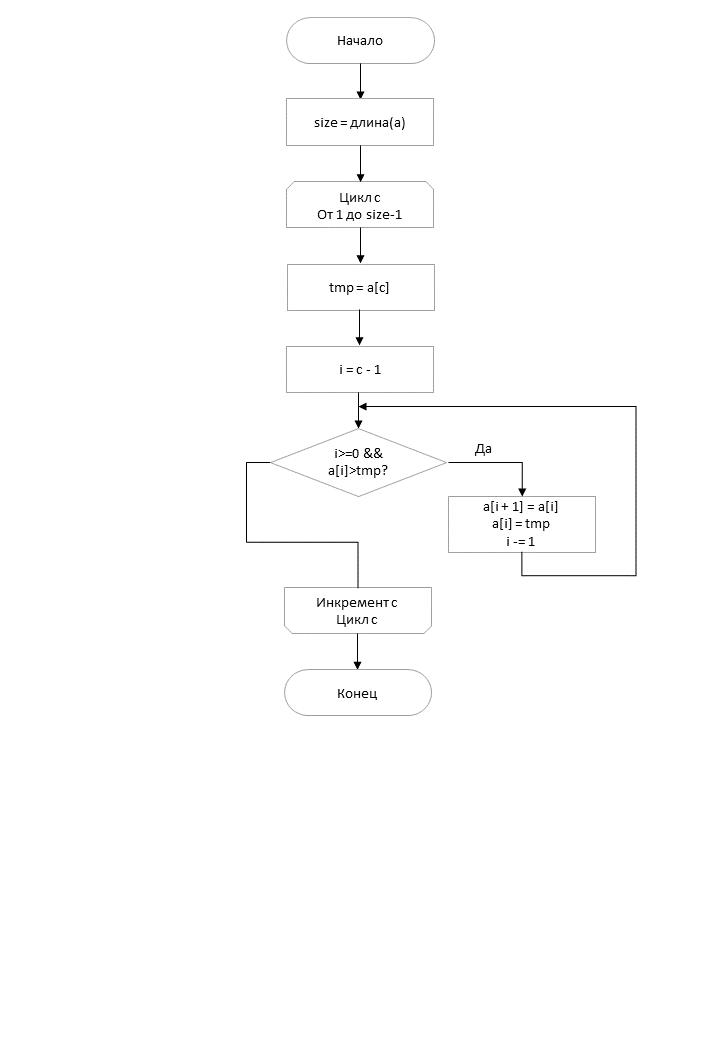


Рисунок 2 - Схема алгоритма сортировки вставками

На рисунке 3 изображена схема алгоритма быстрой сортировки. На данный момент он сочитает в себе быстроту и простоту реализации.

