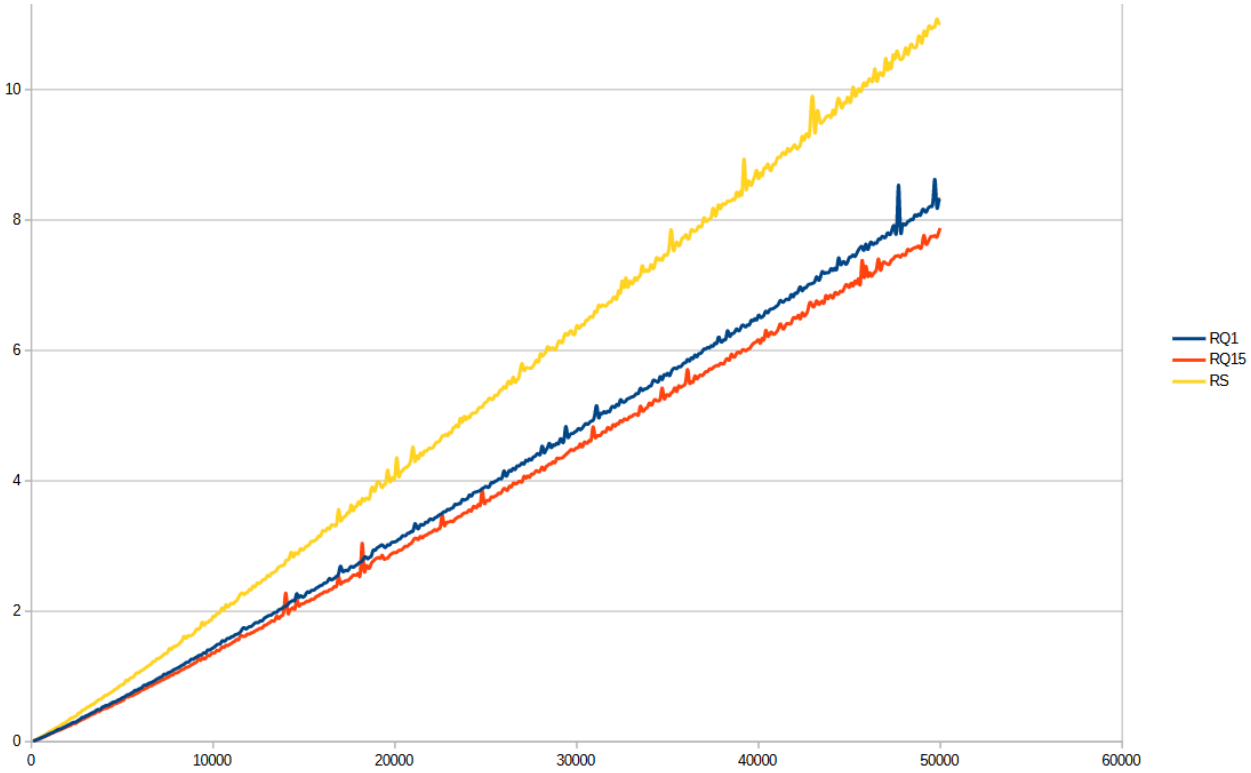
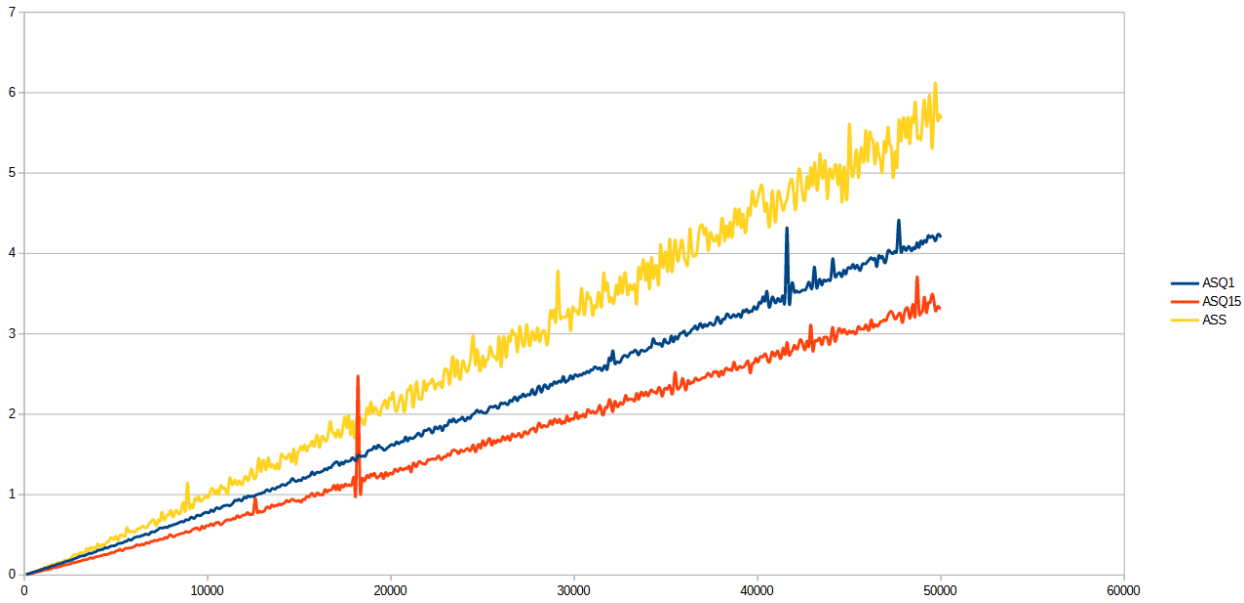
A2.

