**КОНТРОЛЬНОЕ ДОМАШНЕЕ ЗАДАНИЕ**

**ПО КУРСУ**

**ПОСТРОЕНИЕ И АНАЛИЗ АЛГОРИТМОВ**

**АНАЛИЗ И СРАВНЕНИЕ**

**РАЗЛИЧНЫХ МЕТОДОВ СОРТИРОВКИ**

Выполнил студент

Группы БПИ202

Шагаров Дмитрий Александрович

Москва 2022

Методика испытаний

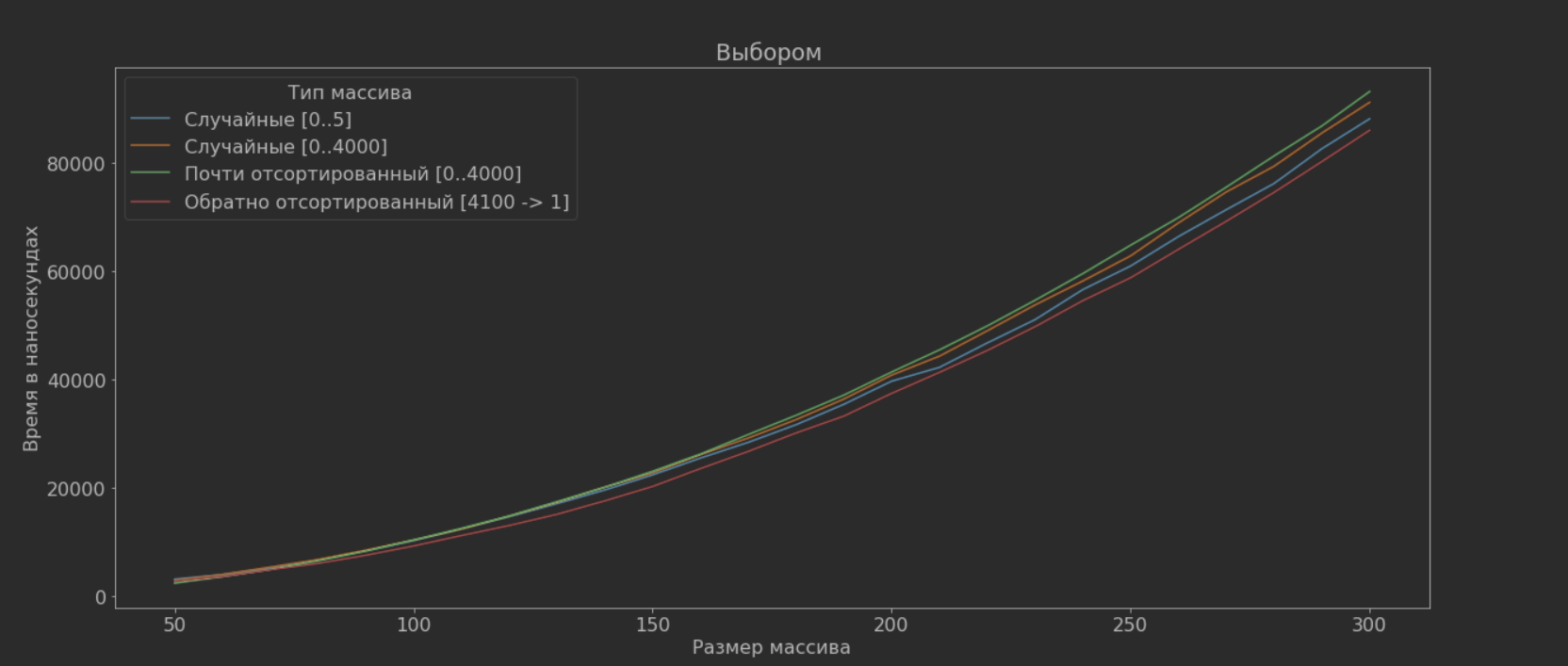
1. Создается набор эталонных массивов 4 различных типов размера 4100, эти массивы заполняются значениями, соответствующими типу.
2. Создается набор из 12 указателей на методы сортировок.
3. Происходят 10 предварительных холостых прогонов, когда каждая сортировка сортирует копию каждого типа массивов.
4. Эталонные массивы перезаполняются новыми значениями, соответствующими типу.
5. Происходит 10 холостых прогонов, полностью соответствующих прогонам, результат которых будет сохранен. В данных 10 прогонах результат не сохраняется в файле.
6. Эталонные массивы перезаполняются новыми значениями, соответствующими типу.
7. Происходит 1000 основных прогонов, результат которых сохраняется в .csv файле. Прогон происходит так:
   1. Идем в первый раз по-маленькому, потом по большому диапазону. Действия ниже аналогичны для каждого диапазона.
   2. Идем по всем методам сортировок.
   3. Идем по всем типам массивов.
   4. Копируем соответствующее диапазону число элементов из выбранного в пункте c эталонного массива в подопытный массив.
   5. Засекаем время.
   6. Сортируем подопытный массив сортировкой, выбранной в пункте b.
   7. Вычисляем затраченное на сортировку время. Прибавляем затраченное время к результатам.
   8. В случае, если массив не отсортирован, кидаем исключение.
8. В результатах делим все значения на число прогонов.
9. Записываем результаты в файл .csv.

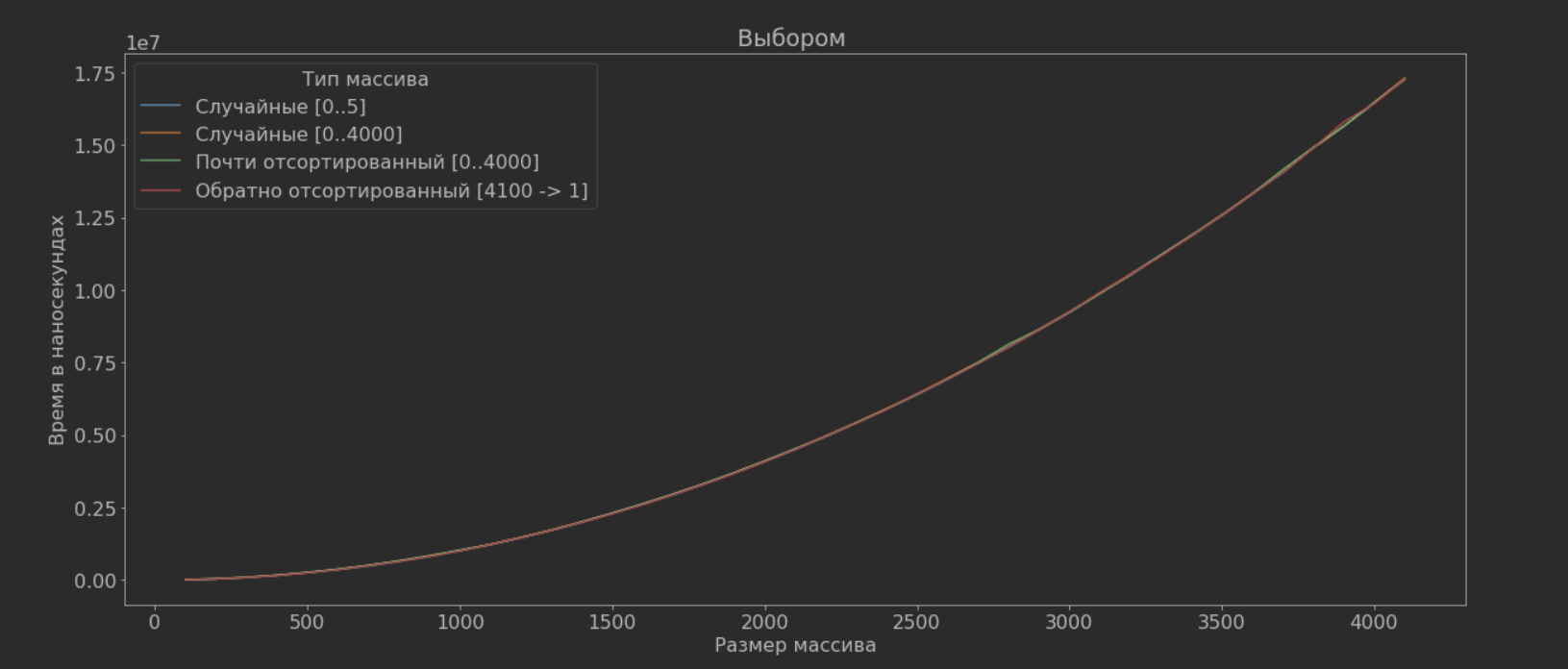
Анализ результатов

Все построения графиков выполнены с использованием Python + pandas + matplotlib + seaborn.

*\*\*Далее n-число сортируемых элементов.*

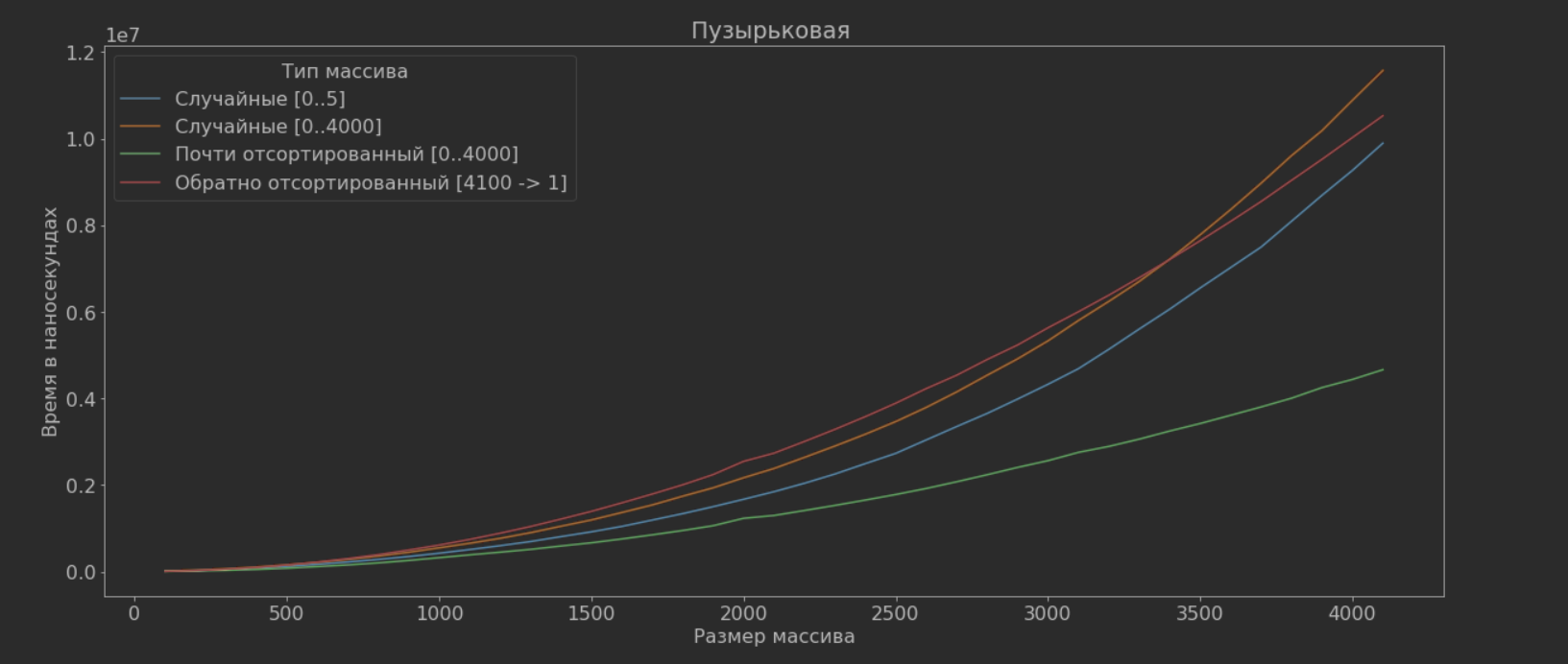
# Сортировка на разных типах массивов.