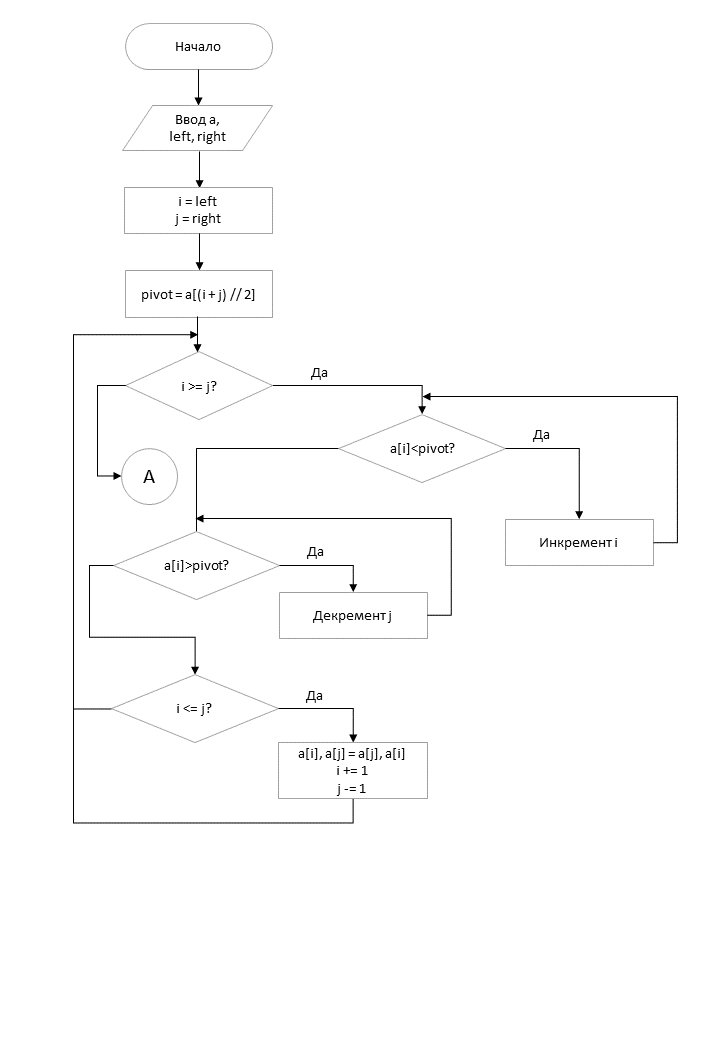


Рисунок 3 - Схема алгоритма быстрой сортировки. Часть А

На рисунке 4 изображена вторая часть алгоритма быстрой сортировки.

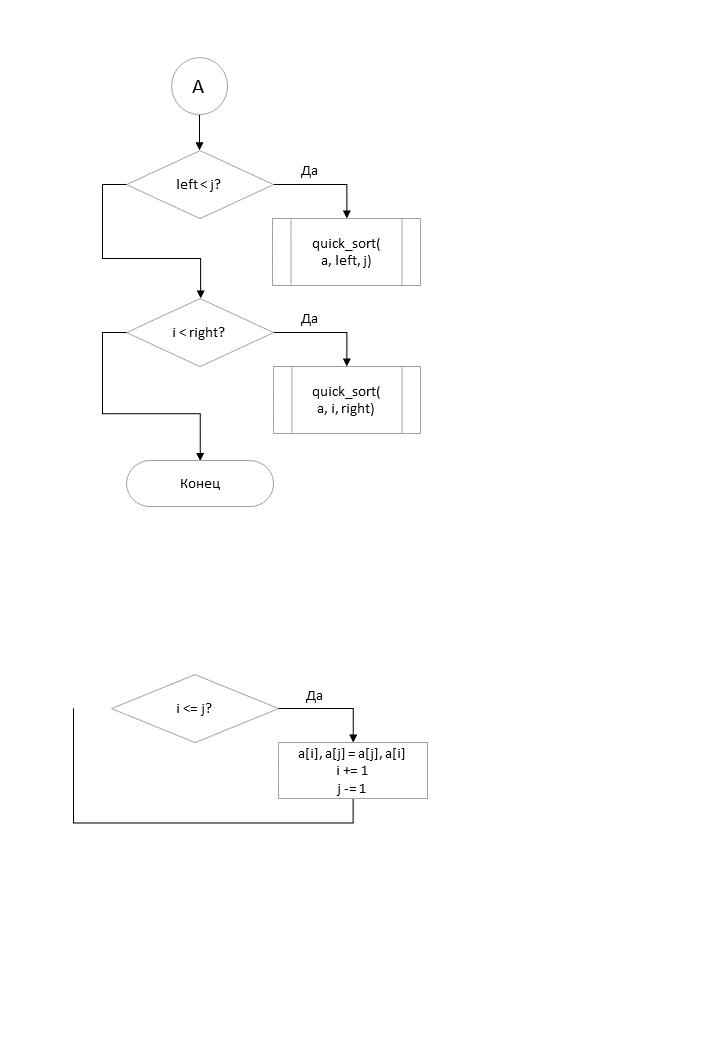


Рисунок 4 - Алгоритм быстрой сортировки. Часть Б

# 3. Технологическая часть

В данном разделе будут приведены листинги для каждого из алгоритмов на языке Python (листинги 1 – 3).

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 для тестирования алгоритмов при условии наступления худшего, лучшего и произвольного случаев.

