Лабораторная работа оценка времени работы разных типов сортировок

Королев Дмитрий Алексеевич

12 октября 2025 г.

1 Введение

Оценим время работы разных сортировок. В работе я использовал три вида квадратичных сортировок - пузырёк, сортировка вставкой, и шейкерная сортировка; три вида логарифмических сортировок - кучей, слиянием и быстрая сортировка Хоара, которая(с некоторыми улучшениями) используется как стандартная std::sort.

Я делал измерение для размера массива от 1 до 4000 с шагом 1. При 4000 квадратичная сортировка работает заметно дольше лоагрифмической.

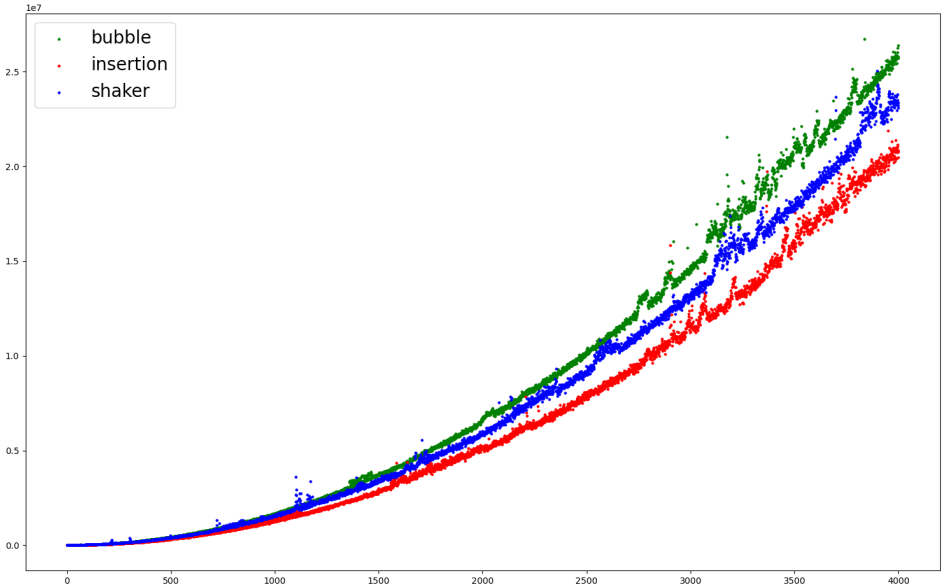
Графики анализировал, строил в Python при помощи matplotlyb.

2 Квадратичные сортировки

2.1 Пузырек, вставка, шейкер

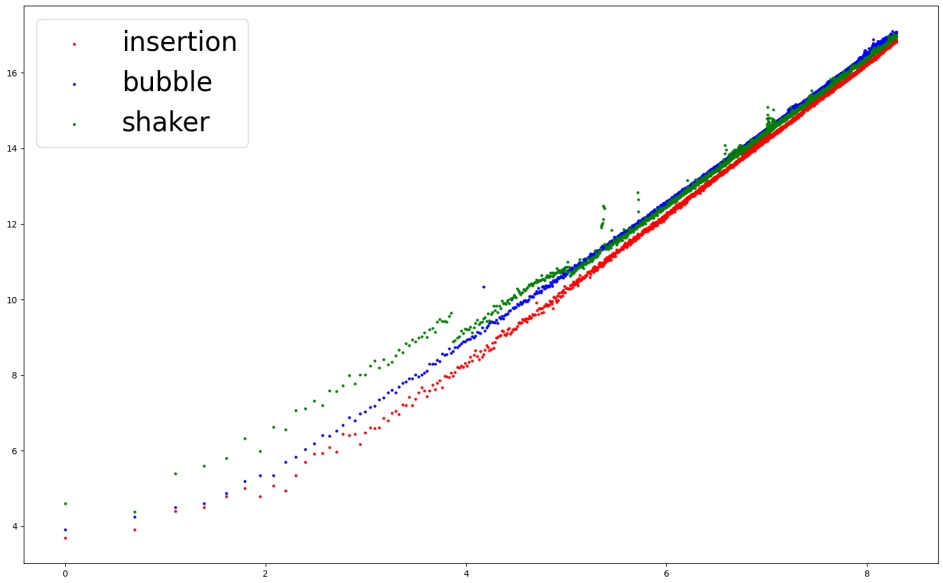
Пузырек - самая простая, и в то же время самая долгая из всех 6 сортировок. Мы просто меняем соседние элементы, если они расположенным в неверном порядке, проходясь так от начала до конца, пока не отсортируем. Шейкер - модификация пузырька, в которой мы одновременно проходим слева направо и справа налево. Сортировка вставкой - сортировка, основанная на принципе сортировки колоды карт - текущую карту засовываем в нужное место массива из всех предыдущих карт. Эта сортировка может быть улучшена до $nlogn$