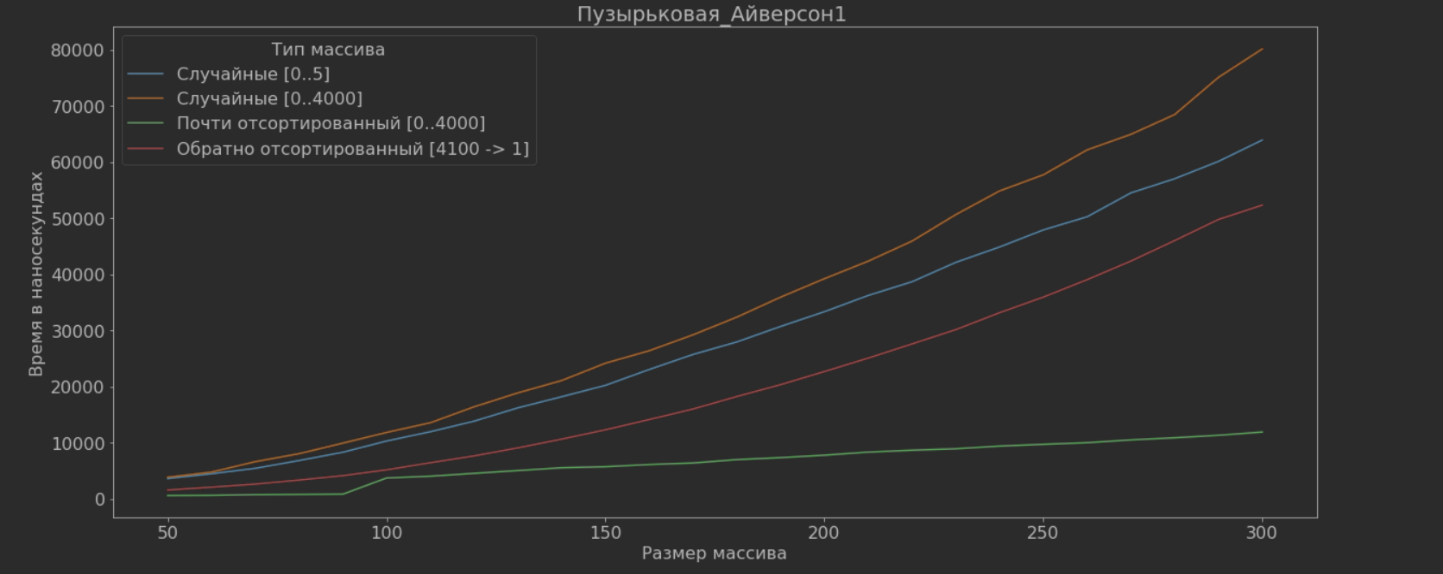


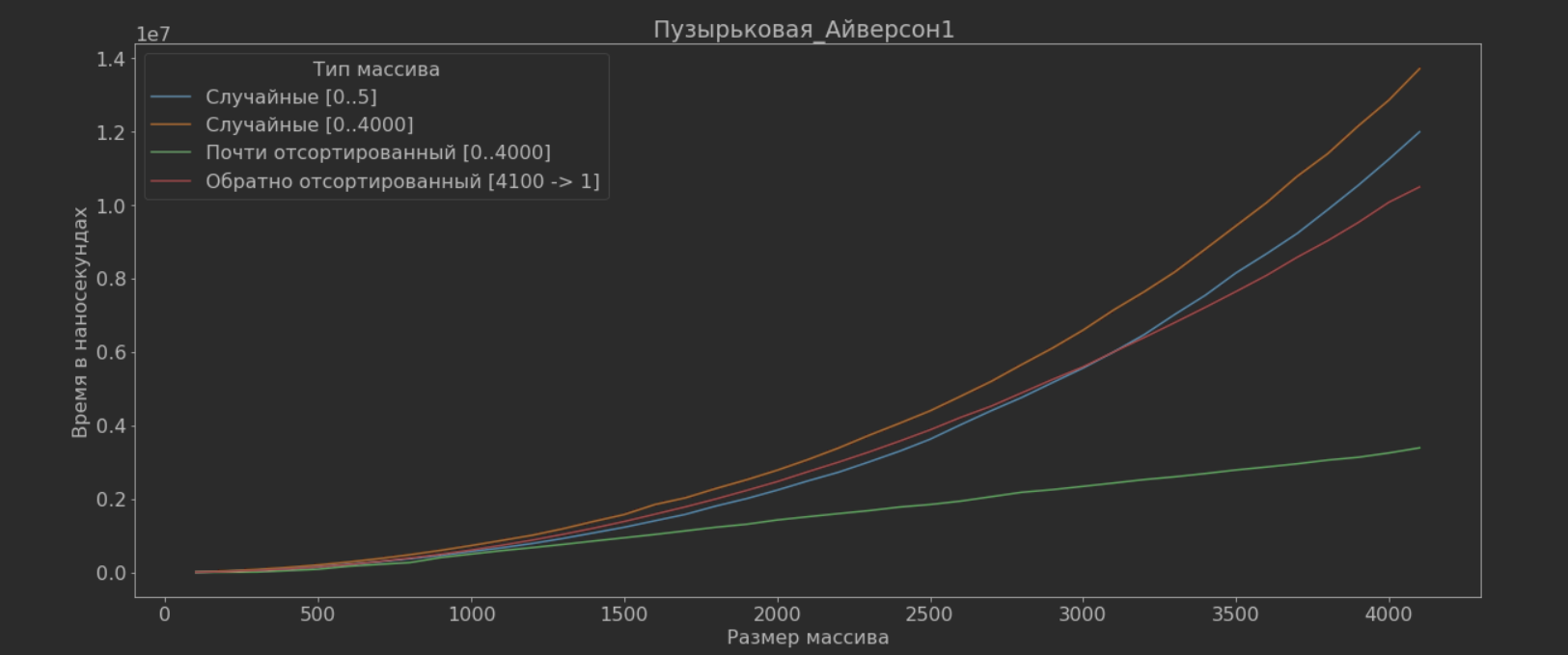


Сортировка выбором работает всегда одинаково за O(n^2). Так же стабильна, как Швейцарский франк, ~~так же слаба, как Российский рубль~~.

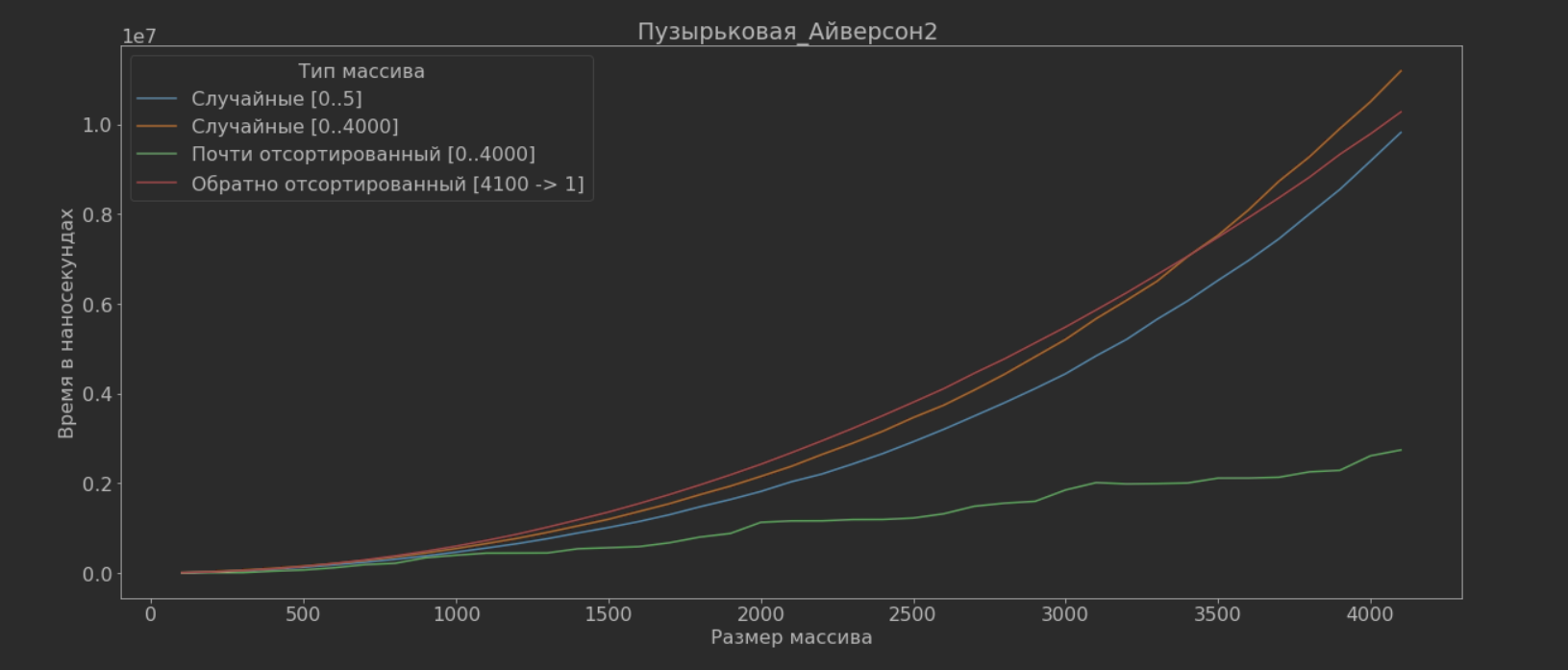
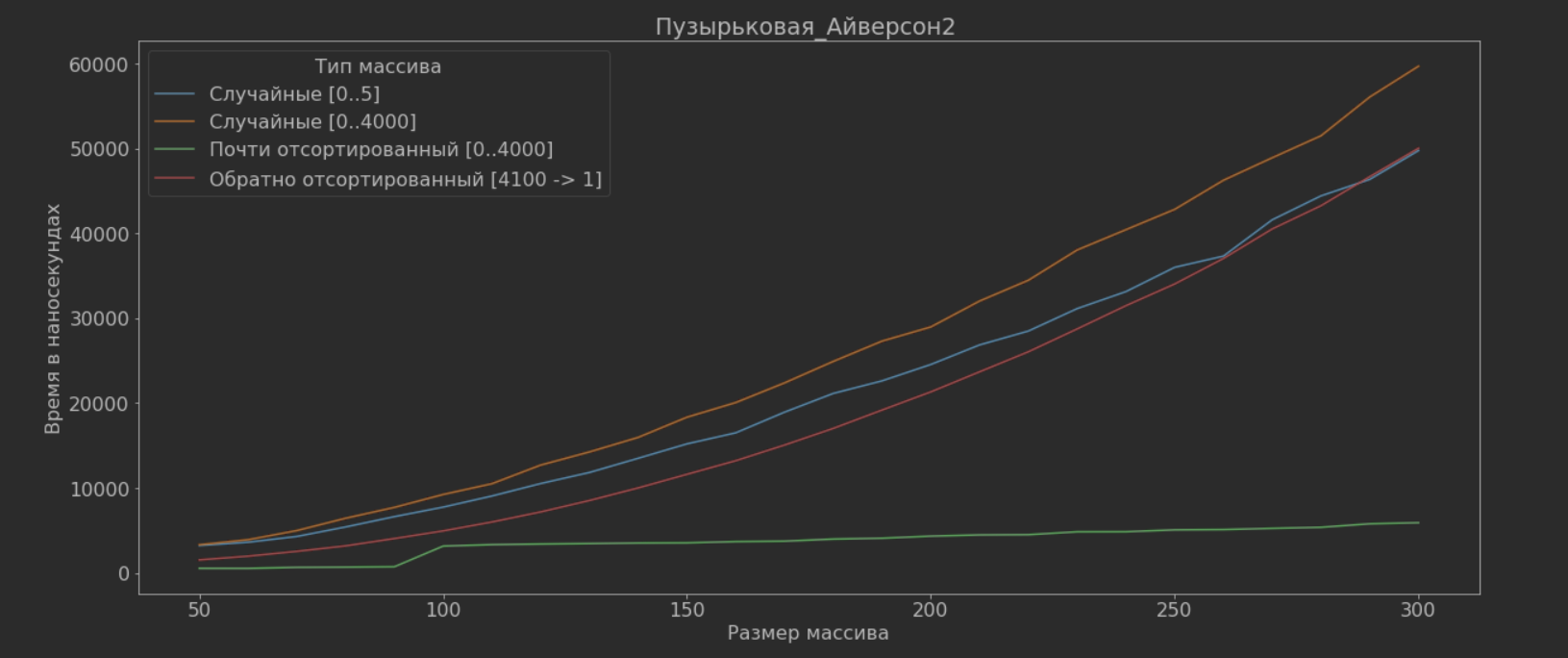


Сортировка пузырьком работает в случае почти отсортированного массива быстрее, нежели в других случаях. Медленнее всего – на большом диапазоне случайных чисел (при большом числе элементов, плохо работает так же обратно отсортированный массив). Видно, что она незначительно быстрее сортировки выбором в случаях, кроме почти отсортированного (тоже O(n^2), но, вероятно, с меньшей константой). Но в случае почти отсортированного, сложность сортировки уже O(n).

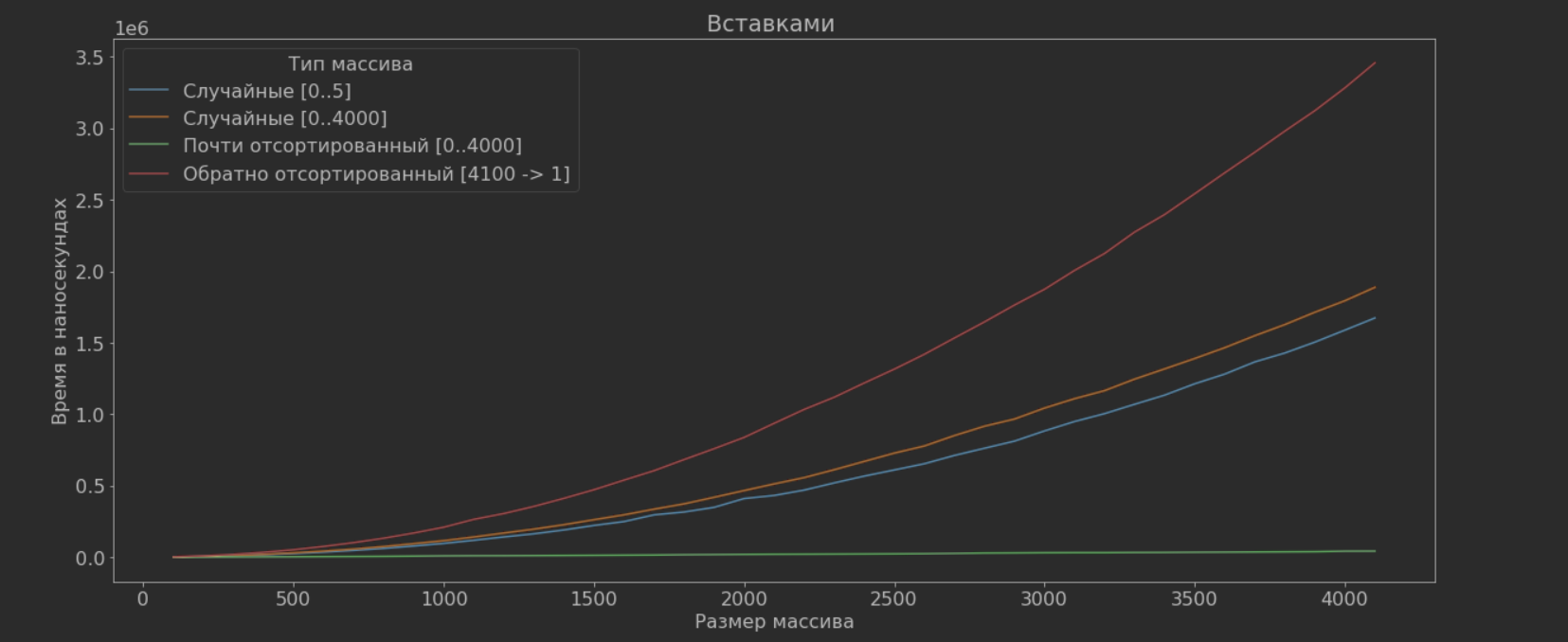
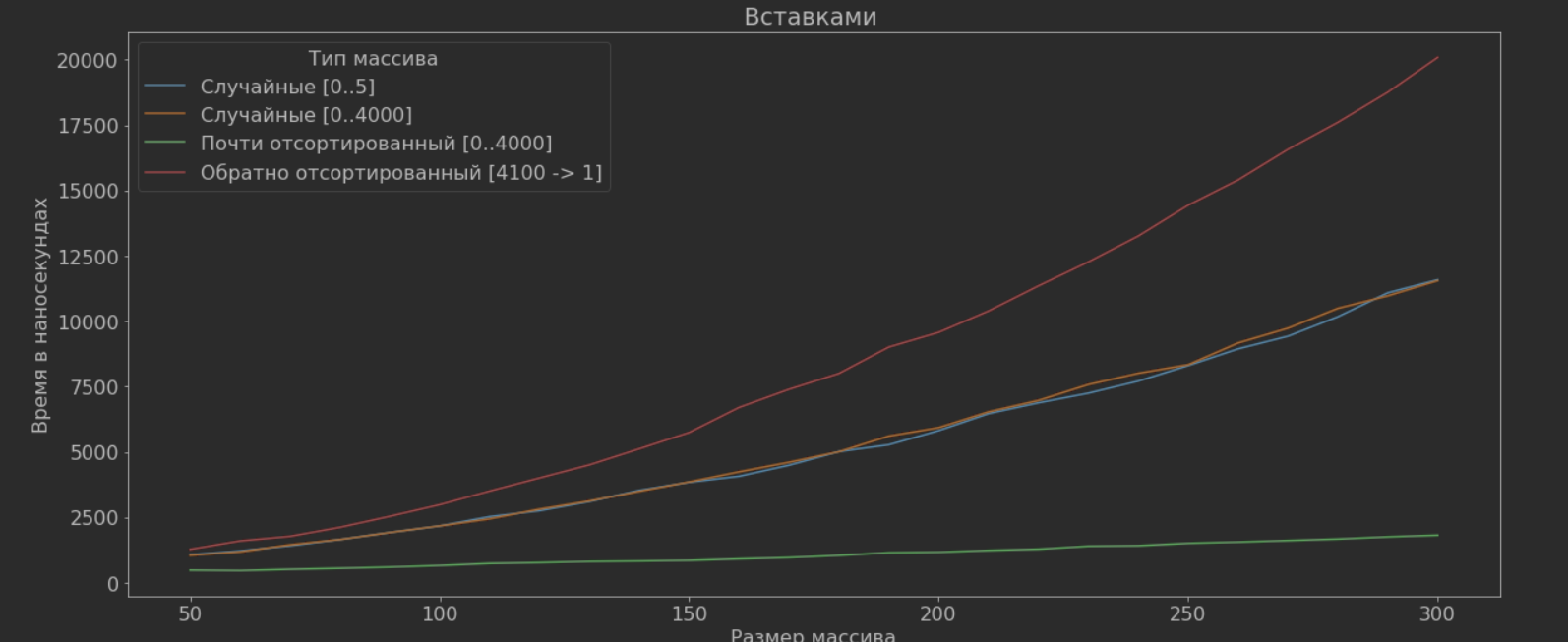




