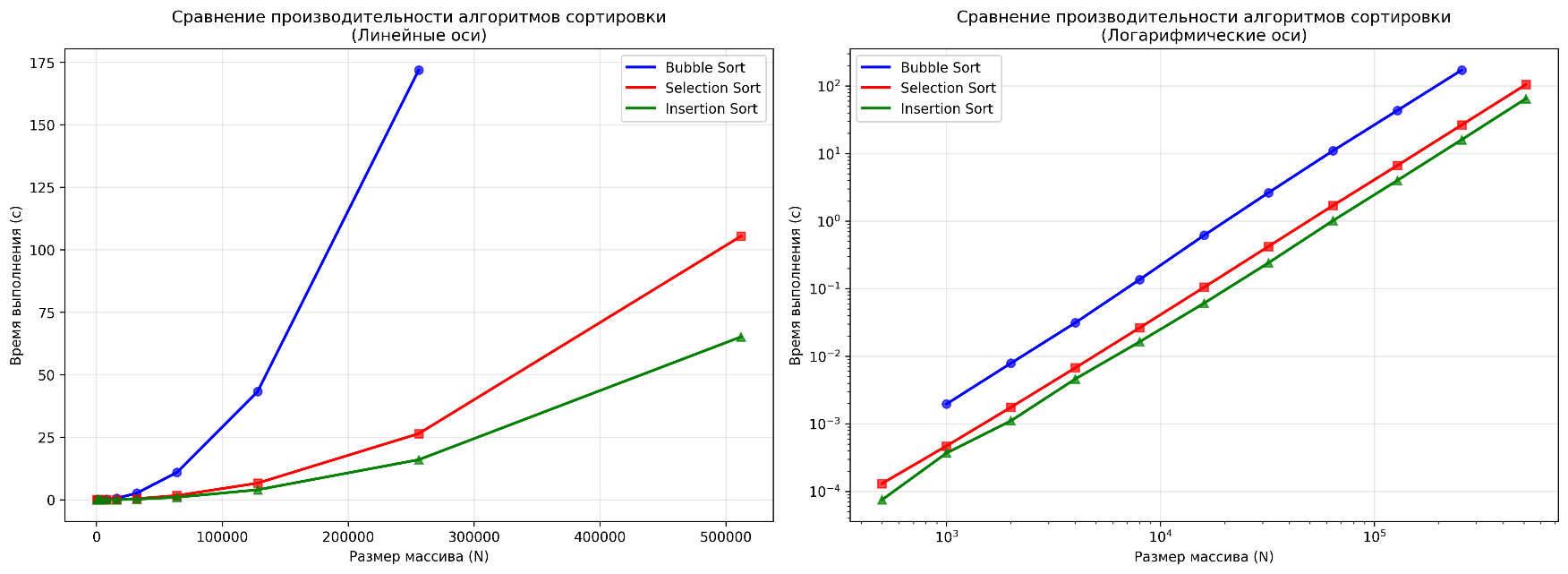
# Пузырёк и его товарищи

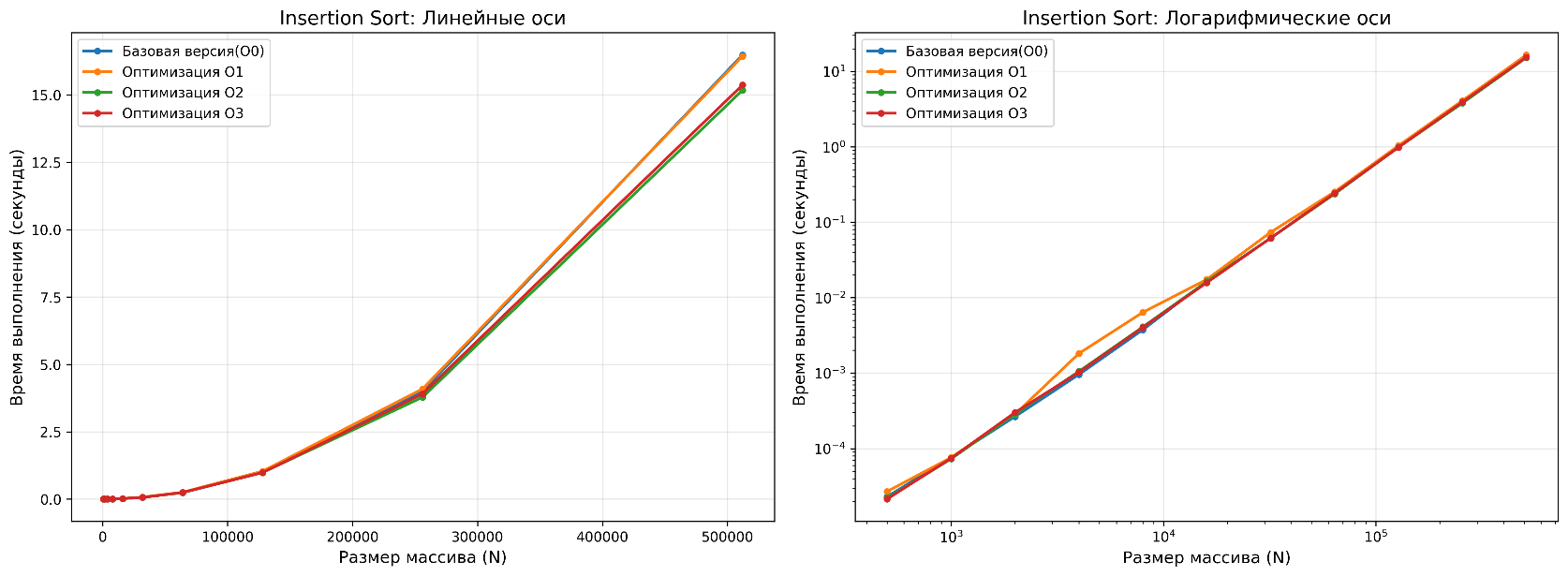


Для изучения были выбраны 3 сортировки O(N2): bubble sort (сортировка пузырьком), selection sort (сортировка выбором), insertion sort (сортировка вставками).

На 1 графике для каждой сортировки построены зависимости времени исполнения