Графики зависимости длины массива от времени его сортировки:



2.2 Логарифмический масштаб

Как видим, худшая сортировка - пузырек, лучшая - вставкой. Убедимся, что это квадратичная зависимость, построим этот же график, но в логарифмических осях(т.е. $ln(t), ln(n)$):



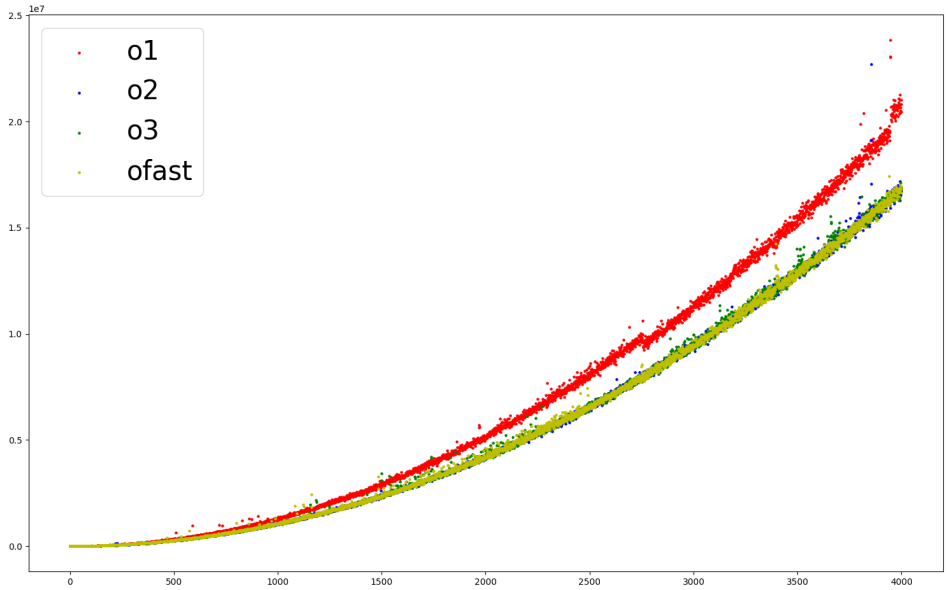
Получили график прямой пропорциональности. Так и должно быть, т.к.:

$$t = x^2 => ln(t) = ln(x^2) = 2ln(x)$$

Можно убедиться, что коэффициент пропорциональности - 2

2.3 Оптимизации

Посмотрим, сильно ли улучшит работу оптимизация кода. Использовал оптимизации o1,o2,o3 и ofast.

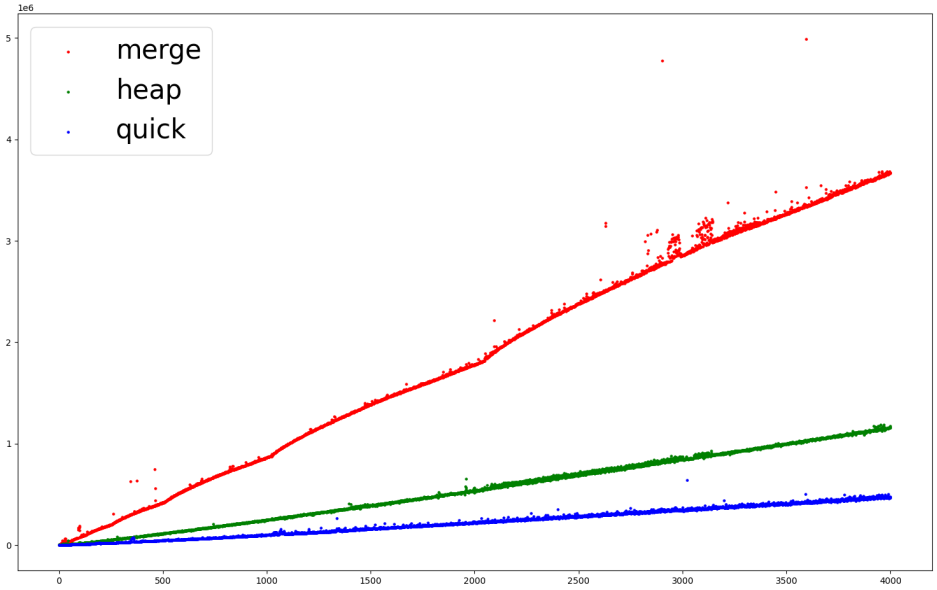


Как видим, оптимизация играет немаленькую роль. Однако на данном коде o2,o3 и ofast ничем не отличаются.