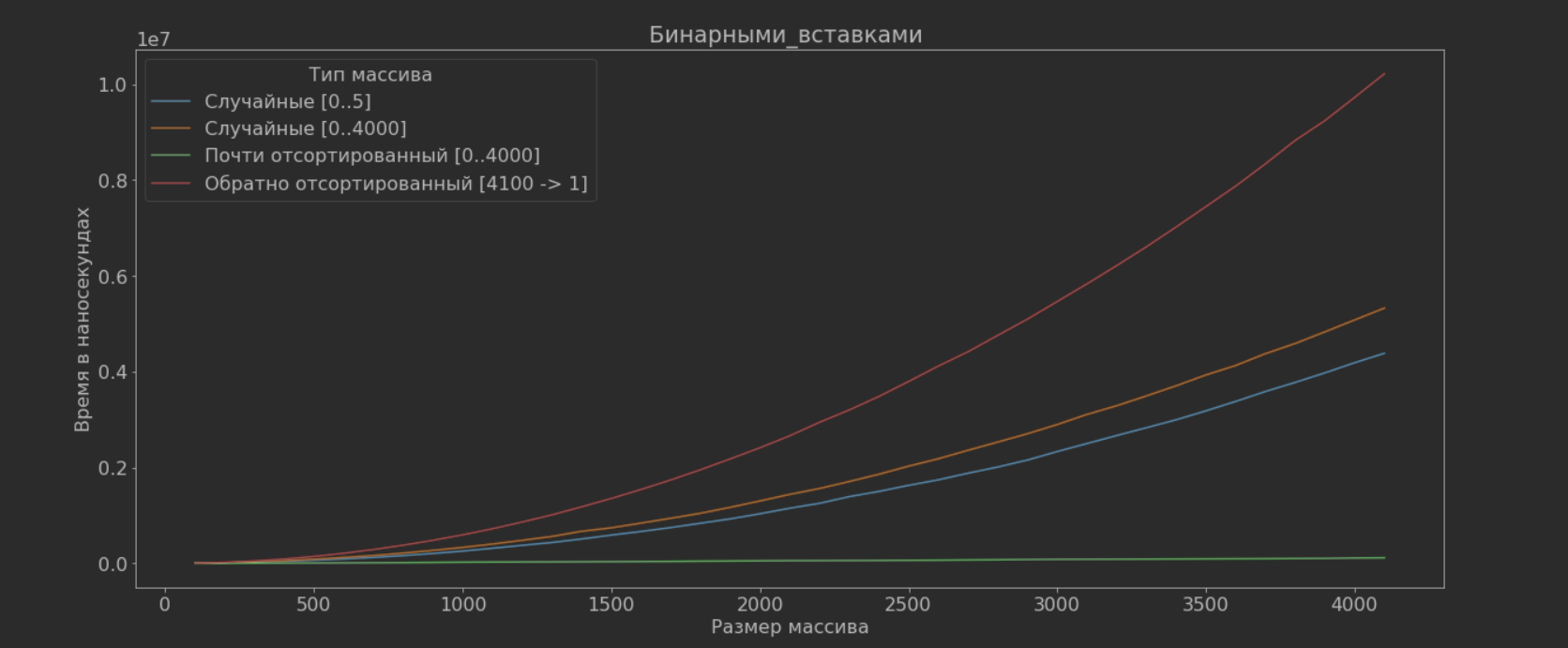
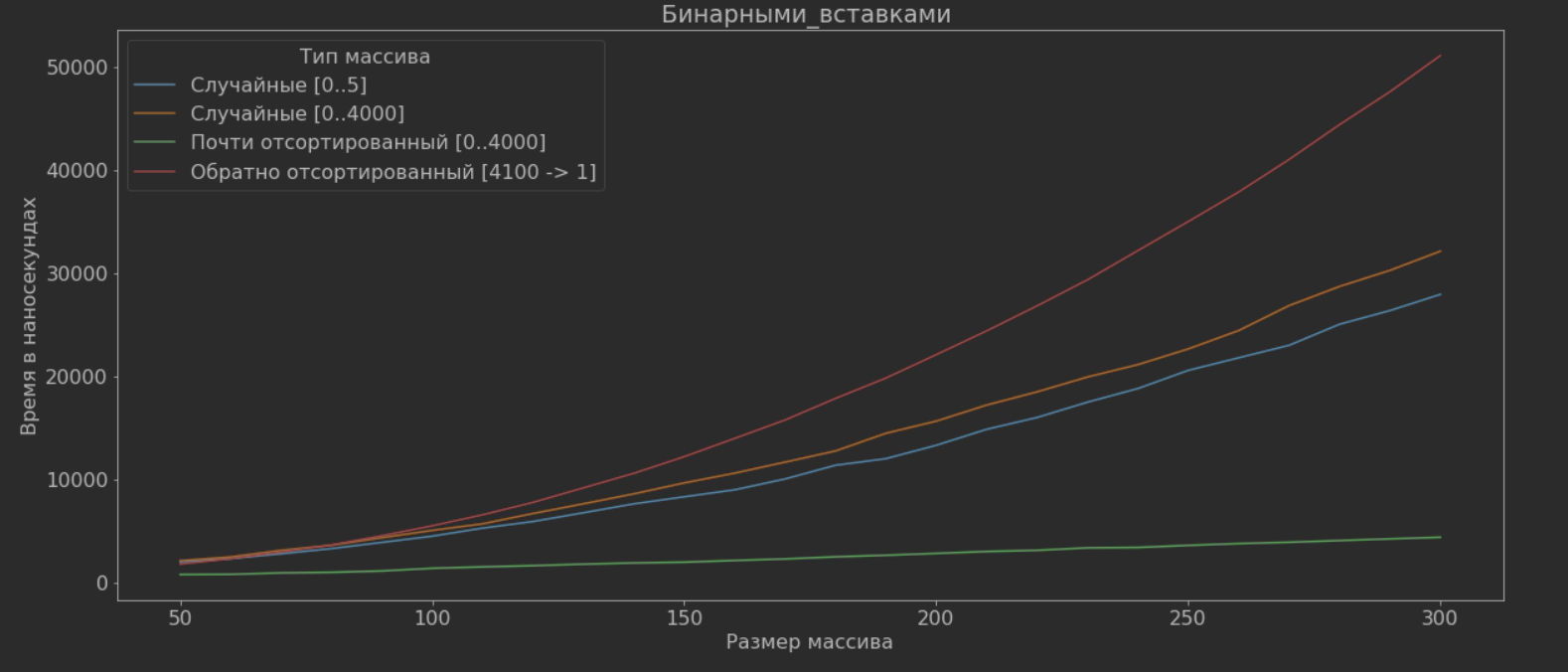
Как можно заметить, условие Айверсона 1 оптимизирует работу сортировки на почти отсортированных данных.



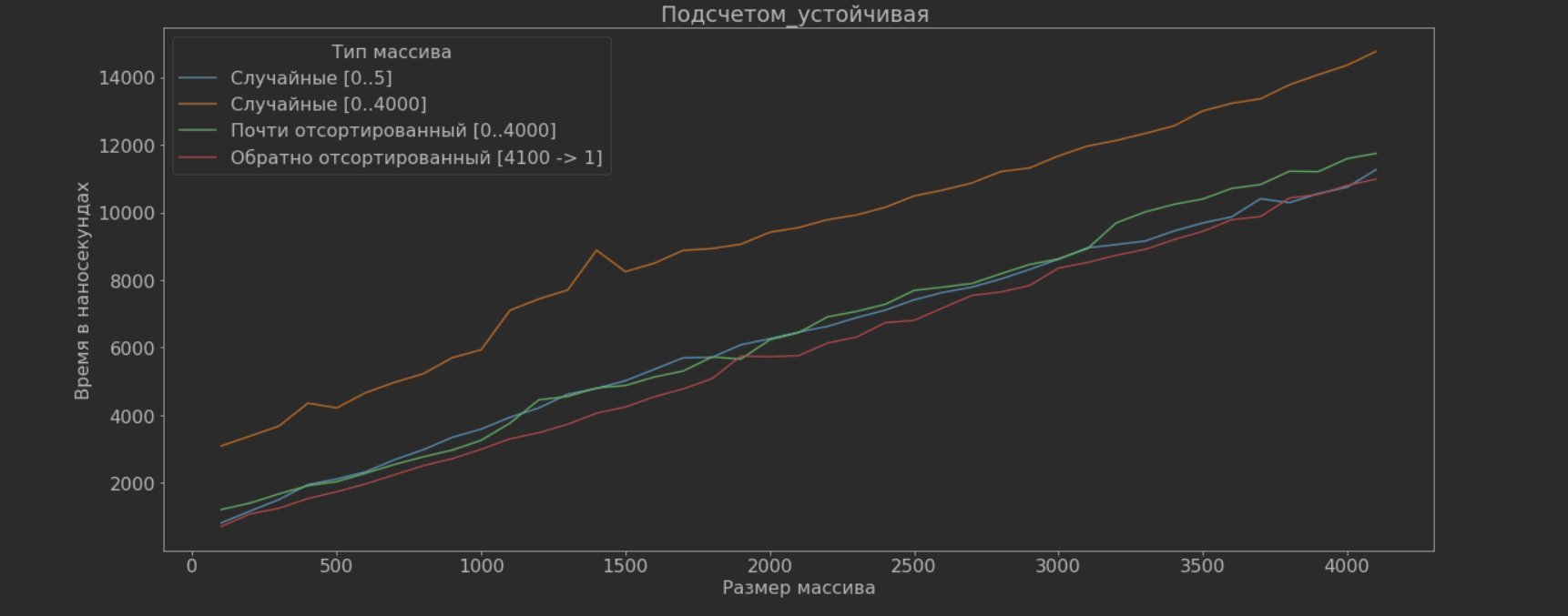
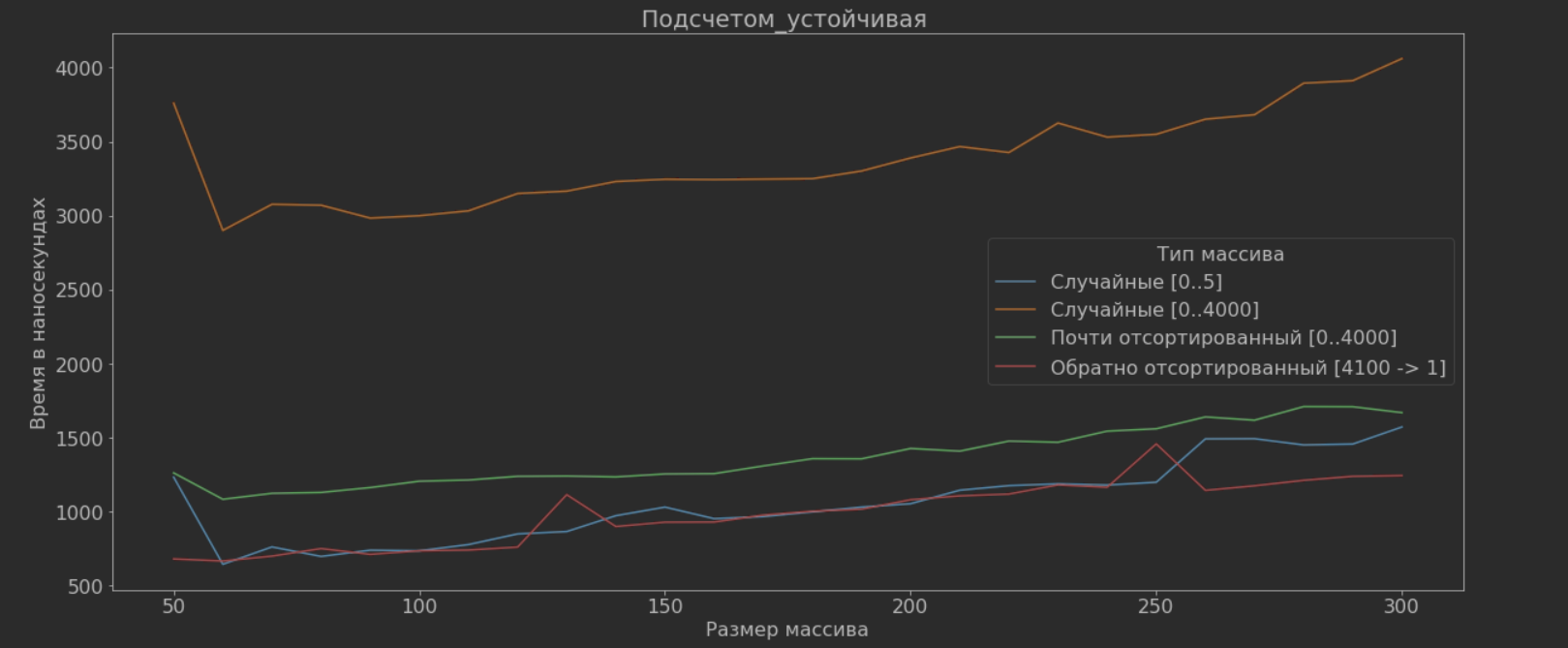
А условие Айверсона 2 еще сильнее оптимизировало случай с почти отсортированными данными.



