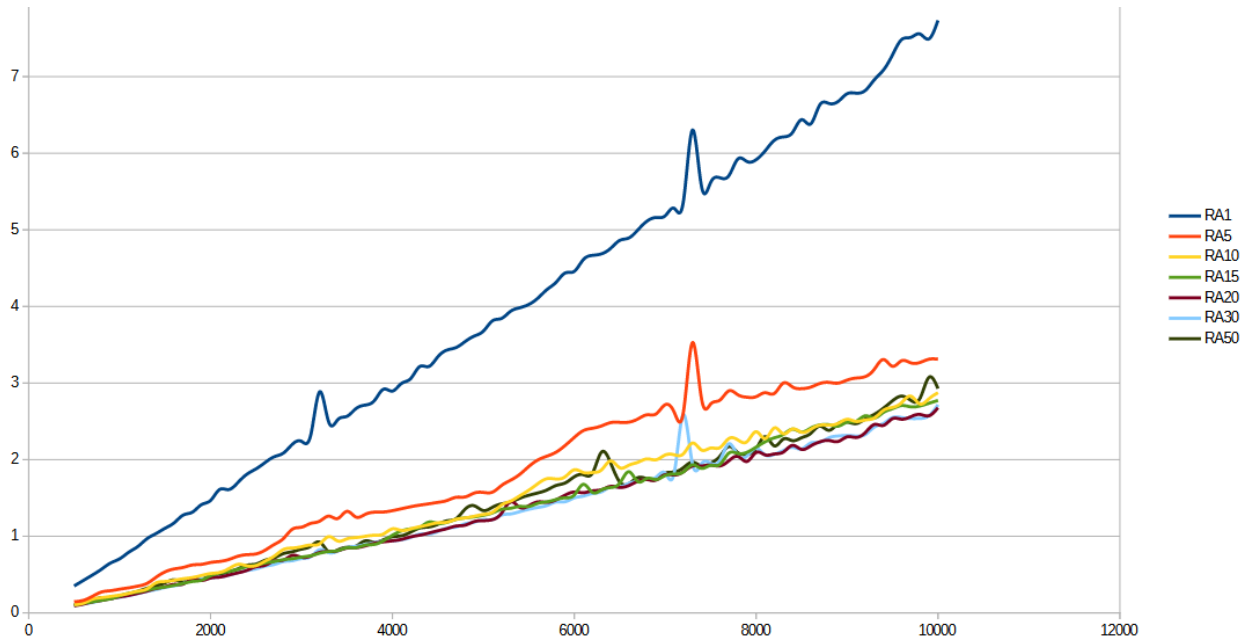
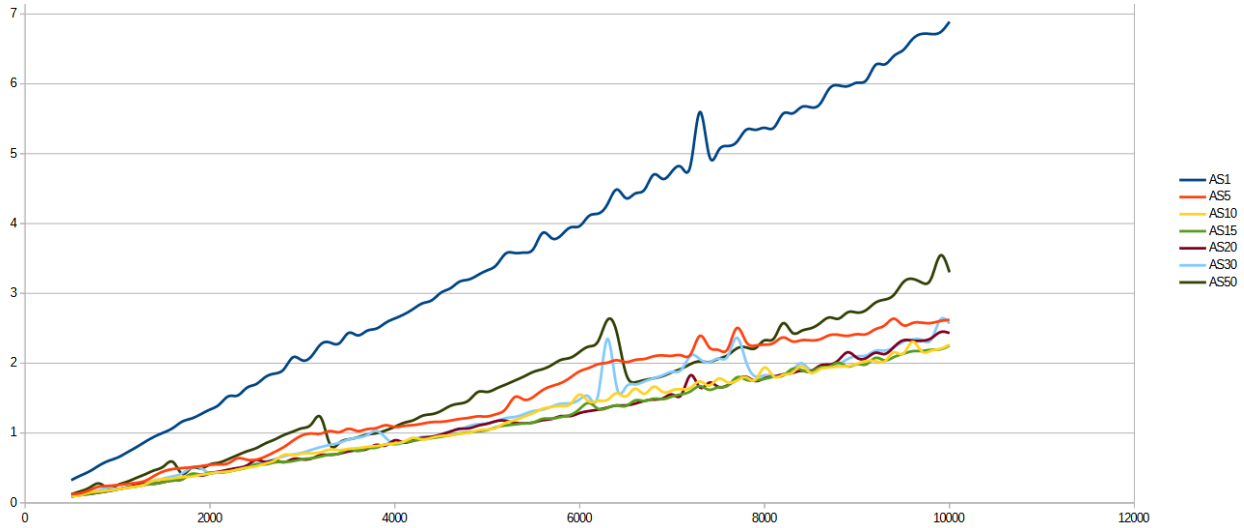
A2.
Рассмотрим график зависимости времени выполнения алгоритмов от размеров рандомных массивов. Абсцисса - размер массива, ордината - время в микросекундах:



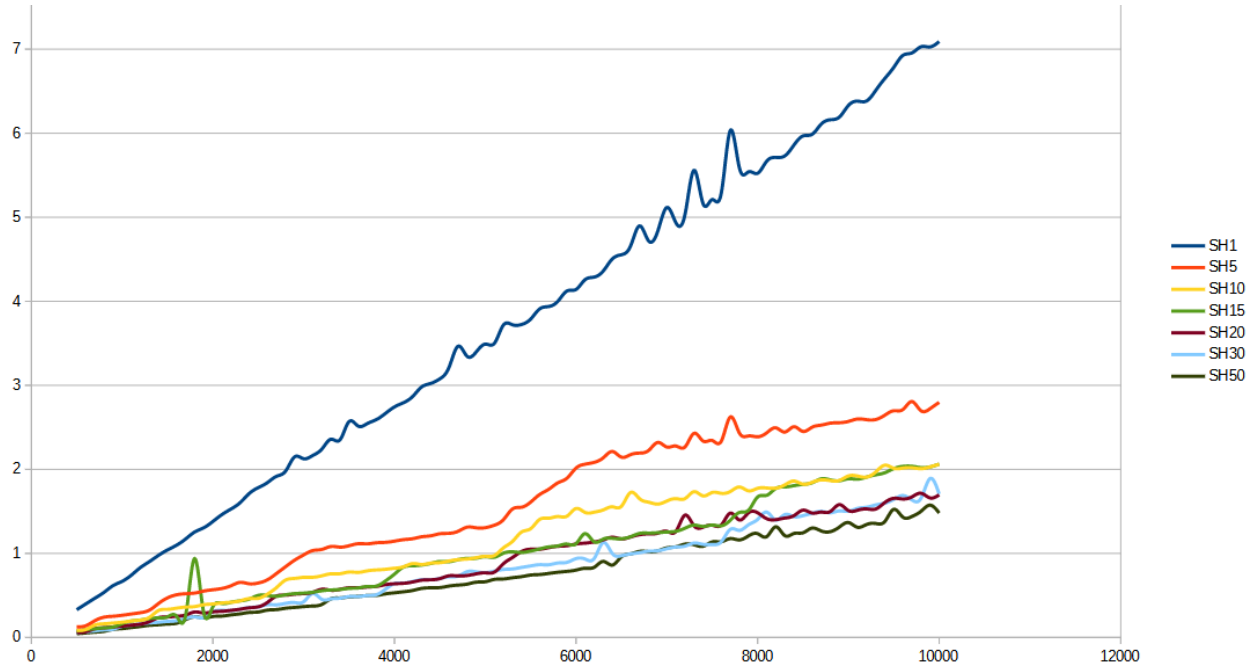

Мы видим, что алгоритм mergesort хоть и имеет $O(nlog(n))$ сложность проигрывает алгоритмам mergesort с переходом на insertionsort на маленьких массивах, не смотря на то, что сложность insertionsort $O(n^2)$. Это можно объяснить тем, что затраты на вызов рекурсии довольно большие, а деление на 2 размера сортируемого массива рано или поздно перестаёт компенсировать растраты на рекурсию. Также можно заметить, что из выбранных границ перехода на insertionsort самыми оптимальными на данных до 10000 является RA20 и RA30.