от размера массива. Видно, что самой быстрой является insertion sort, а самой медленной ­– bubble sort.

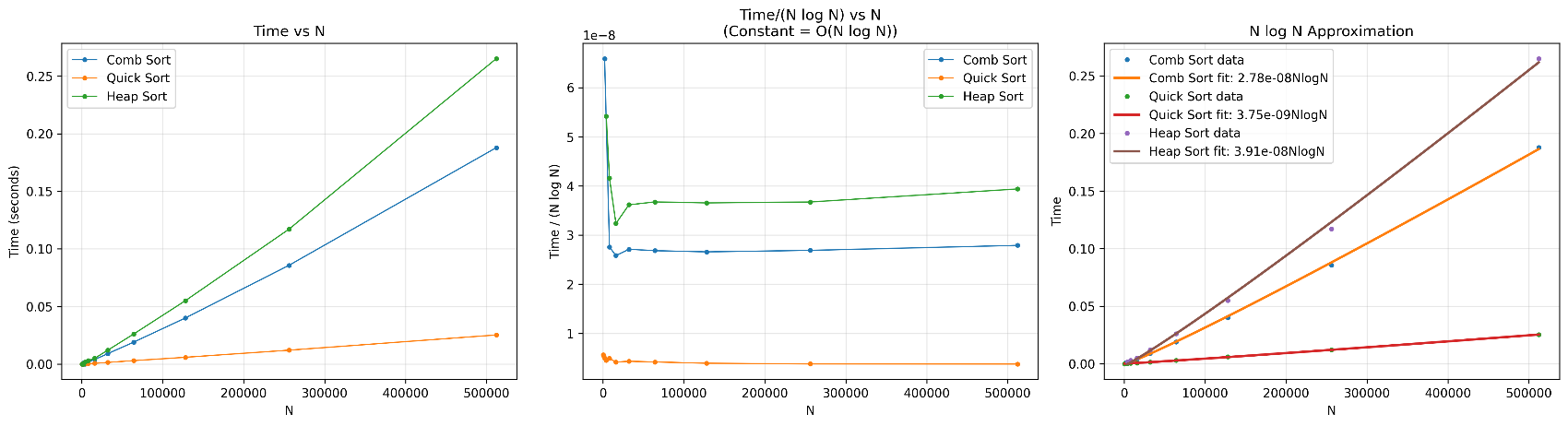
На 2 графике зависимости построены в log координатах, чтобы доказать, что сложность данных сортировок – N2. Ведь если то . Это будет прямая с угловым коэф – 2. Что на графиках и получилось.