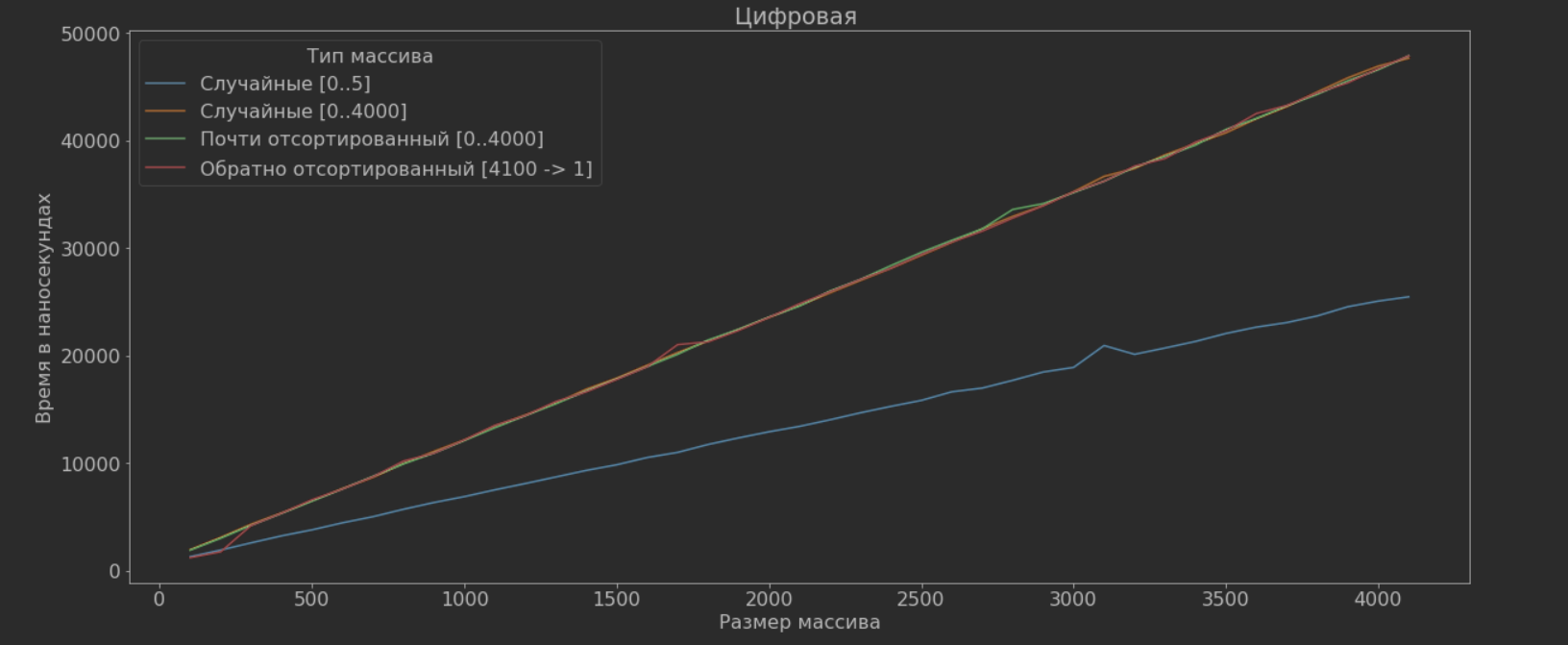
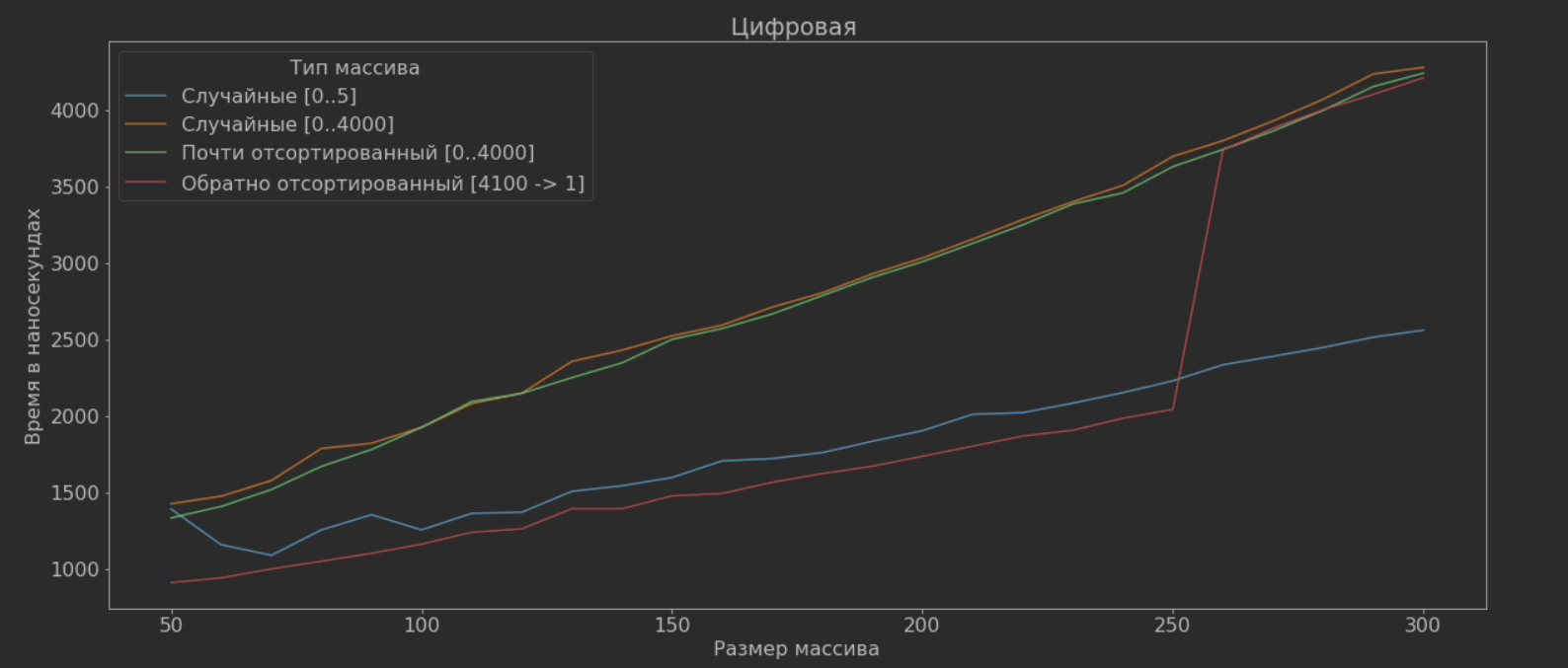
Сортировка вставками прекрасно работает на почти отсотированных данных, однако плохо на обратно отсортированных данных. Опять же имеет сложность O(n^2) в среднем и худшем случае, но при почти отсортированных данных сложность падает до O(n). Судя по времени, константы тут еще меншье, чем у пузырьковой сортировки.



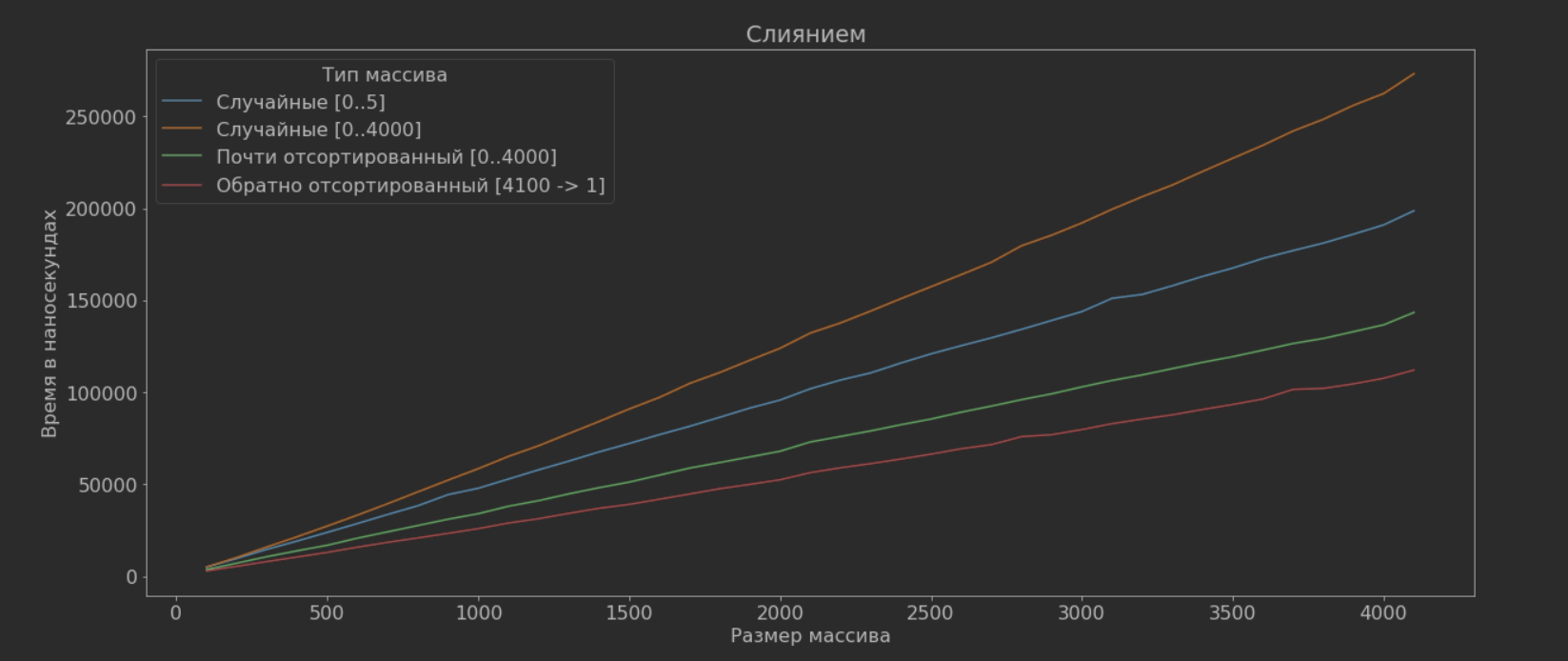
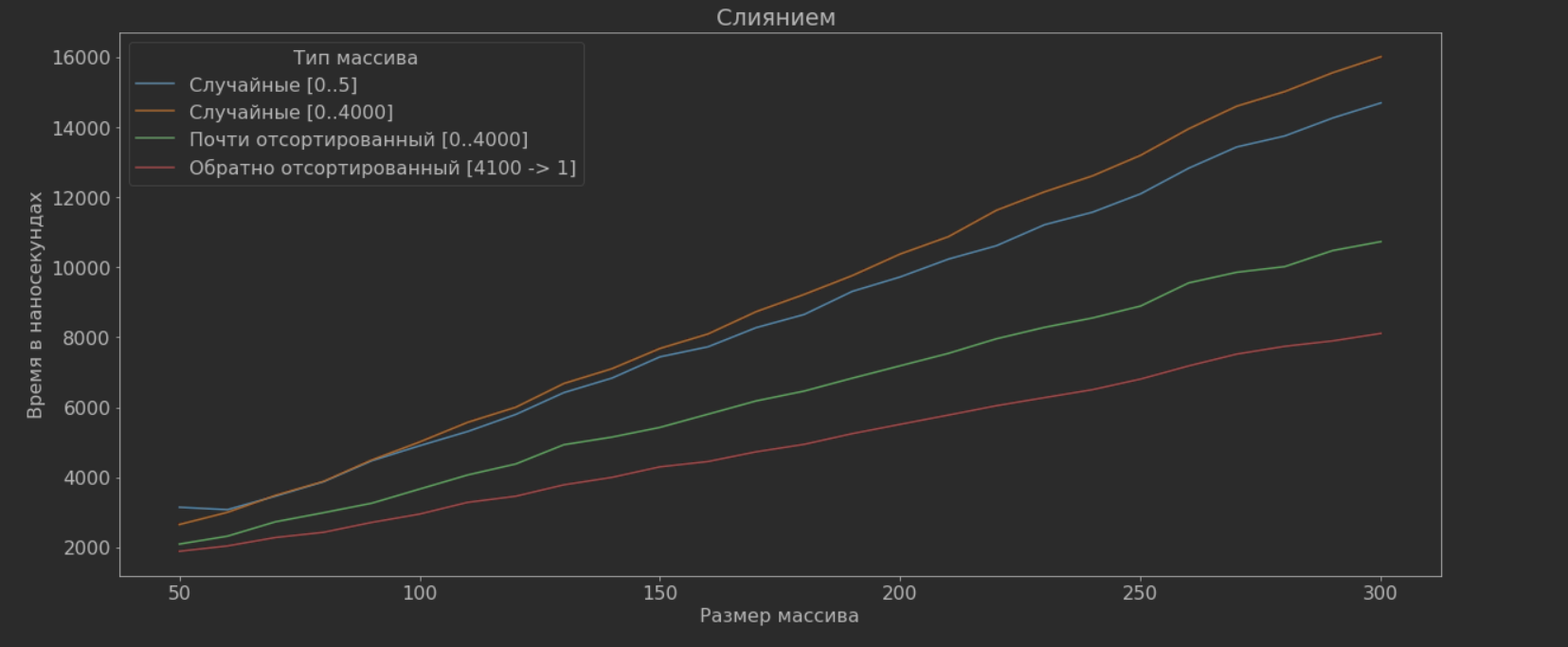
Так получилось, что вместо выигрыша на сгенерированных данных сортировка бинарными вставками показала результат худший, нежели обычными вставками. Видимо бинарный поиск долго искал необходимый элемент.