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

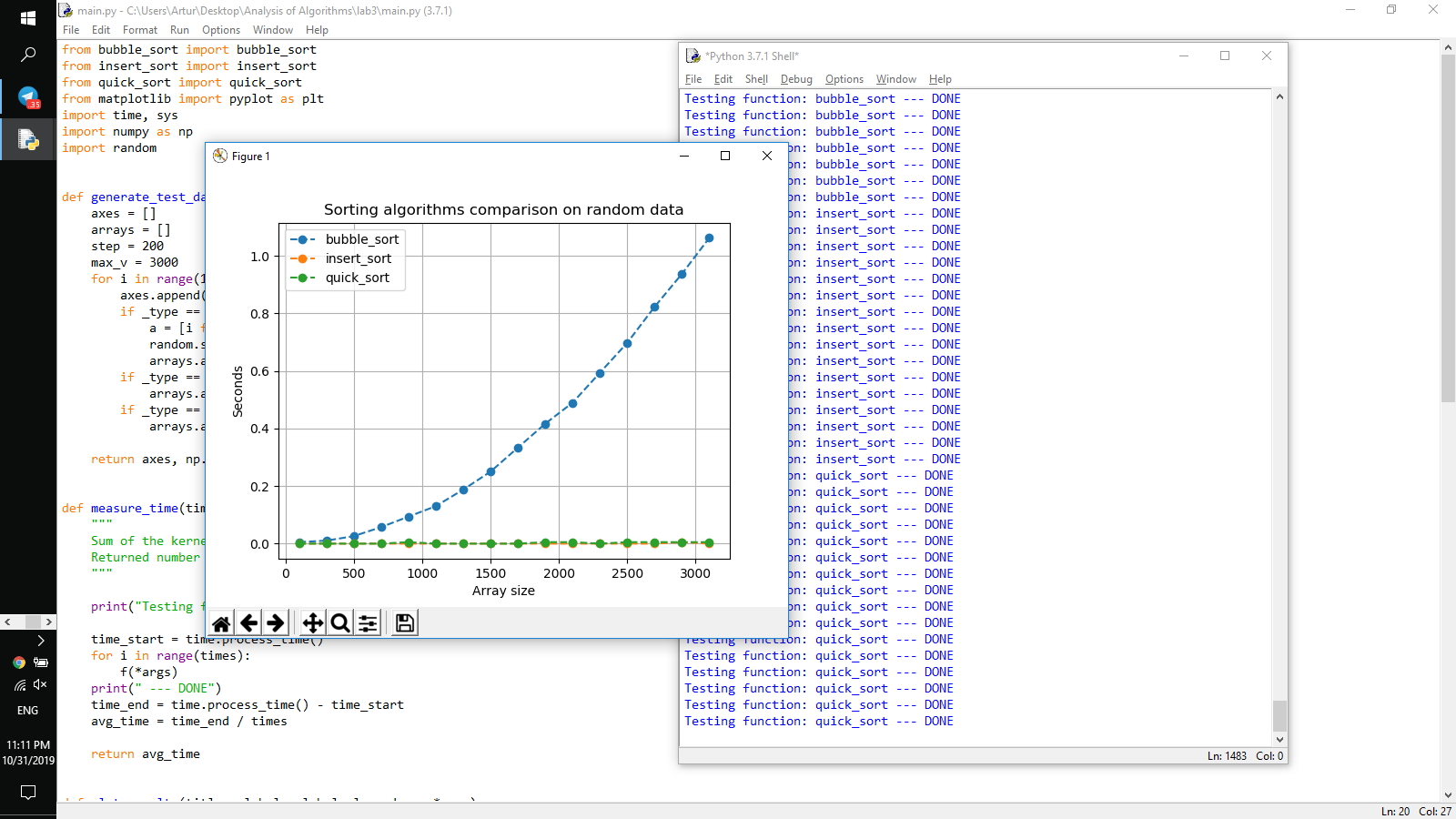


Рисунок 5 - Сравнение алгоритмов по скорости