# Пузырёк, но быстрее



Для изучения зависимости времени исполнения от степени оптимизации была выбрана insertion sort. По графикам видно, что степень оптимизации почти не влияет на время выполнения сортировки (различие на N = 500\_000 меньше 1 секунды).

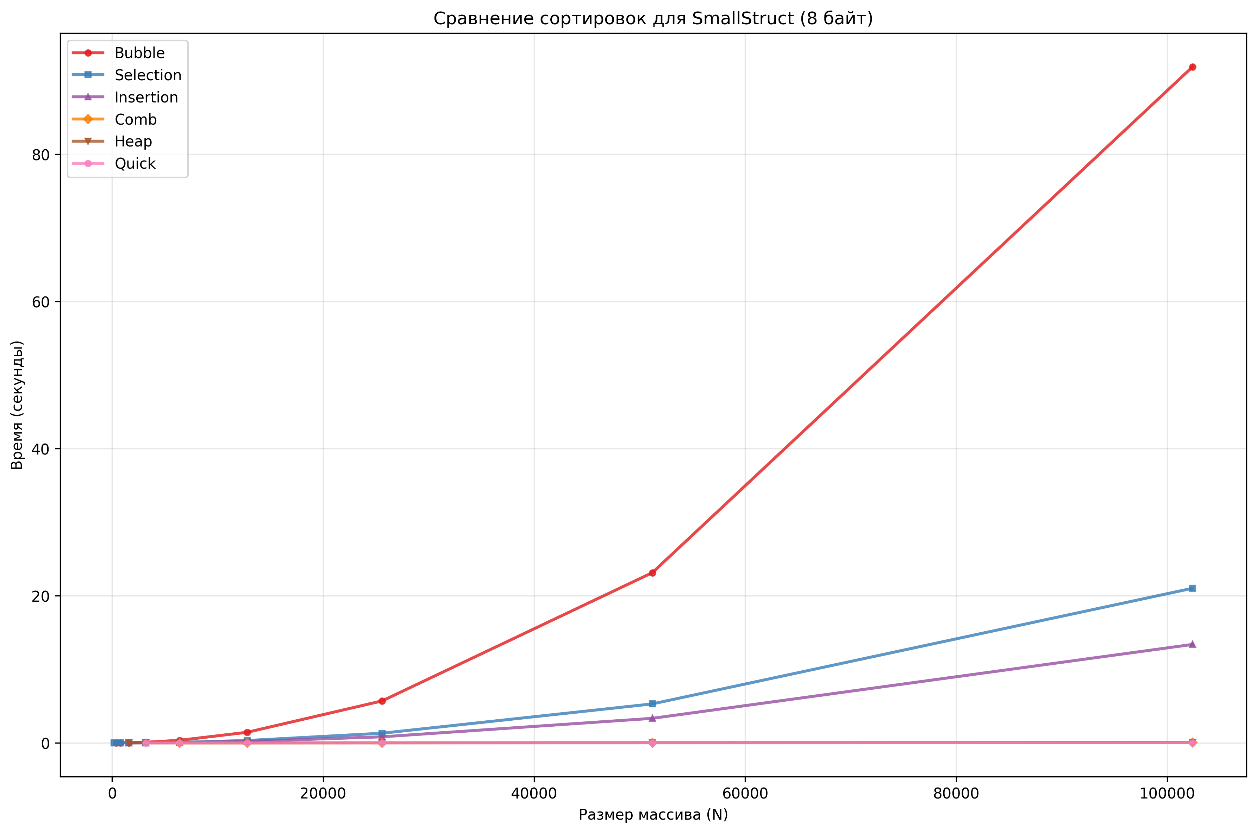
# А теперь настоящие быстрые сортировки



Для изучения быстрых сортировок были выбраны: comb sort (сортировка расчёской), heap sort (сортировка кучей), quick sort (сортировка Хоара).