Рассмотрим график зависимости времени выполнения алгоритмов от размеров рандомных массивов. Абсцисса - размер массива, ордината - время в микросекундах:



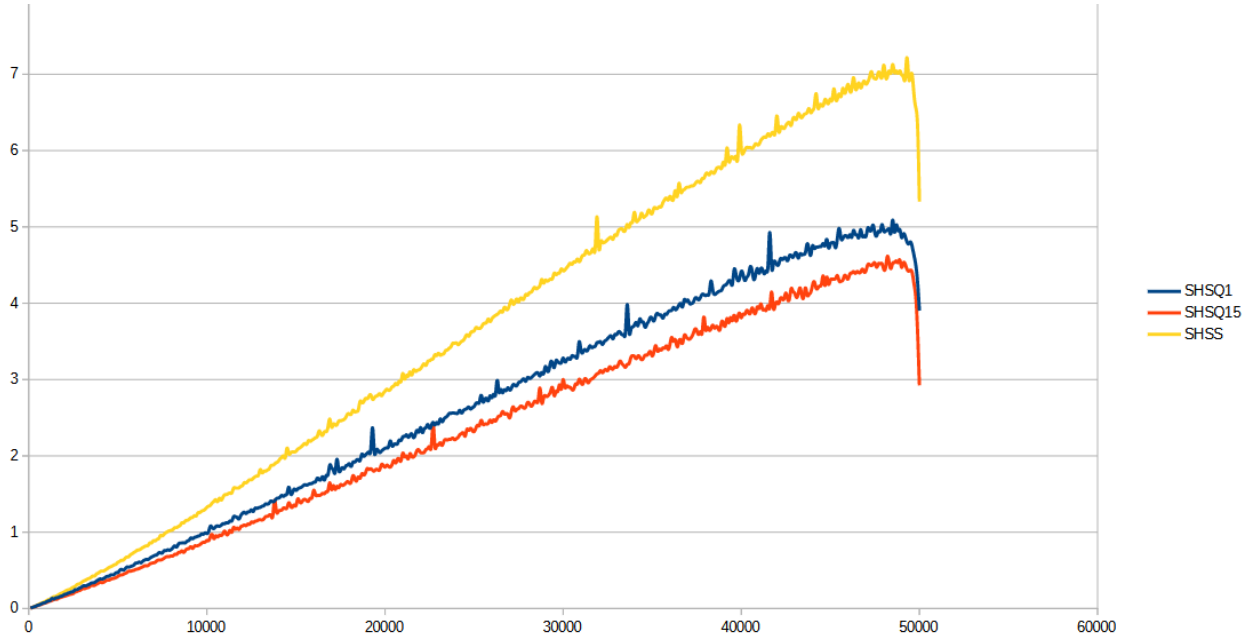
Мы видим, что алгоритм быстрой сортировки без оптимизаций работает в среднем медленне, чем алгоритм с оптимизациями. Но на данных до 50000 эта разница почти не ощутима. Также стоит отметить, что std::sort работает примерно с такой же скоростью, что и алгоритмы быстрой сортировки, но чуть медленнее их.

Теперь рассмотрим график зависимости времени выполнения алгоритмов от размеров отсортированных в обратном порядке массивов. Абсцисса - размер массива, ордината - время в микросекундах:



Сначала необходимо заметить, что все сортировки выполняются в этом случае быстрее, чем на рандомных массивах. Но на удивление оптимизированная сортировка в этом случае также показывает наилучший результат. Хотя казалось бы, для сортировки вставками массив, отсортированный в обратном порядке, является худшим случаем. Но это можно объяснить особеннотью выбранного partition, благодаря которому с большой вероятностью массивы маленьких размеров уже почти отсортированны(5 4 3 2 1 с пивотом в 3 -> 1 2 3 4 5)

Теперь рассмотрим график зависимости времени выполнения алгоритмов от размеров почти отсортированных массивов(число перестановок элементов $\approx 2\%$). Абсцисса - размер массива, ордината - время в микросекундах:



Здесь мы видим, что тенденция продолжается и quicksort с оптимизацией снова выигрывает у обычной быстрой сортировки и встроенной сортировки. Сразу бросается в глаза, что в размерах, близких к 50000 идёт ощутимое уменьшение времени выполения алгоритма. К сожалению точно сказать причину этого я не смог, но эксперементально утсановил, что виной этому то, что размер сортируемого массива почти совпадает с размером сгенерированного массива.

Id посылки: 293124936

Github: https://github.com/Tema123321/A_DS/tree/master/Set3_A3