работы на несортированных данных

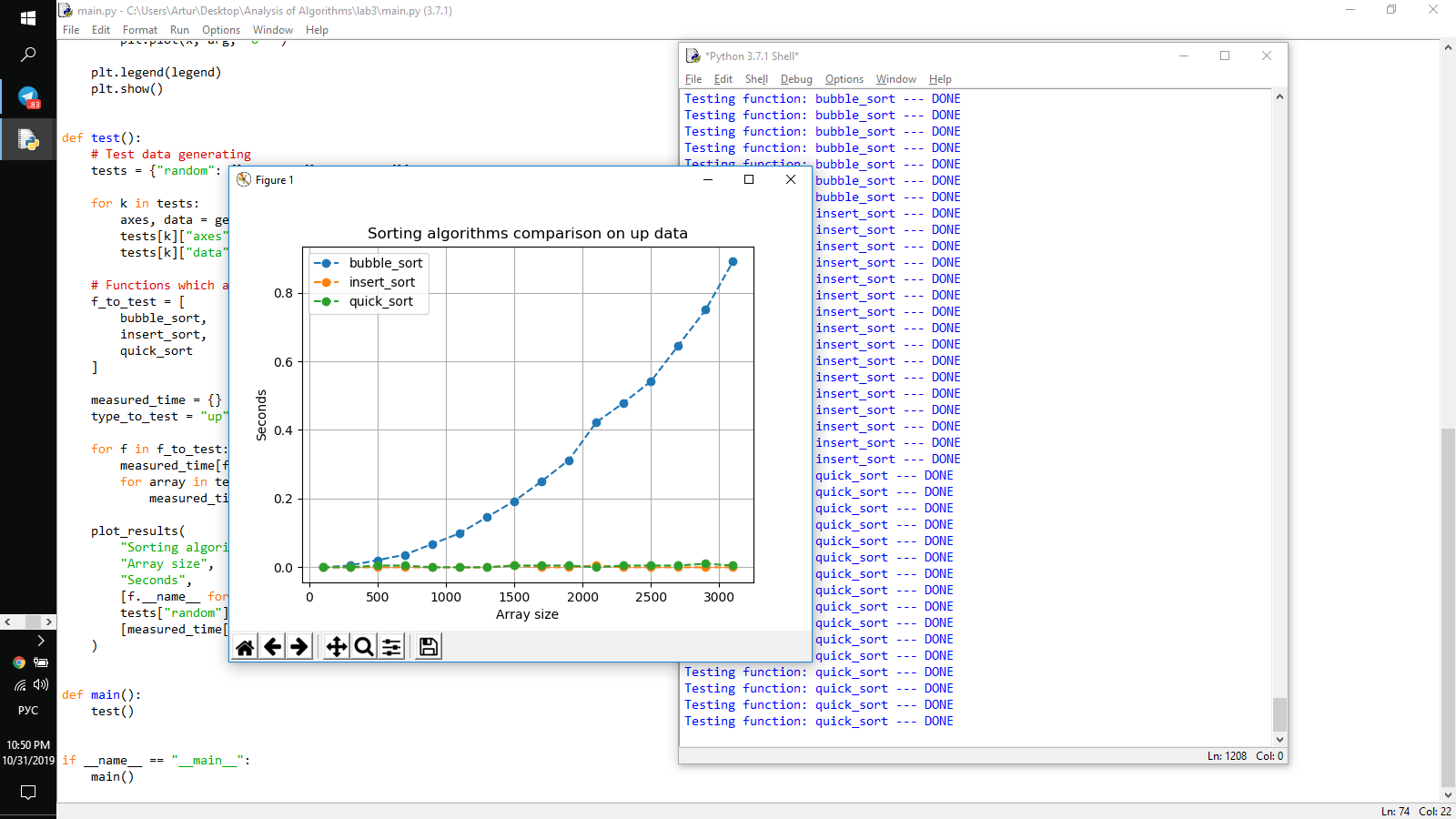


Рисунок 6 - Сравнение алгоритмов по скорости работы на сортированных данных