Были построены 3 графика:

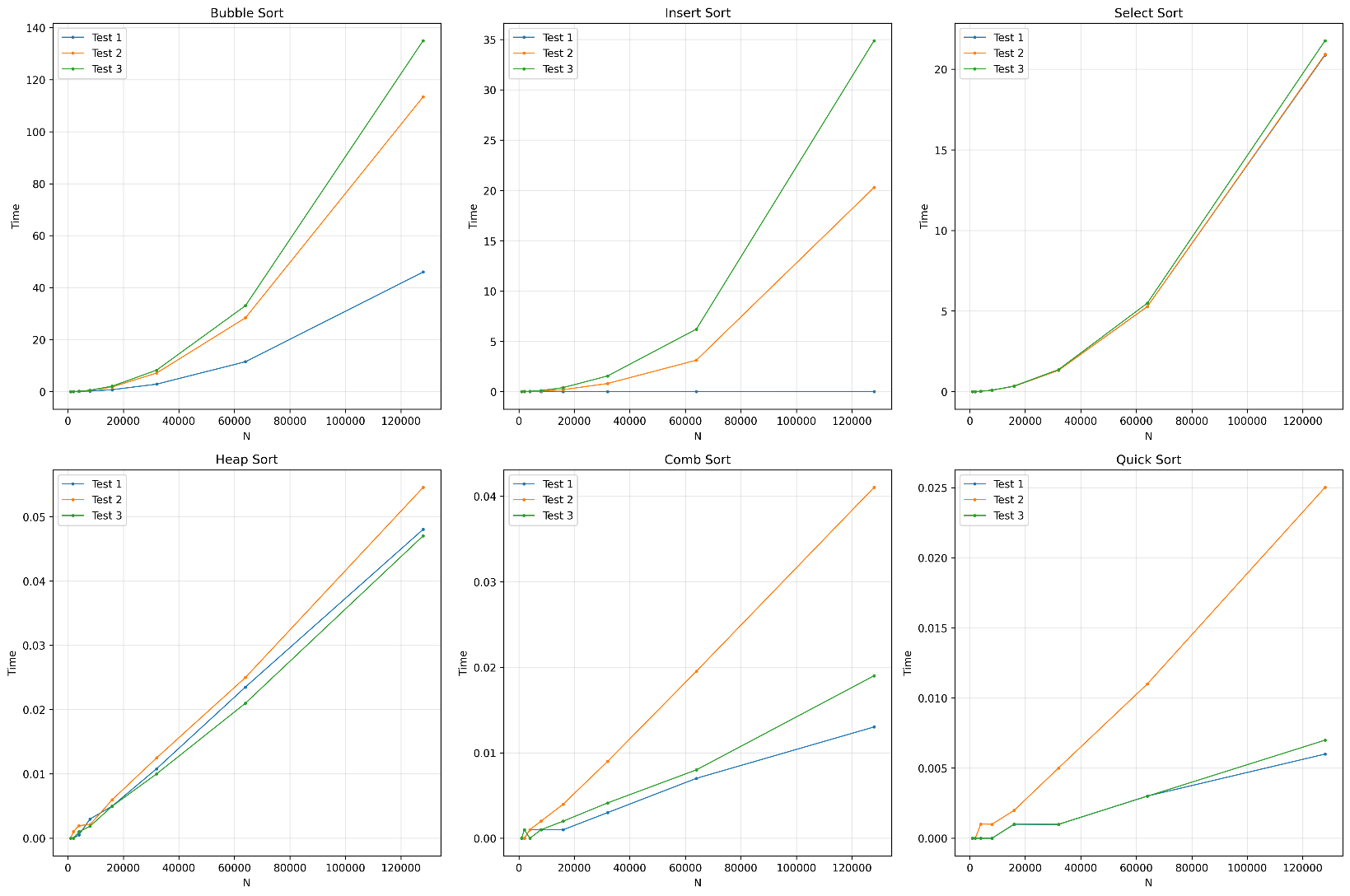
1. Обычная зависимость времени от N. По которой мы видим, что самая быстрая – quick sort, самая медленная – heap sort.
2. График зависимости , чтобы доказать, что сложность у данных сортировок именно такая. Ведь если . Из графиков мы это и видим, начиная с относительно больших N, график примерно const. При малых N график ведёт себя там из-за влияния как раз констант, о которых мы будет говорить чуть позже.
3. Ещё один способ доказать, что сложность алгоритмов ., построить по точкам аппроксимацию нужной зависимостью и изучить отличие полученной зависимости от теоретической. Это также привело к доказательству, что сложность алгоритмов .