3 Логарифмические сортировки

3.1 Слияние, Хоар и heap-sort

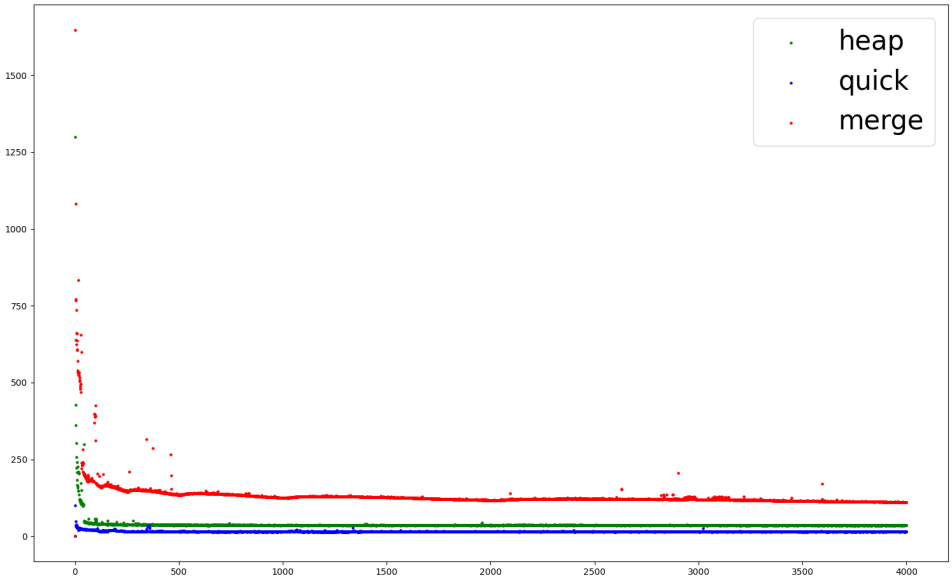
Я выбрал три сортировки, которые работают за *nlogn*. Построим аналогично их графики:



Как видим, самая быстрая - быстрая(:)) сортировка.

3.2 Логарифмический масштаб

Покажем, что зависимость логарифмическая, построив в масштабе $t/NlogN - N$:



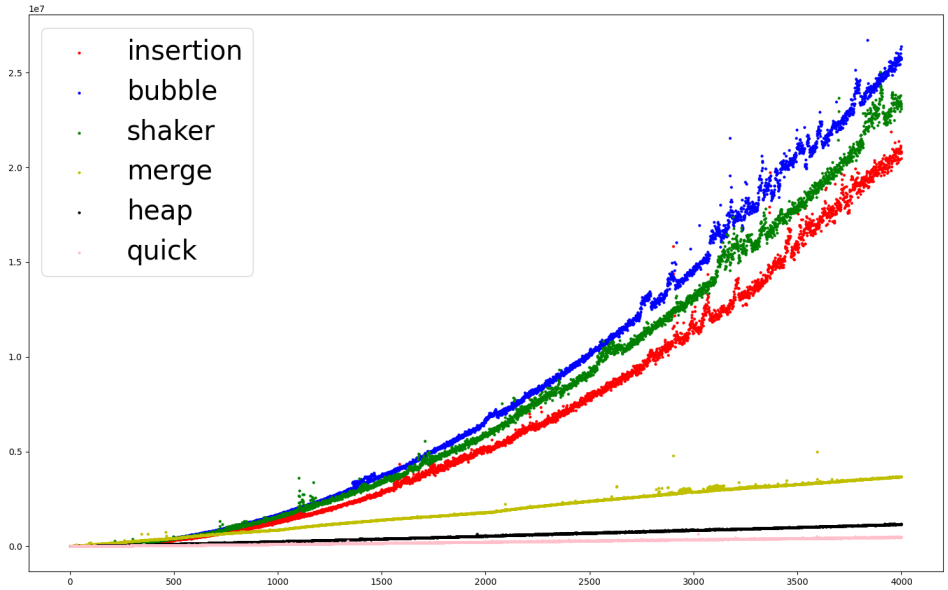
Получили константу, причем самая большая - у merge, самой долгой из быстрых.