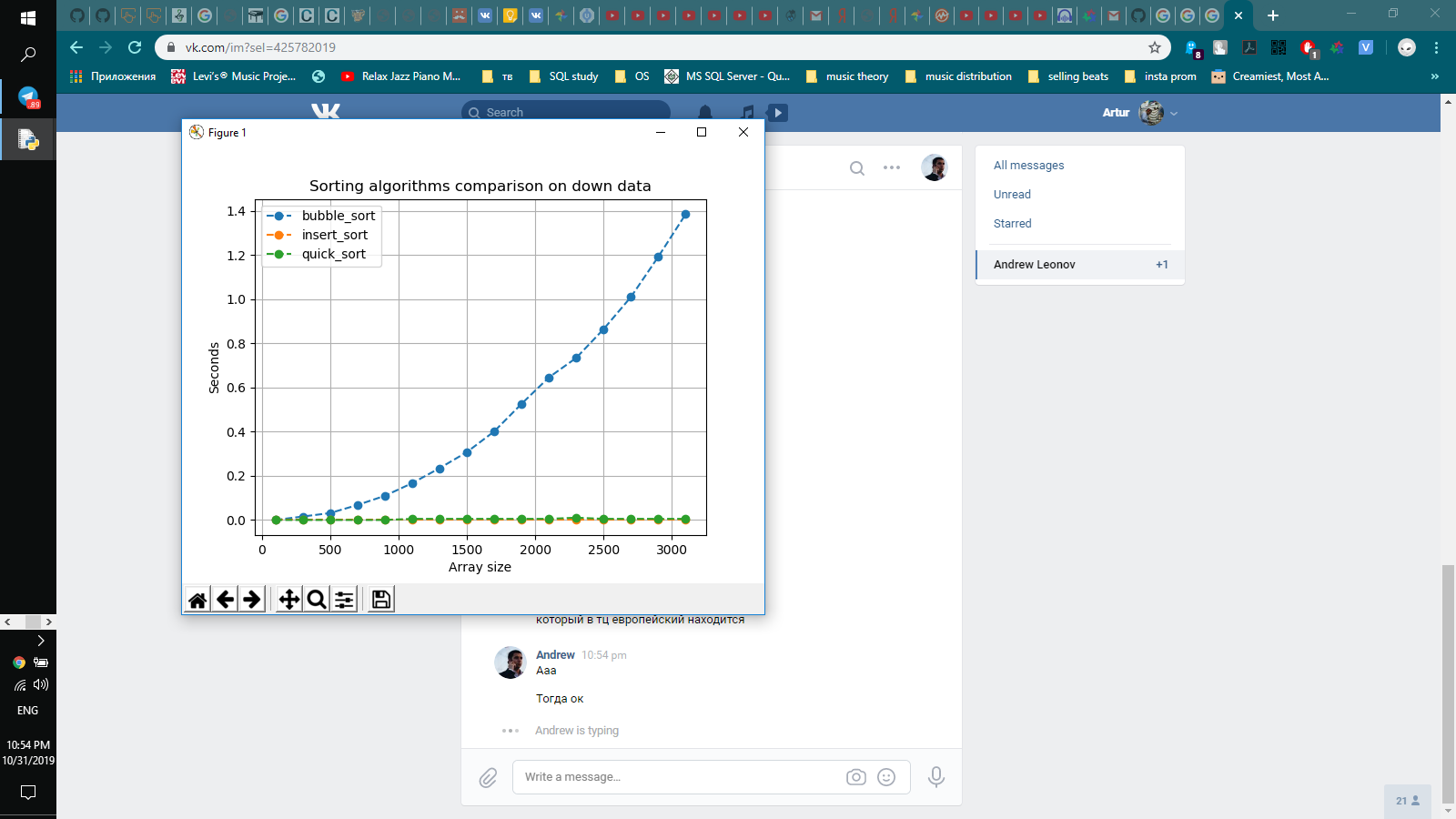


Рисунок 7 - Сравнение алгоритмов по скорости работы на сортированных в обратном порядке данных