На одном графике построены зависимость времени исполнения от N для всех шести сортировок, которые мы изучали ранее. Степень оптимизации для всех – О0.

По графику видно, насколько сортировки быстрее сортировок .

# Зависимость от начальных данных

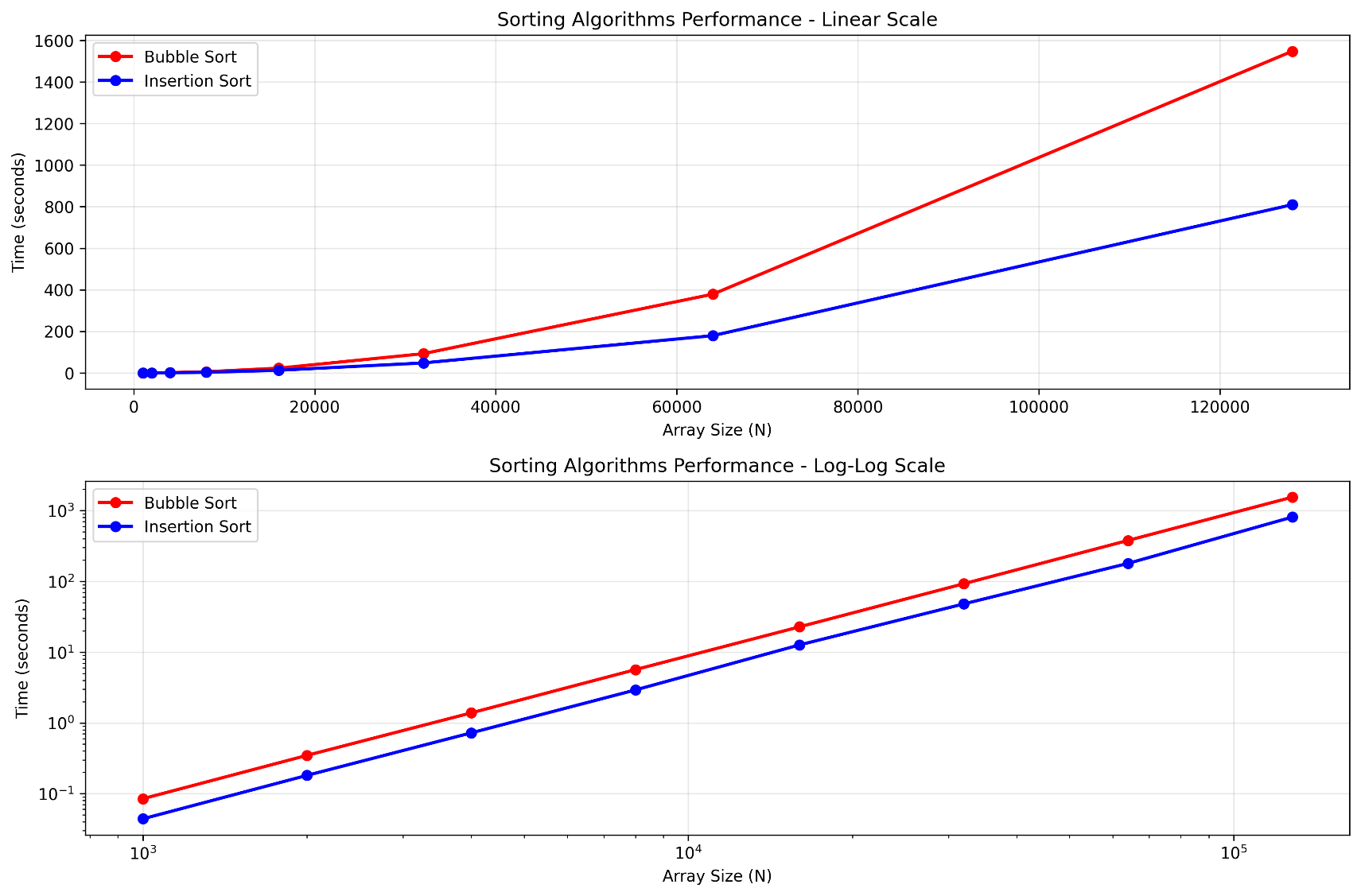


В данном пункте надо изучить, как начальный вид массива влияет на время выполнения. То есть рассмотреть 3 случая: best (лучший – массив уже отсортирован, на графиках – Test 1), rand (средний – обычный рандомный массив, на графиках – Test 2) и worst (худший – массив отсортирован в обратном порядке, на графиках – Test 3).

Посмотрим, как себя повела каждая сортировка в каждом случае.

1. Bubble sort: Закономерно время на лучший случай – минимальное, а на худший – максимальное. При этом отличия при больших N существенны. При всех случаях явно сохранилась сложность .
2. Insertion sort: В лучшем случае показала очень хороший результат, явно не В худшем закономерно медленнее выполнилась чем в среднем и в обоих сложность - .
3. Selection sort: Во всех 3 случаях сохранилась сложность Отличие по времени между лучшем и средним случаем практически отсутствует, а между ними и худшим очень мало.
4. Heap sort: Отличие между всеми 3 случаями невелико (. Средний случай выполнялся дольше всех, а худший быстрее всех.
5. Comb sort: Средний случай выполнялся дольше всех, а лучший быстрее всех. Относительные отличия между временем выполнения уже велики (
6. Quick sort: Лучший и худший случаи выполнились примерно одинаково (лучший чуть быстрее). Средний случай выполнялся в раз дольше.

# \*Python and C++



В данном пункте было оценено время выполнения bubble sort и insertion sort в языке Python.