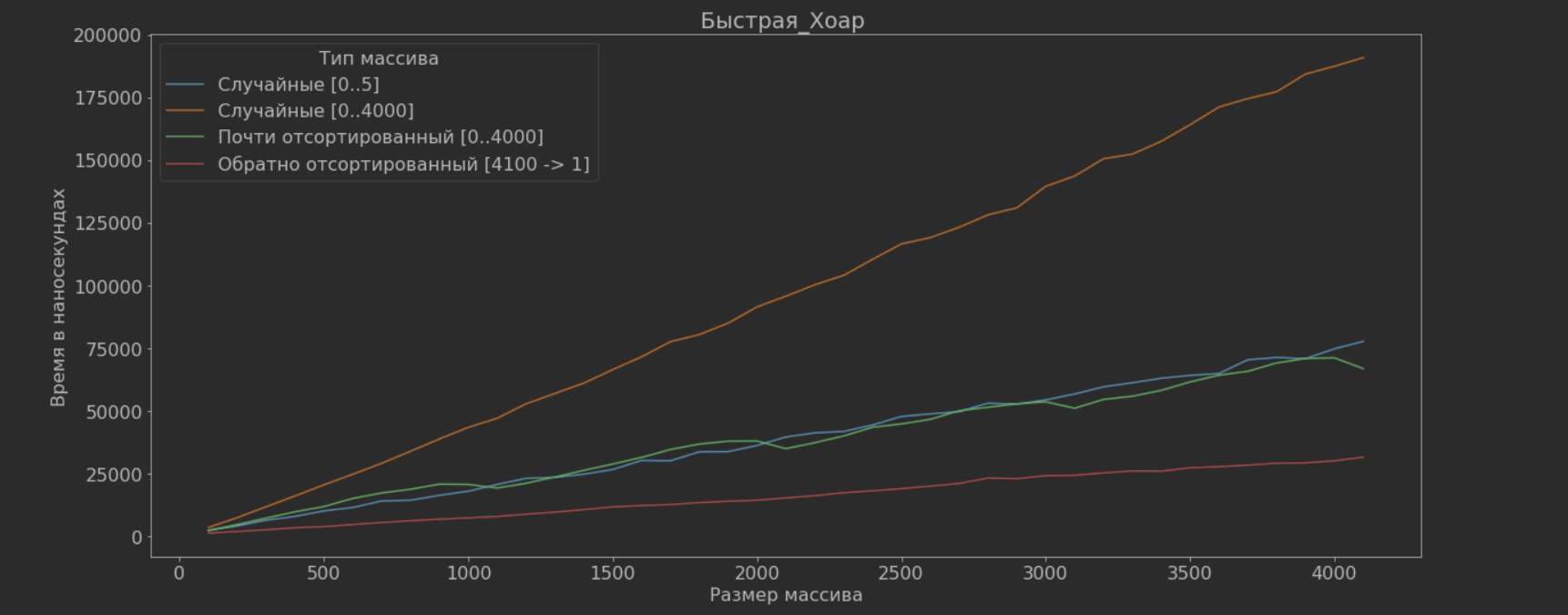
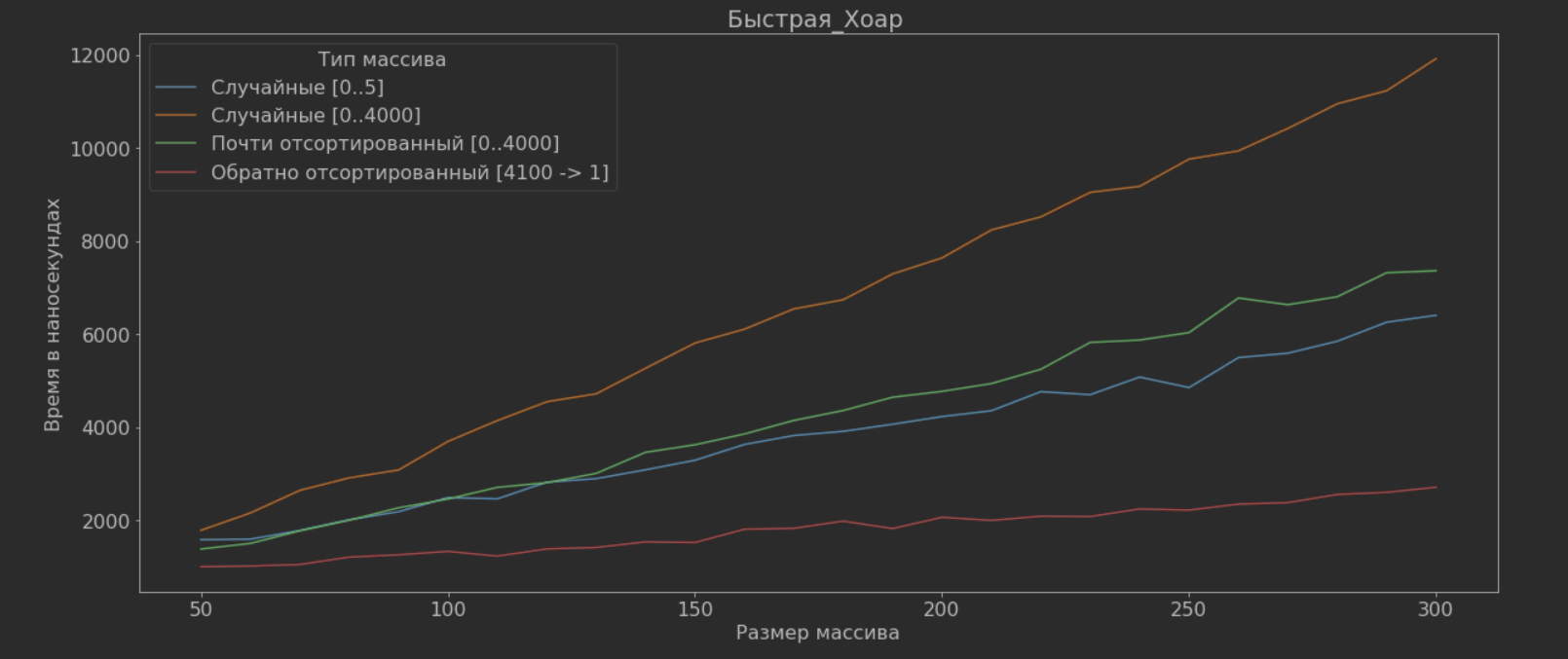
Сортировка подсчетом работает отлично, пока диапазон чисел маленький. Видно, что при увеличении диапазона с 5 до 4000, время работы увеличилось приблизительно в 3 раза. Это при том, что диапазон 4000 — это сравнительно маленький диапазон. Все потому, что сложность алгоритма O (n + k), где k – разница между минимальным и максимальным элементами массива. При небольшом числе элементов массива от 50 до 300, сложность по началу вообще была квадратичная, относительно n.



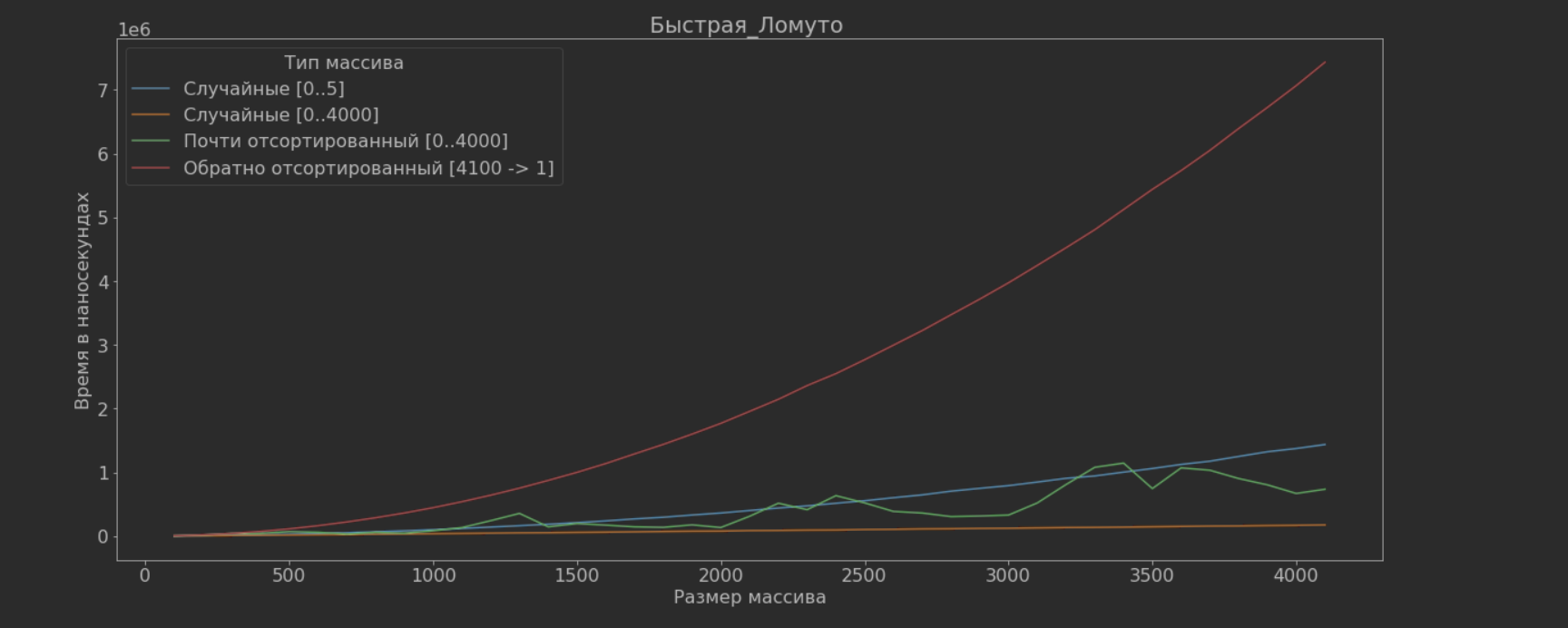
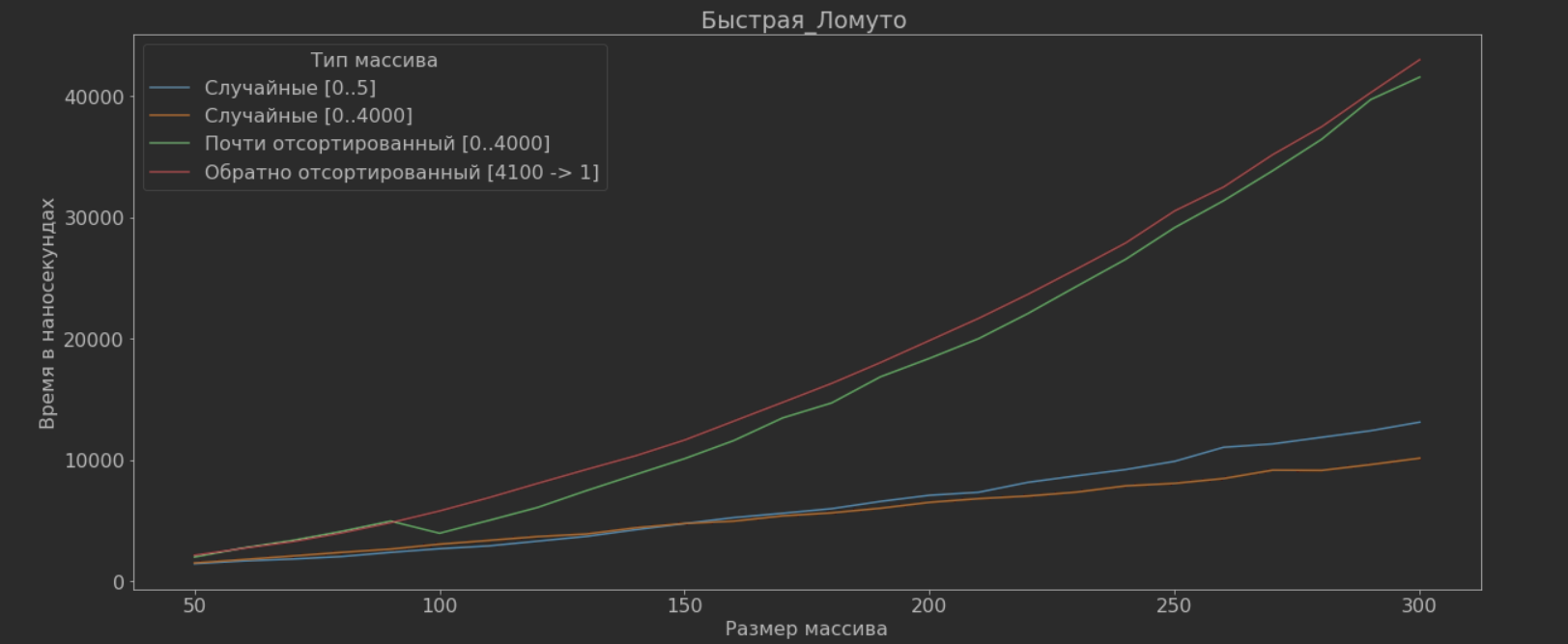
Цифровая сортировка по основанию 256 лучше всего работает на маленьком диапазоне чисел. Сложность сортировки O(m(n + k)), где m- число разрядов сортируемых чисел, k-диапазон занчений цифр сортируемых чисел.