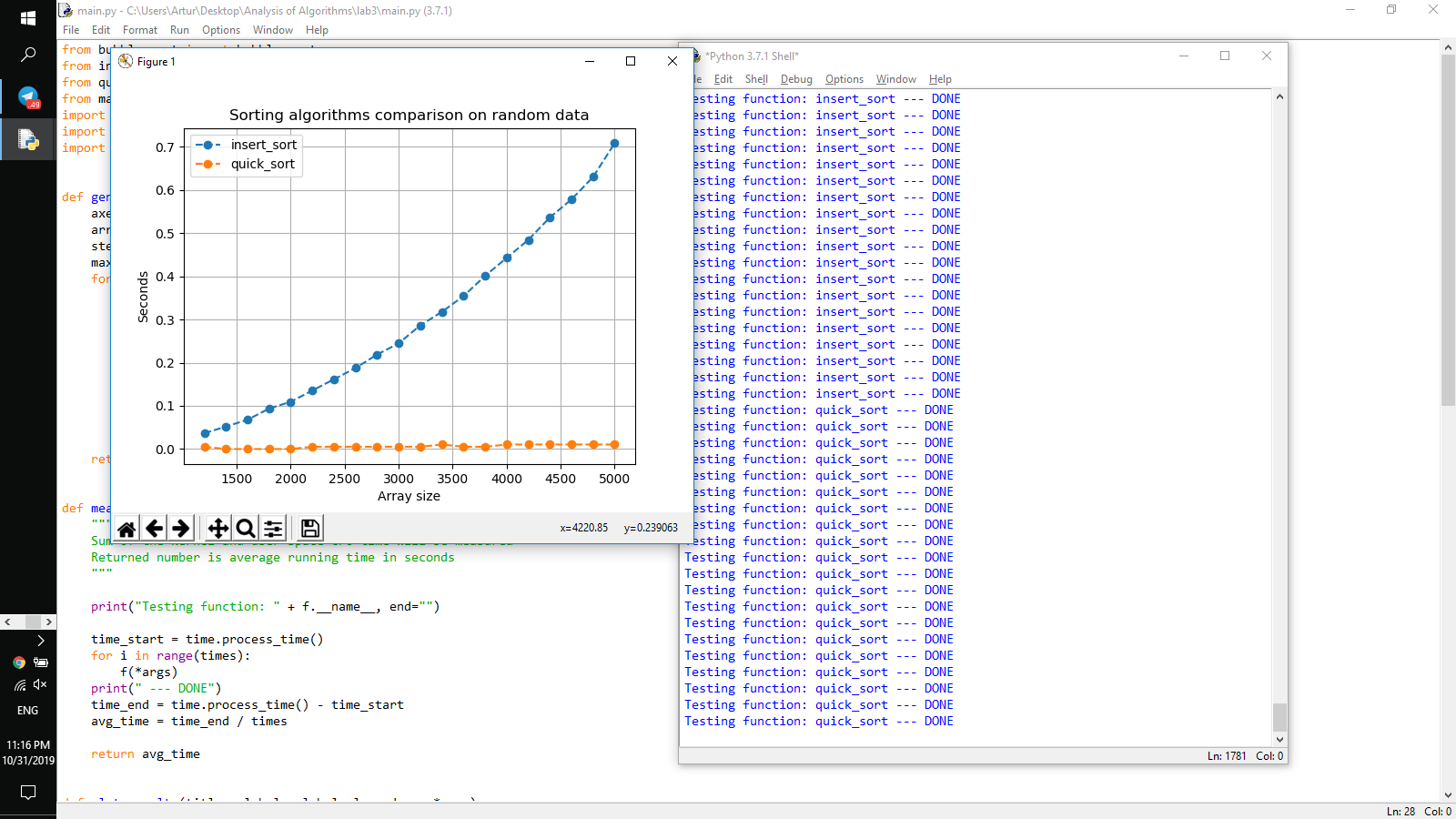


Рисунок 8 - Сравнение алгоритма быстрой сортировки и сортировки вставками на случайных данных