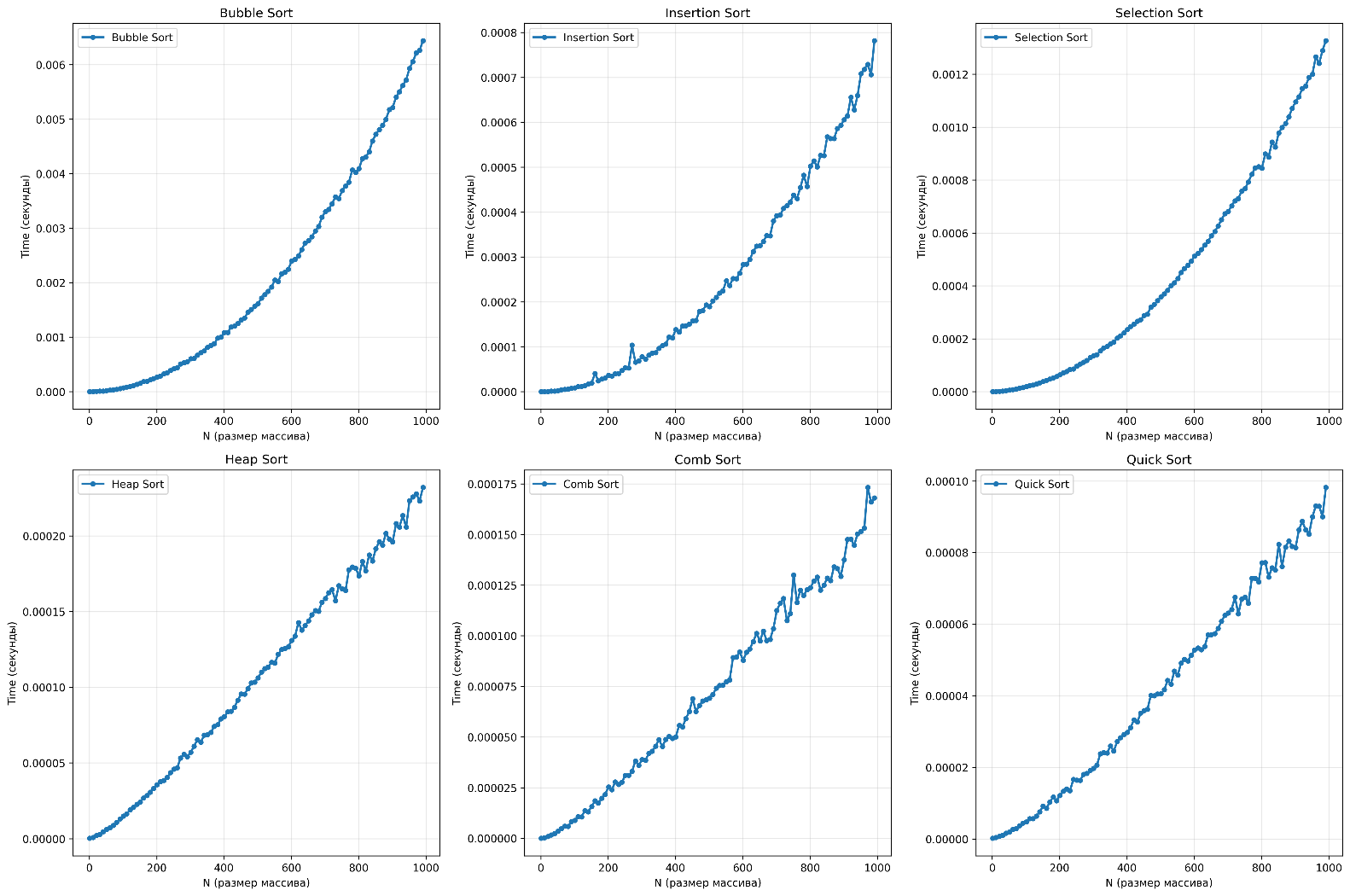
Явно видно, что время выполнения в разы больше.

Если мы вернёмся к 0 пункту и посмотрим, сколько там выполнялись эти сортировки при таких же N, то увидим, что время выполнения Bubble sort при А на Python при тех же N, время выполнения .

А для insertion sort на C++ при тех же N время выполнения , для Python

В итоге, на Python сортировки выполнялись в порядок 100 раз медленнее.

# 6.1. \*Небольшие массивы



В данном пункте исследуется поведение сортировок при небольших . Когда влияние на время выполнения оказывают константы, которые заложены в сортировки.