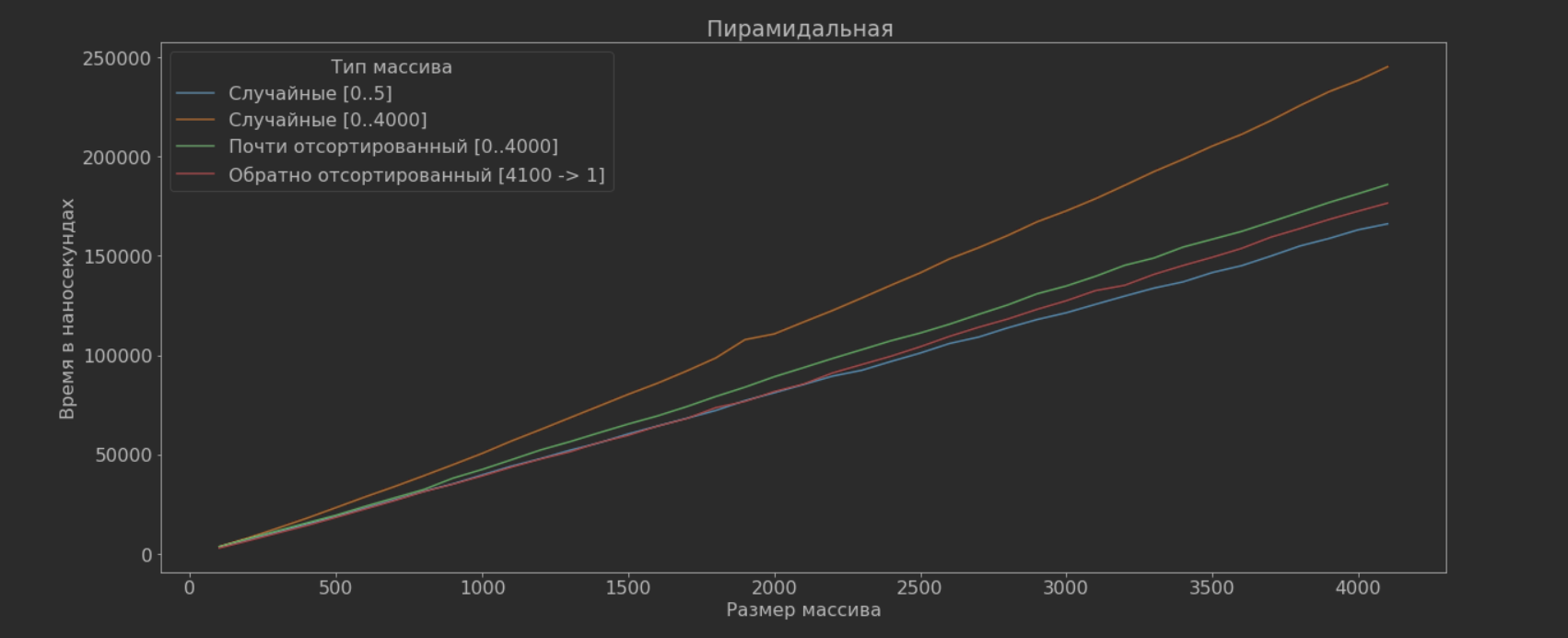
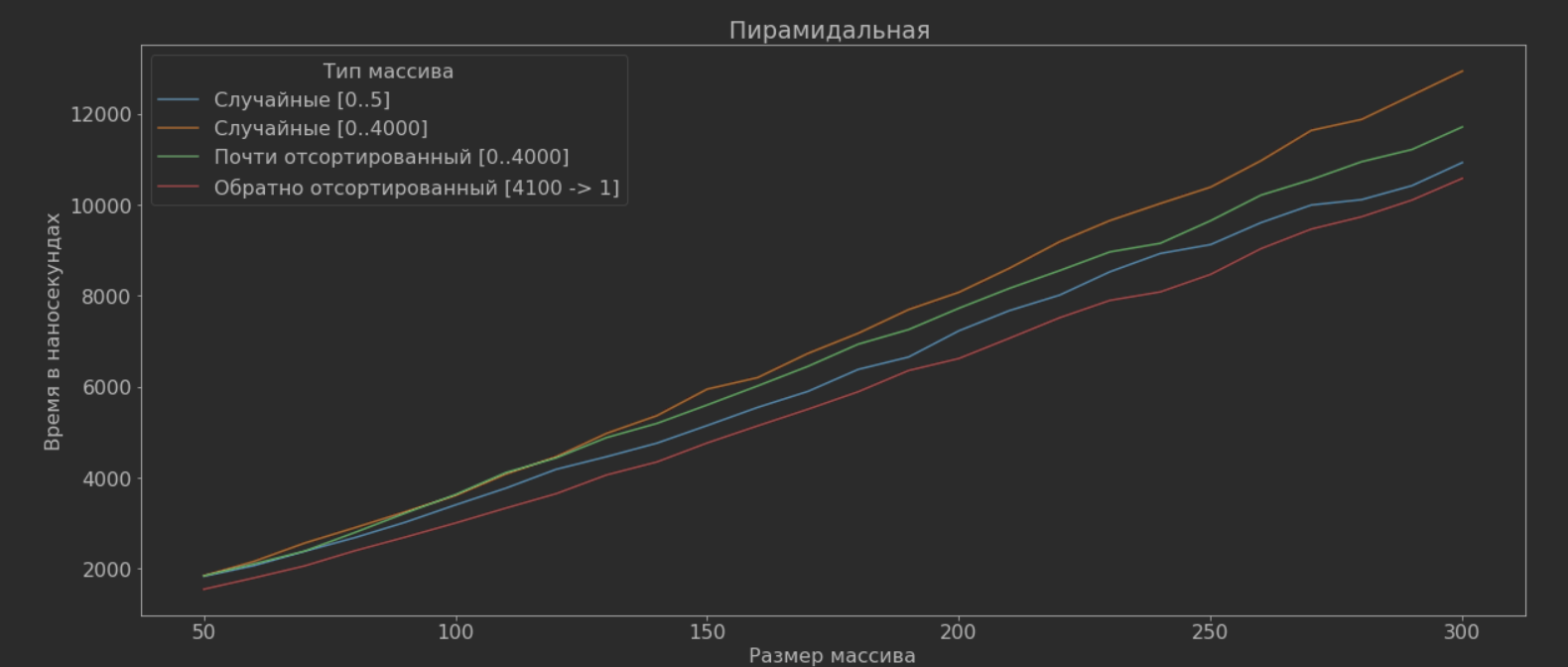
Потрясающая сортировка. Всегда стабильно, спокойно работает за O(nlogn). Чуть хуже, но незначительно, сортирует числа случайные числа.



Быстрая сортировка с разбиением Хоара. Лучший случай, когда массив отсортирован обратно. Это идеально как раз таки, когда разделительный элемент-центральный. В целом, сложность сортировки в лучшем и среднем случае – O(nlogn). Можно подобрать такой худший случай, что сортировка деградирует до O(n^2). Константа при этом меньше, нежели в сортировке слиянием.

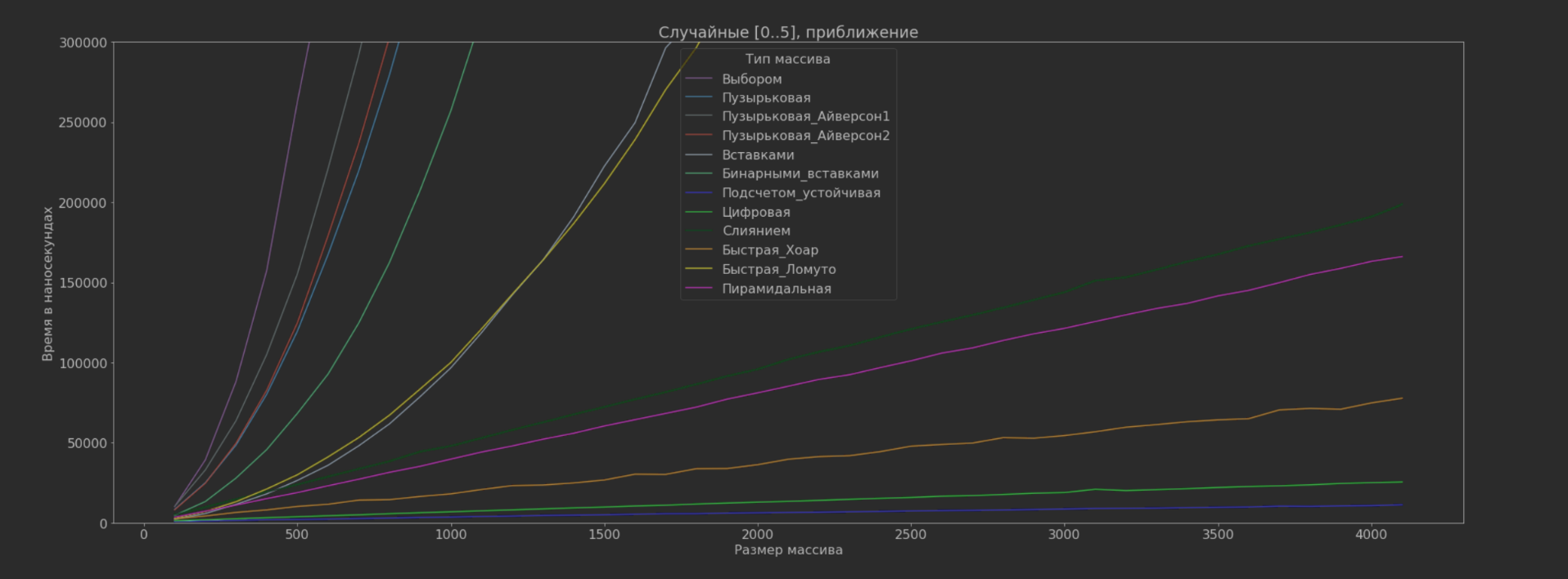
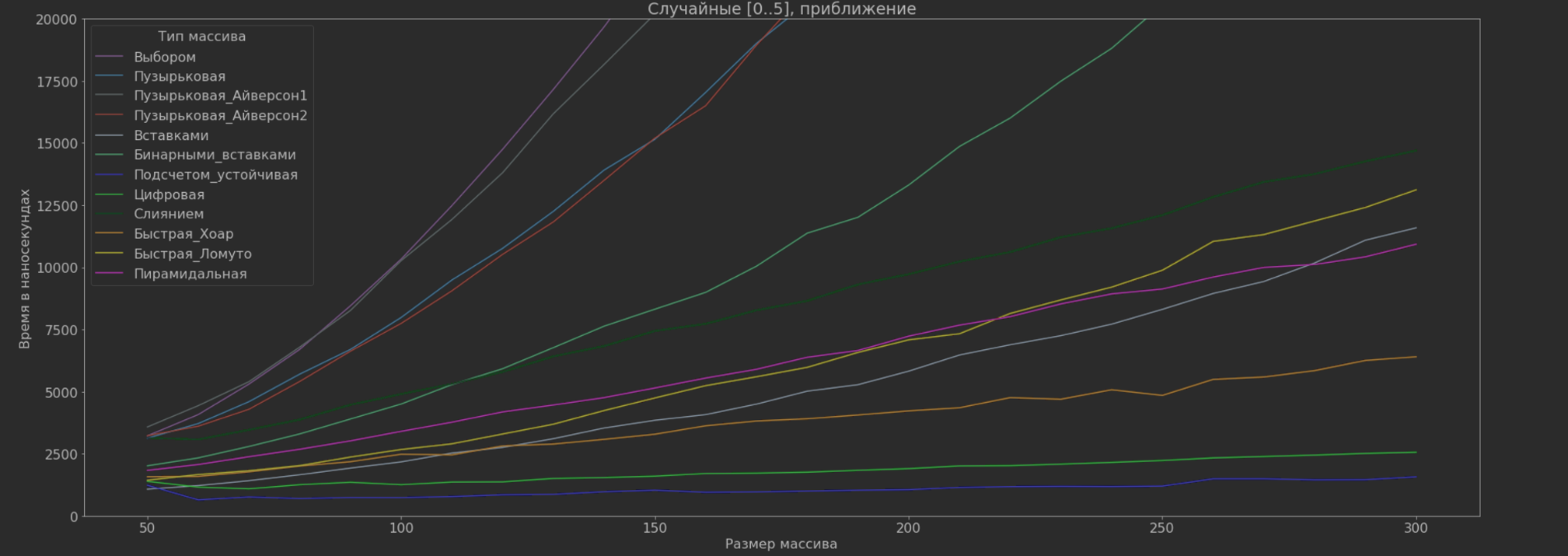
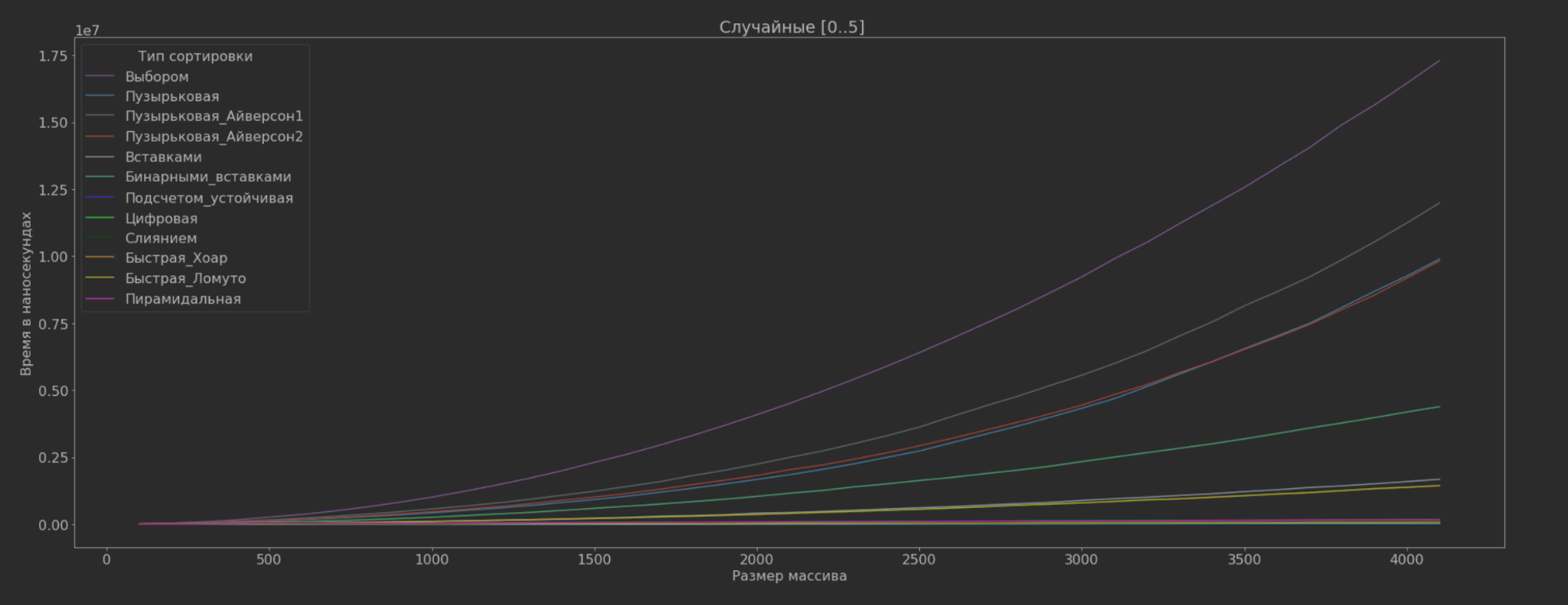
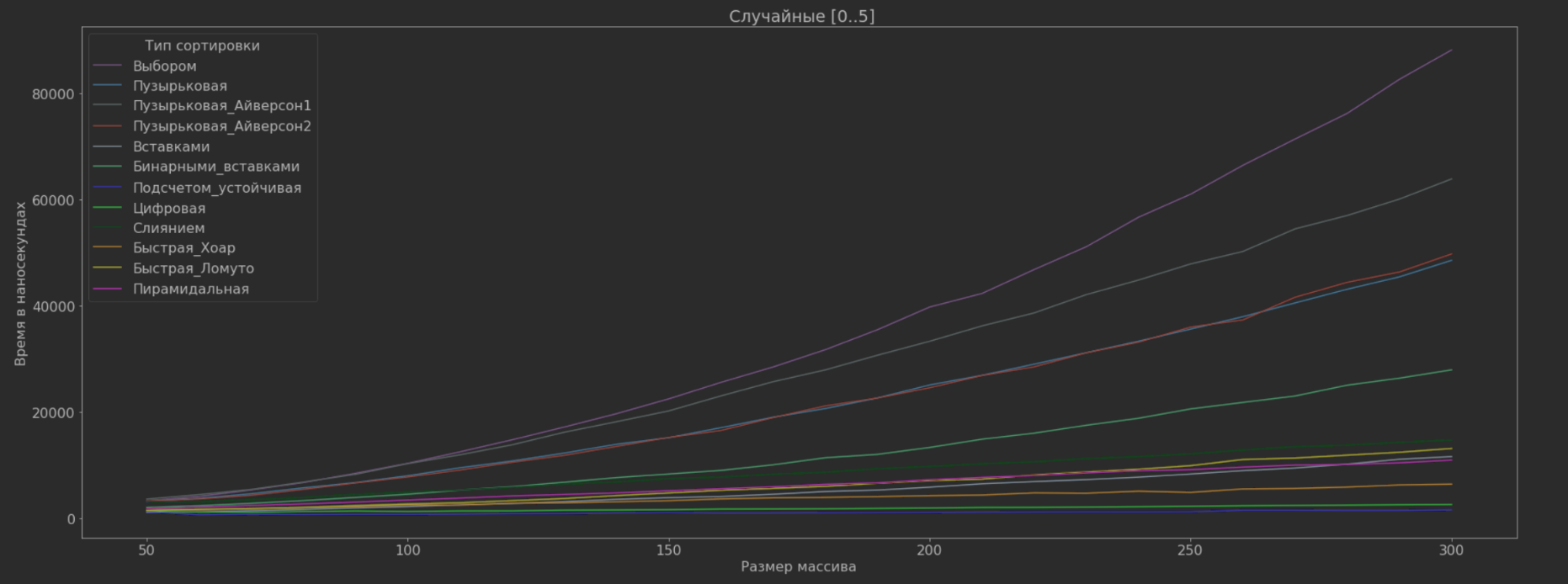