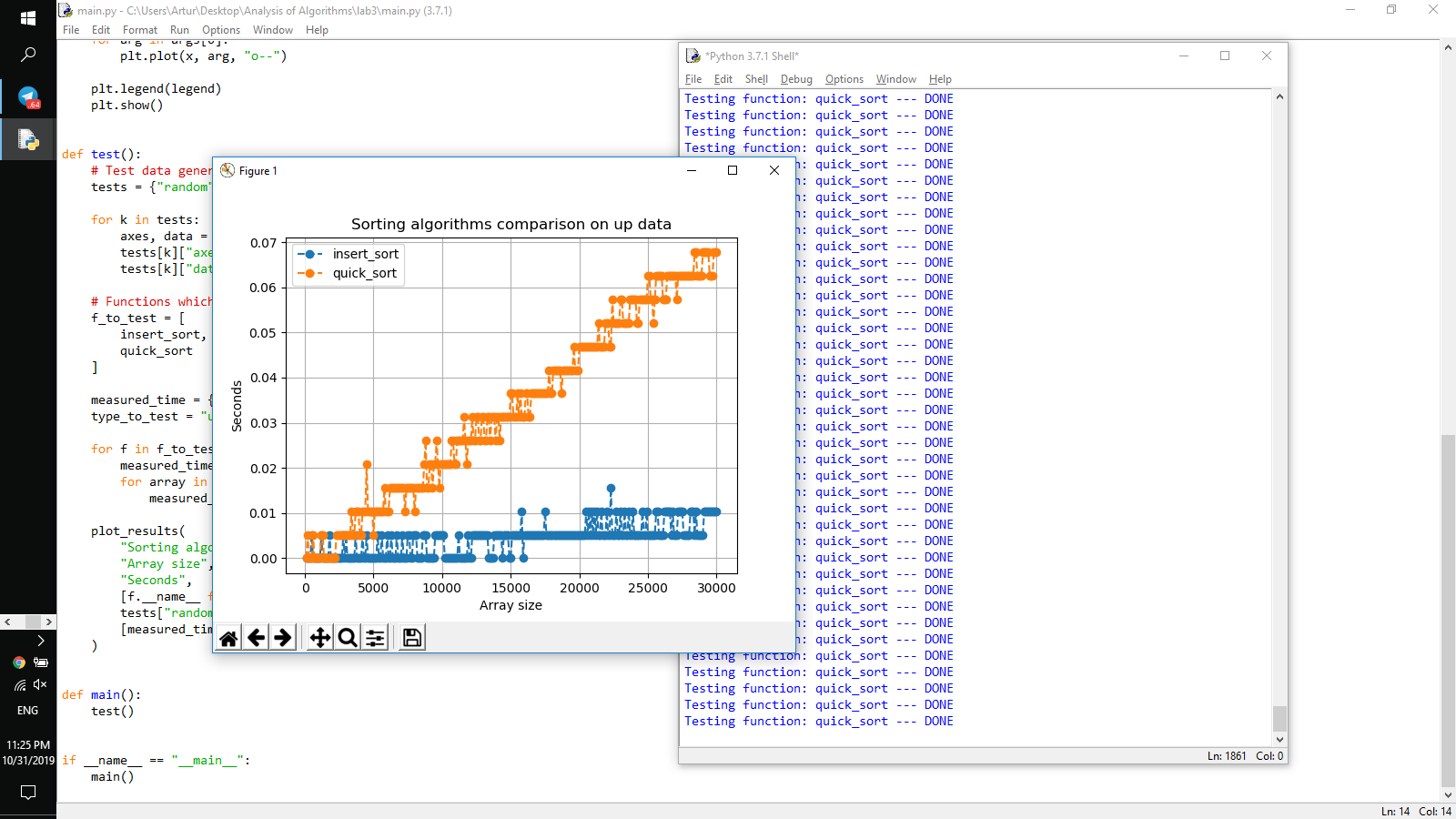


Рисунок 9 - Сравнение времени работы реализаций алгоритмов быстрой сортировки и сортировки вставками на сортированных данных

Опыт показал, что самым быстрым является алгоритм быстрой сортировки – quick sort на частично отсортированных и неотсортированных данных; при отсортированных данных этот алгоритм уступает сортировке вставками. 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.

# Литература

[1] Коварцев, Александр Николаевич (Ред.). Алгоритмы и анализ сложности: учебник / А.Н. Коварцев, А.Н. Даниленко. С. 124

[2] Коварцев, Александр Николаевич (Ред.). Алгоритмы и анализ сложности: учебник / А.Н. Коварцев, А.Н. Даниленко. С. 125-126