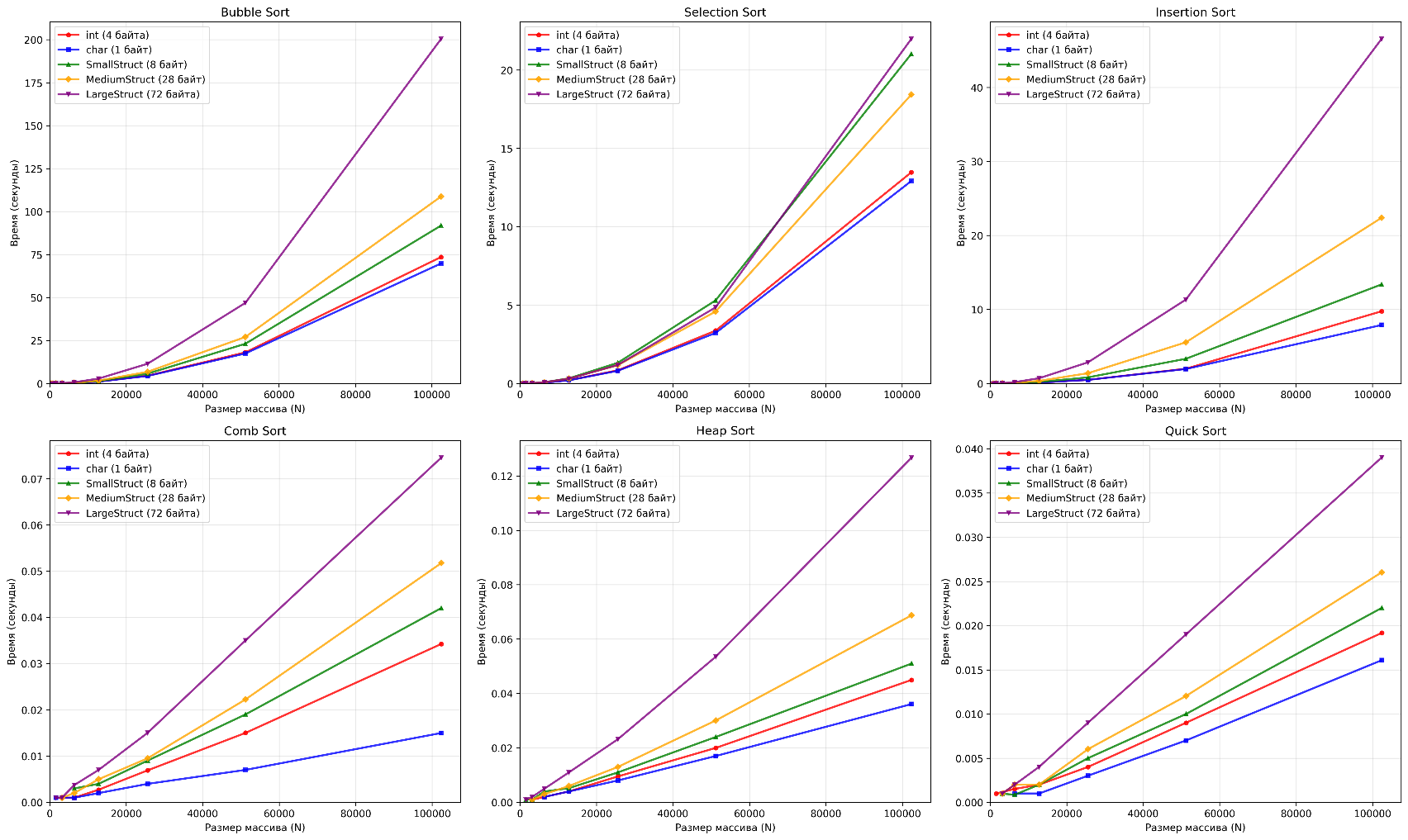
Чтобы добиться некой плавности в графиках, пришлось проводить 10000 «прогонов» каждой сортировки и усреднение результатов.

По графикам видим, что bubble sort опять же самая медленная из всех сортировка. А вот insertion и selection sorts по порядку времени уже сравнимы с быстрыми сортировками. Но всё равно медленнее выполняются.

При больших N selection sort выполнялась быстрее, чем insertion sort, а вот при маленьких N всё наоборот, что значит, что константа, заложенная в selection sort меньше константы, заложенной в insertion sort.

В других сортировок подобного не произошло.

# 6.2. \*Зависимость от типа данных



В данном пункте требовалось исследовать зависимость времени выполнения сортировок в зависимости от типа данных, хранящихся в массиве.

Сортировки те же, типы были выбраны: int, char, small structure (2 int), medium structure (3 int, 2 double), large structure (5 int, 3 double, массив из 8 int).

Результаты достаточно логичны. Для тяжелых типов время больше независимо от вида сортировки. Отношение времени выполнения для самого тяжёлого типа (LargeStruct(72 байта)) и для самого лёгкого (char(1 байт)) в зависимости от сортировки.

Меньше всего показания отличаются для selection sort и quick sort. Больше всего для comb sort и для insertion sort.