Быстрая сортировка с разбиением Ломуто. В целом, сложность сортировки в лучшем и среднем случае – O(nlogn). Однако виден худший случай, когда сортировка деградирует до O(n^2). Это обратно отсортированный массив. Это происходит из-за того, что в качестве опорного выбирается последний элемент сортируемой части. На этих данных сортировка работает в целом медленнее, чем при разбиении Хоара (есть подозрение, что так бывает чаще всего).

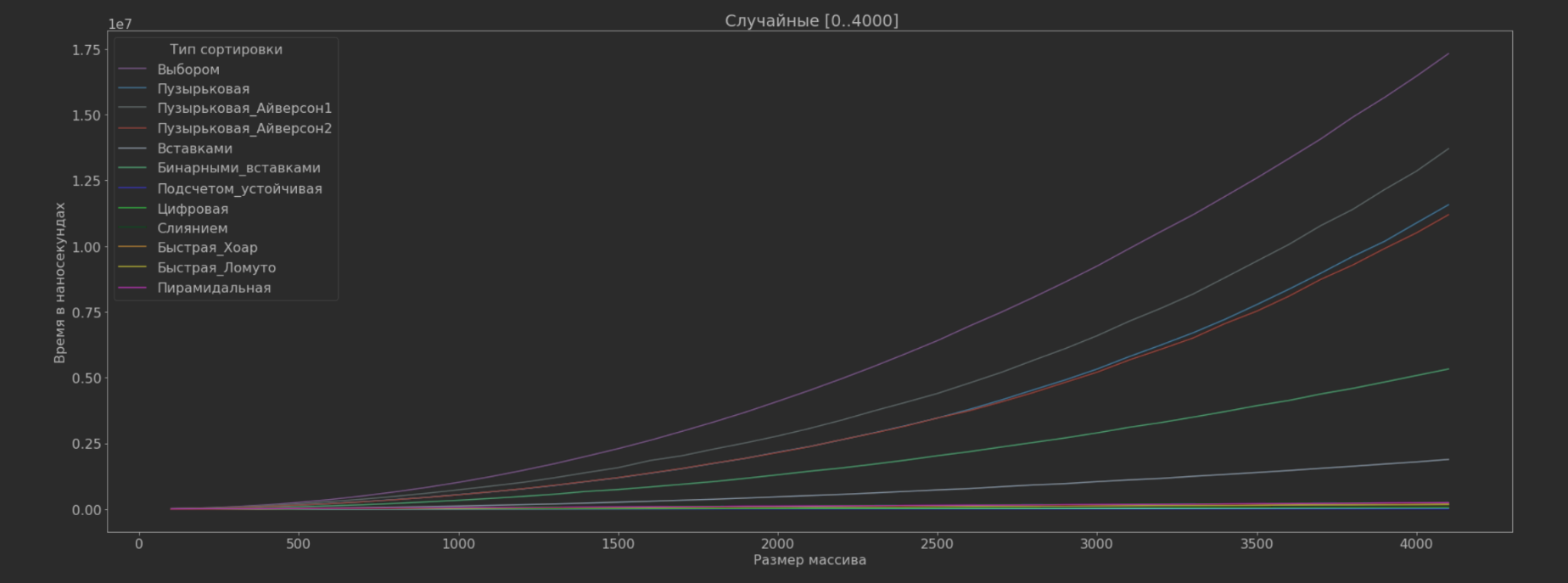
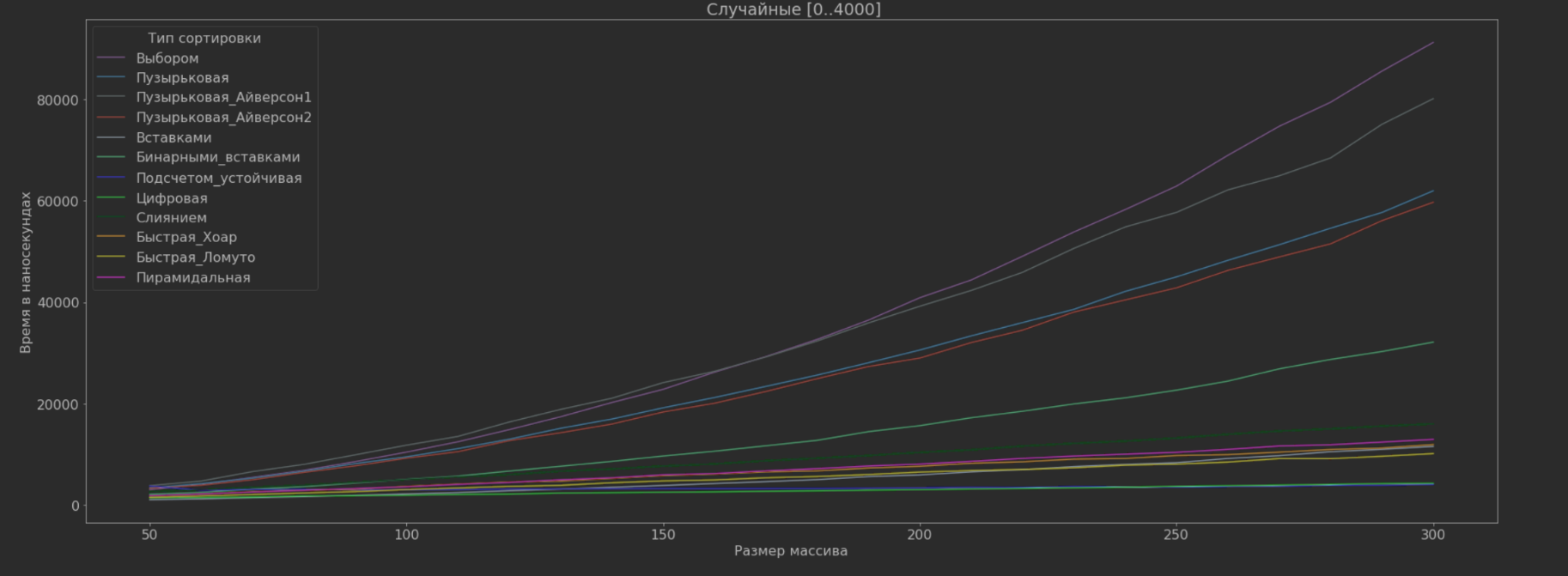


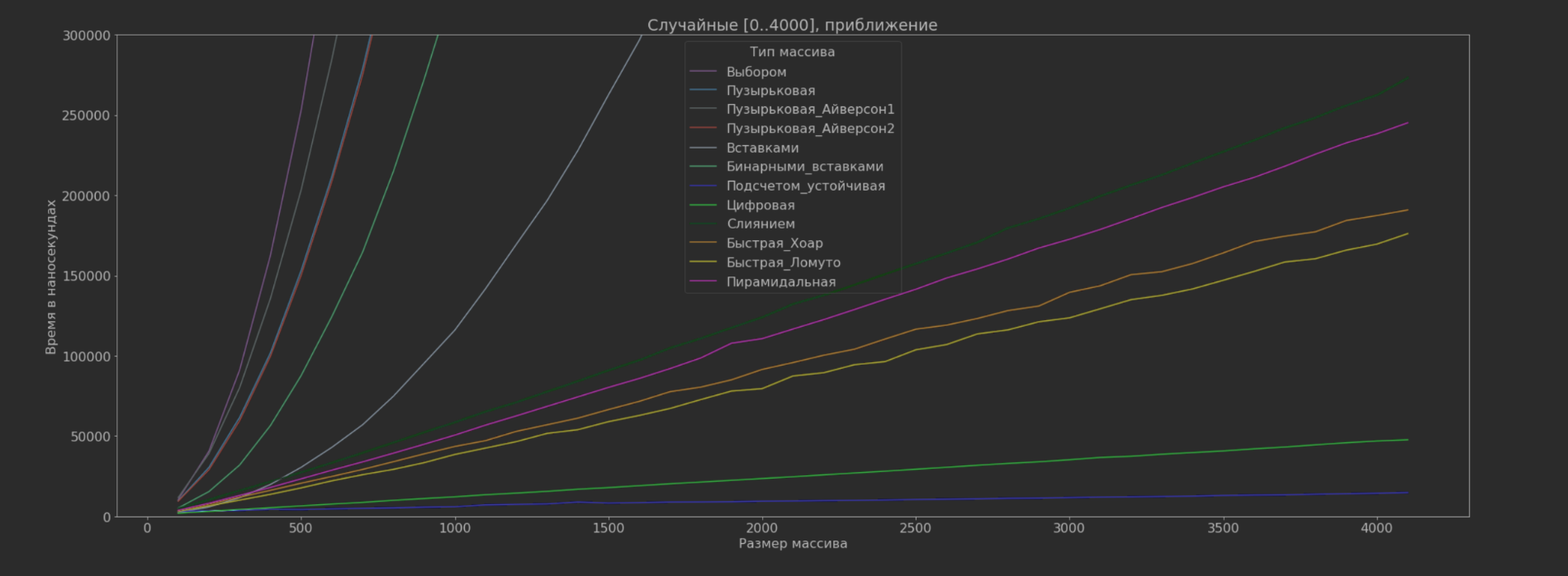
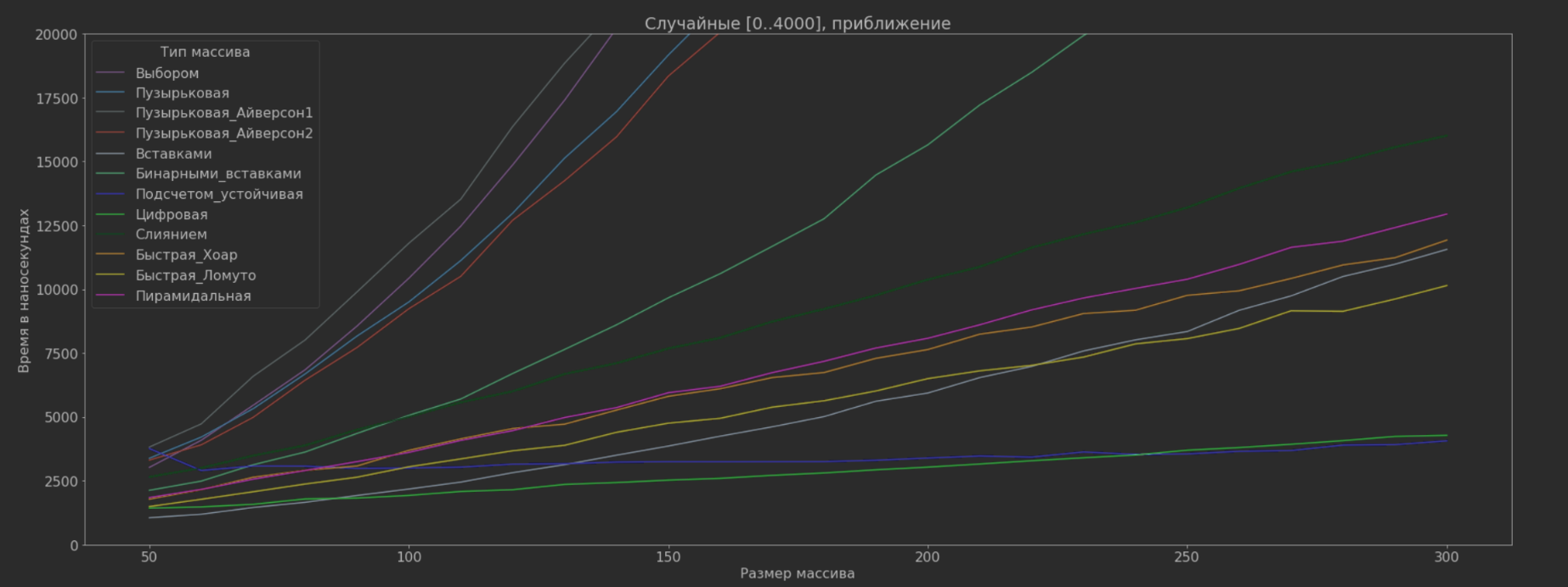
Как и сортировка слияением, прекрасная сортировка. Всегда O(nlogn). Даже точное время сильно похоже на сортировку слиянием. Единственное, меньший разброс при различных типах массивов.