3.3 Оптимизации

Оптимизации не принесли ничего полезного.

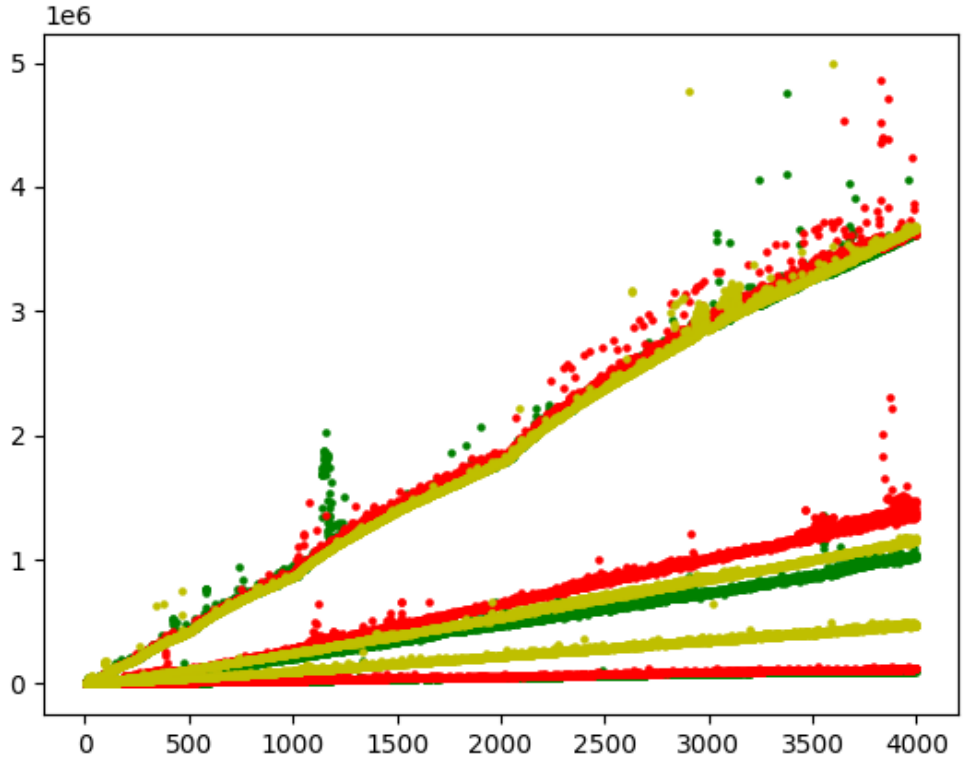
4 Общее сравнение

Сравним все сортировки на одном графике



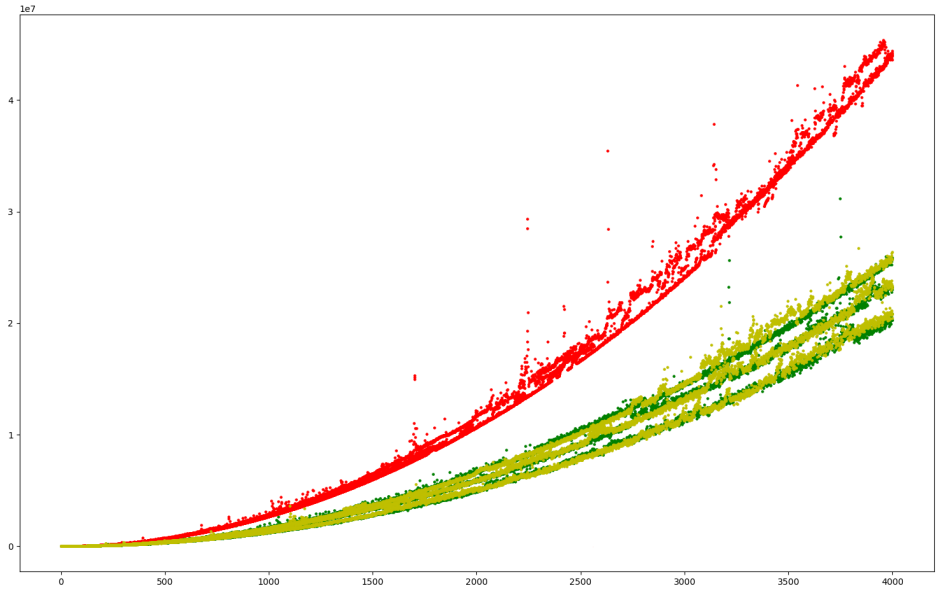
5 Зависимость от входных данных

Теперь посмотрим, что будет, если подавать худшие/лучшие варианты массива. До этого мы подавали случайные значения. Даля логарифмических сортировок:



Верхняя область - merge-sort. Средняя область - heap-sort. Нижняя область - quick-sort.

Как видим, больше всего "страдает"от случайных данных быстрая сортировка.

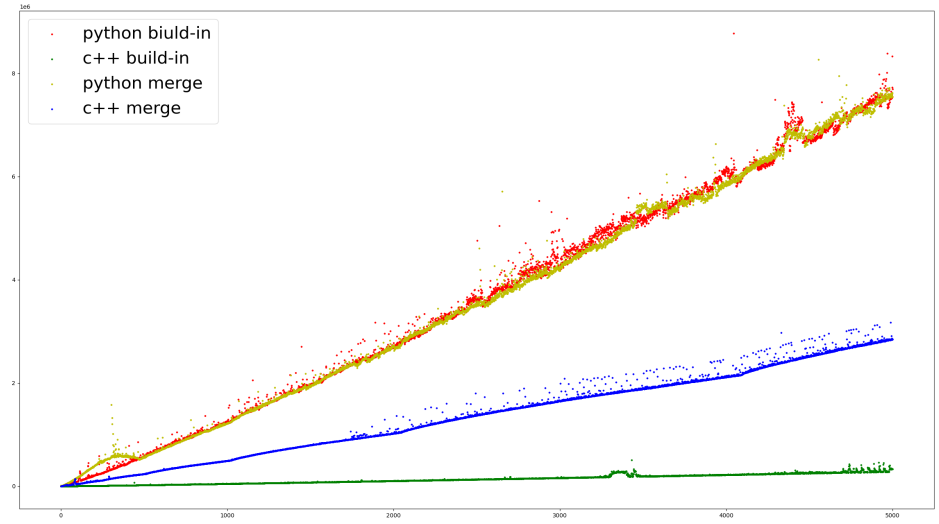


Графики для квадратичных сортировок.

Видно, что графики работают в два раза дольше на худших вариантах, а на лучших примерно также, как и на средних.

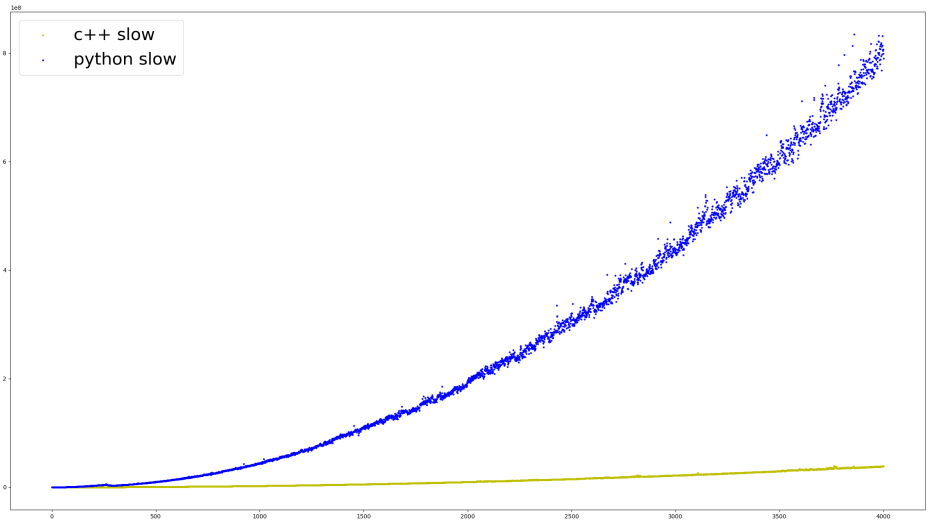
6 Python против C++

Теперь сравним время работы c++ и python.



Как видим, питон сильно уступает c++ по времени работы даже на логарифмических сортировках.

Если говорить про квадратичные сортировки, то получается следующее:



Как видно, Python работает на порядок медленнее