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



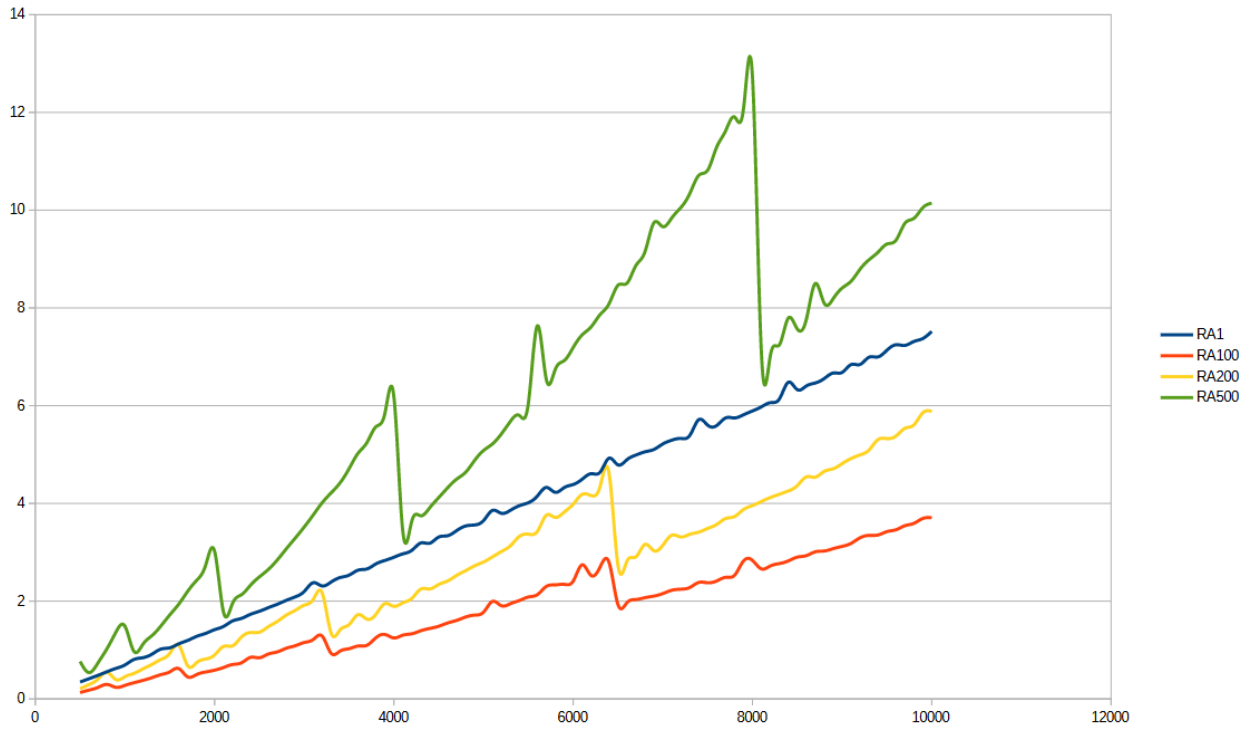

Сначала необходимо заметить, что mergesort выполняется в этом случае быстрее, чем mergesort на рандомных массивах. Это можно объяснить тем, что при выполнении слияния двух подмассивов, сначала будет полность записан один из подмассивов, а потом второй, что возможно влечёт уменьшение времени выполнения алгоритма. Также можно заметить, что здесь AS50 начинает работать заметно медленнее других усовершенствованных mergesort. Это объясняется тем, что при увеличении порога перехода на insertionsort всё больший вклад вносит сортировка вставками, а для неё массив, отсортированный в обратном порядке, является худшим случаем, и поэтому AS50 показывает себя заметно хуже.

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 можно заметить, что mergesort опять не сильно отличается от предыдущих графиков. Но сраху бросается в глаза на графиках то, что улучшения сортировки слиянием на почти отсортированных массивах работают заметно быстрее. Это объясняется тем, что insertionsort работает довольно быстро на почти отсортированных массивах. И мы видим, что самым быстрым алгоритмом в этом случае становится с наибольшим порогом перехода на insertionsort, то есть тот алгоритм, где вклад insertionsort наибольший.

Теперь рассмотрим график зависимости времени выполнения алгоритмов с большим порогом перехода на insertionsort на рандомных массивах. Абсцисса - размер массива, ордината - время в микросекундах:




На графике мы видим, что уже с порогом в 500 mergesort+insertionsort является хуже по ассимптотике, чем обычная сортировка слиянием. То есть улучшать mergesort до бесконечности нельзя(. Ну и конечно, тяжело было не заметить скачки по времени в алгоритмах с переходом на insertionsort. Это можно объяснить на примере чисел 8000 и 8200 для алгоритма RA500, при длинне массива в 8000 мы делим массив пополам и получаем: 8000, 4000, 2000, 1000, 500 и после этого на 500 запускается insertionsort. А вот при 8200 последовательность будет: 8200, 4100, 2050, 1025, 512, 256 и только теперь запускается insertionsort, и тоесть при 8200 RA500 $\sim$ RA256. Поэтому идёт такое ускорение алгоритма, так как RA500 заметно медленнее по времени, чем RA256, из-за квадратичной сложности.

Id посылки: 292788511

Github: https://github.com/Tema123321/A_DS/tree/master/Set3_A2