# Тип массива на разных сортировках.

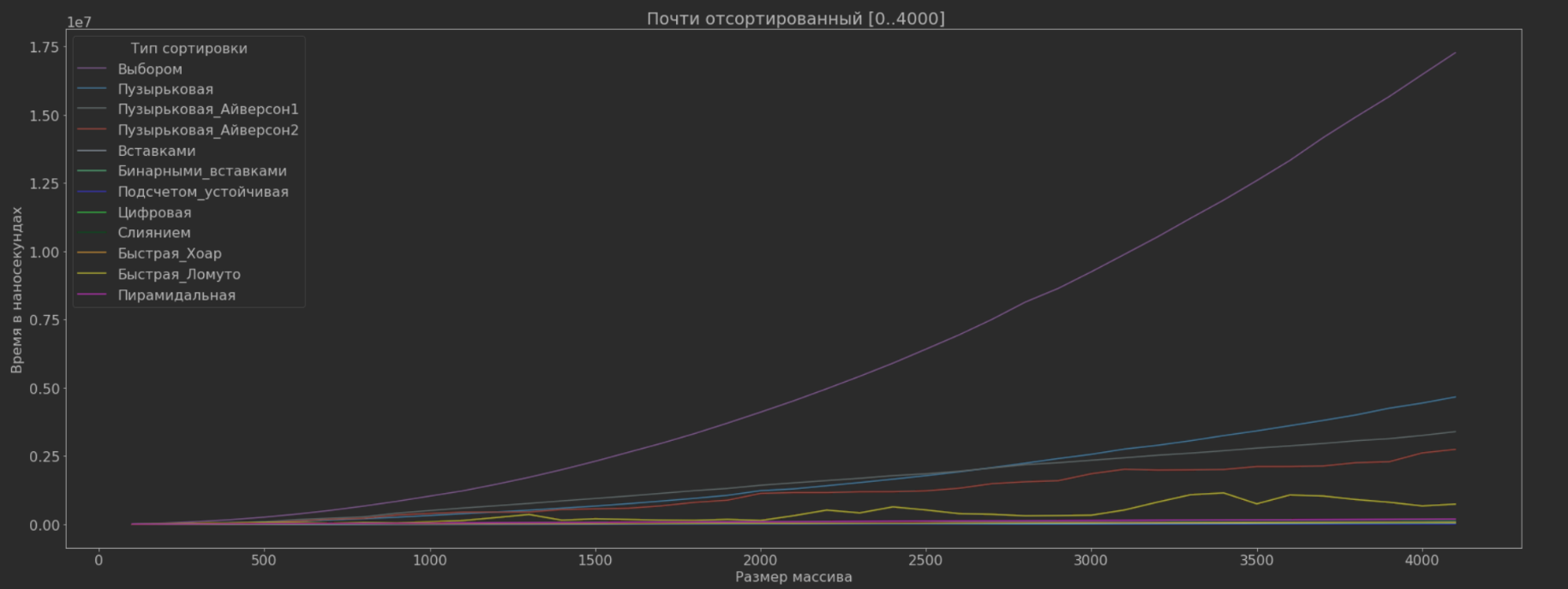
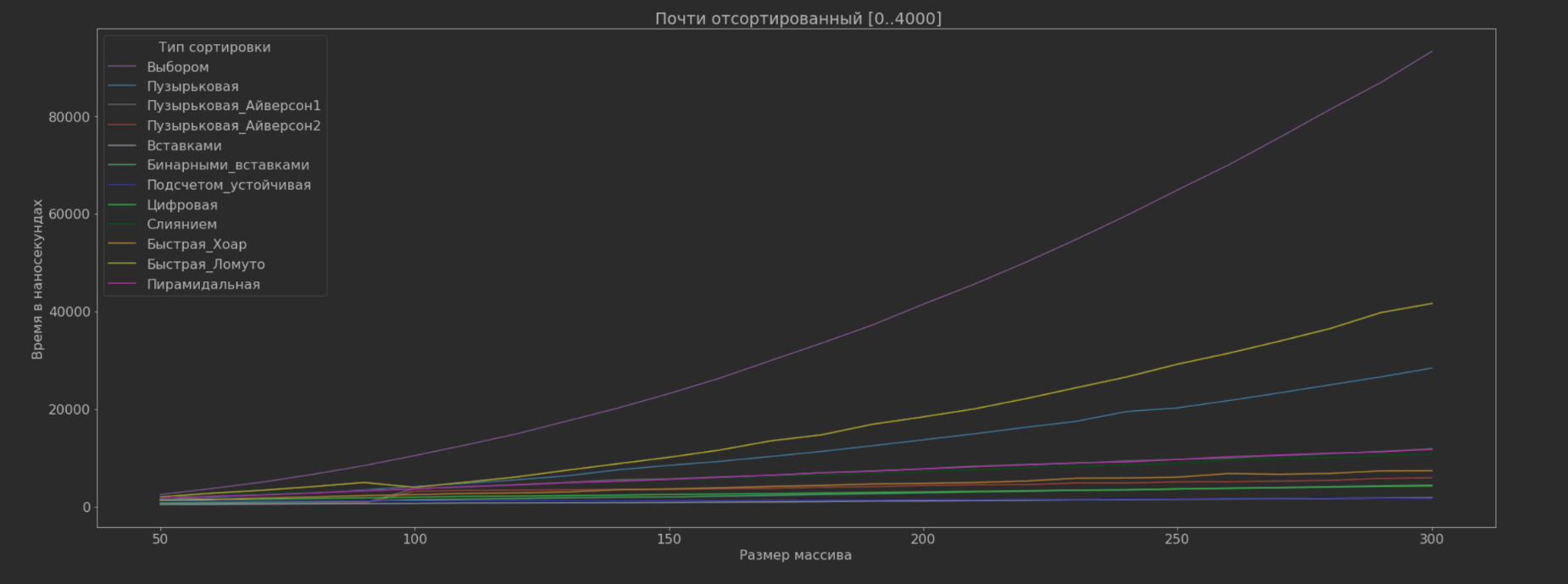


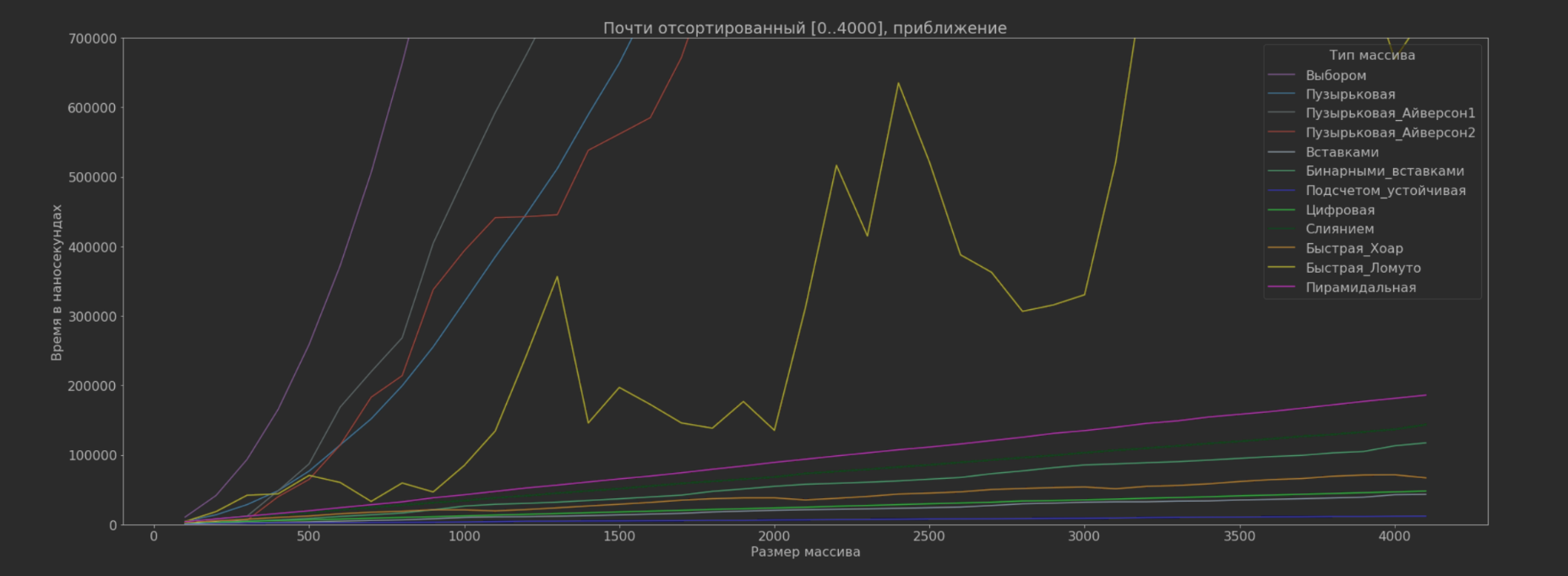
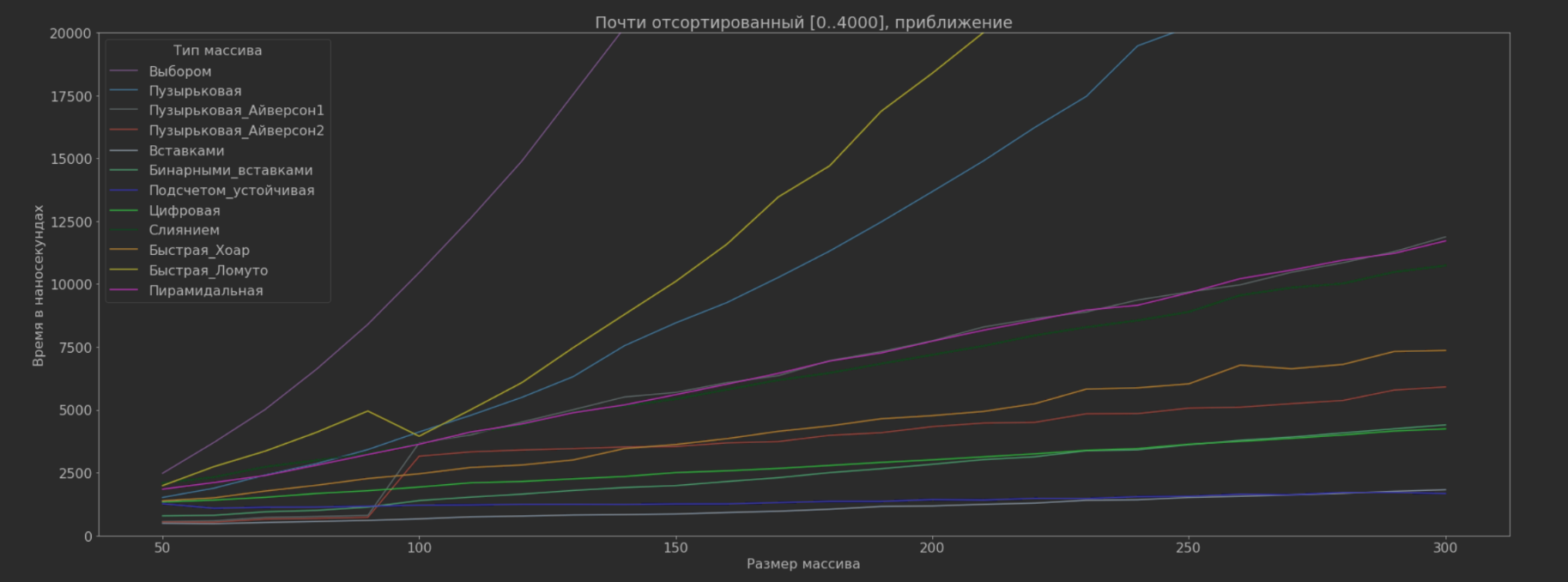
На случайных маленьких числах хорошо можно видеть, как различается сложность алгоритмов. Квадратичные быстро уходят вверх, логарифмические похож на прямые под углом в 30 градусов, а линейные вовсе на прямую x = 0.





Ситуация похожа на предыдущий пункт, только цифровая в данном случае имеет больший угол наклона.





Видно, как сильно прыгает быстрая сортировка с разбиением Ломуто при почти отсортированных массивах. Скачки наблюдаются, вероятно, в тех местах, где было поменяно много элементов.

