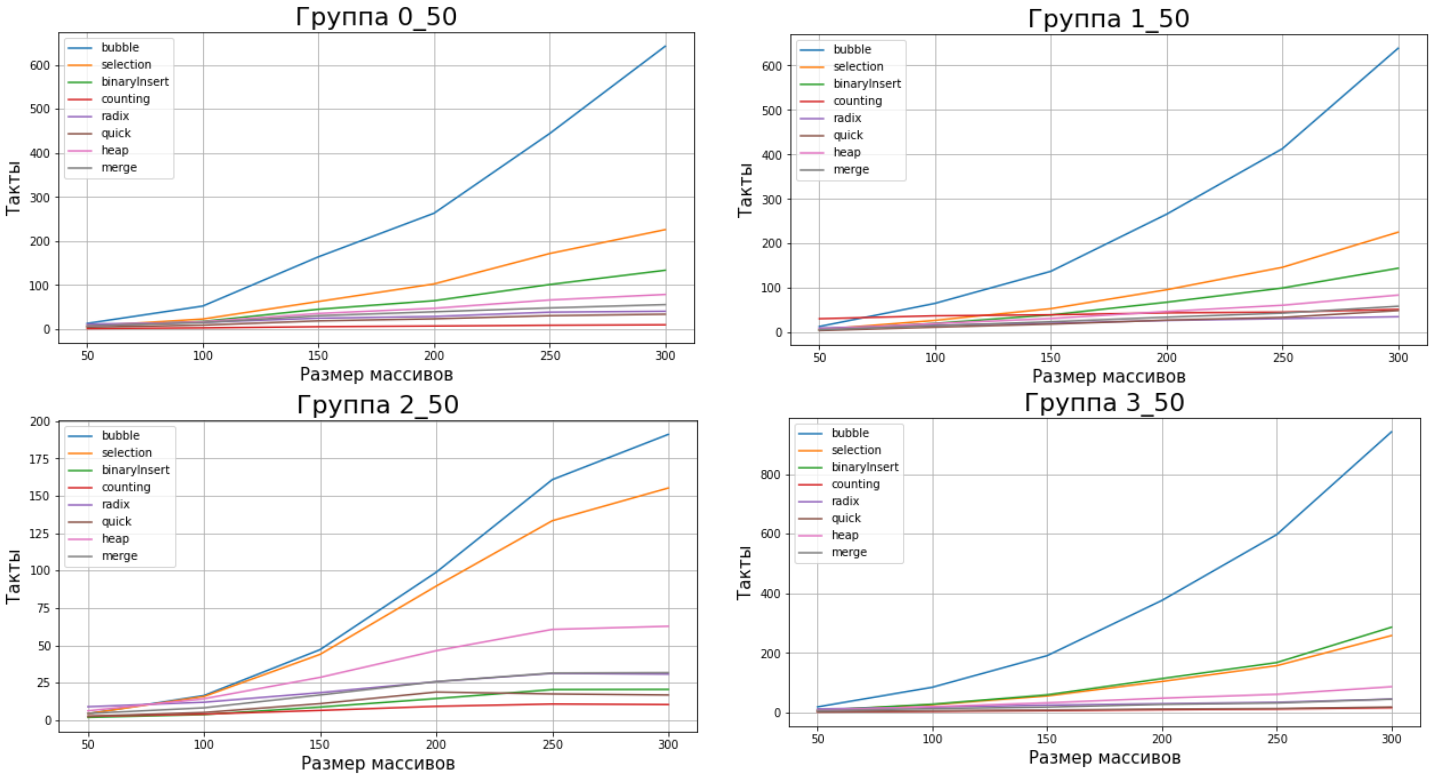
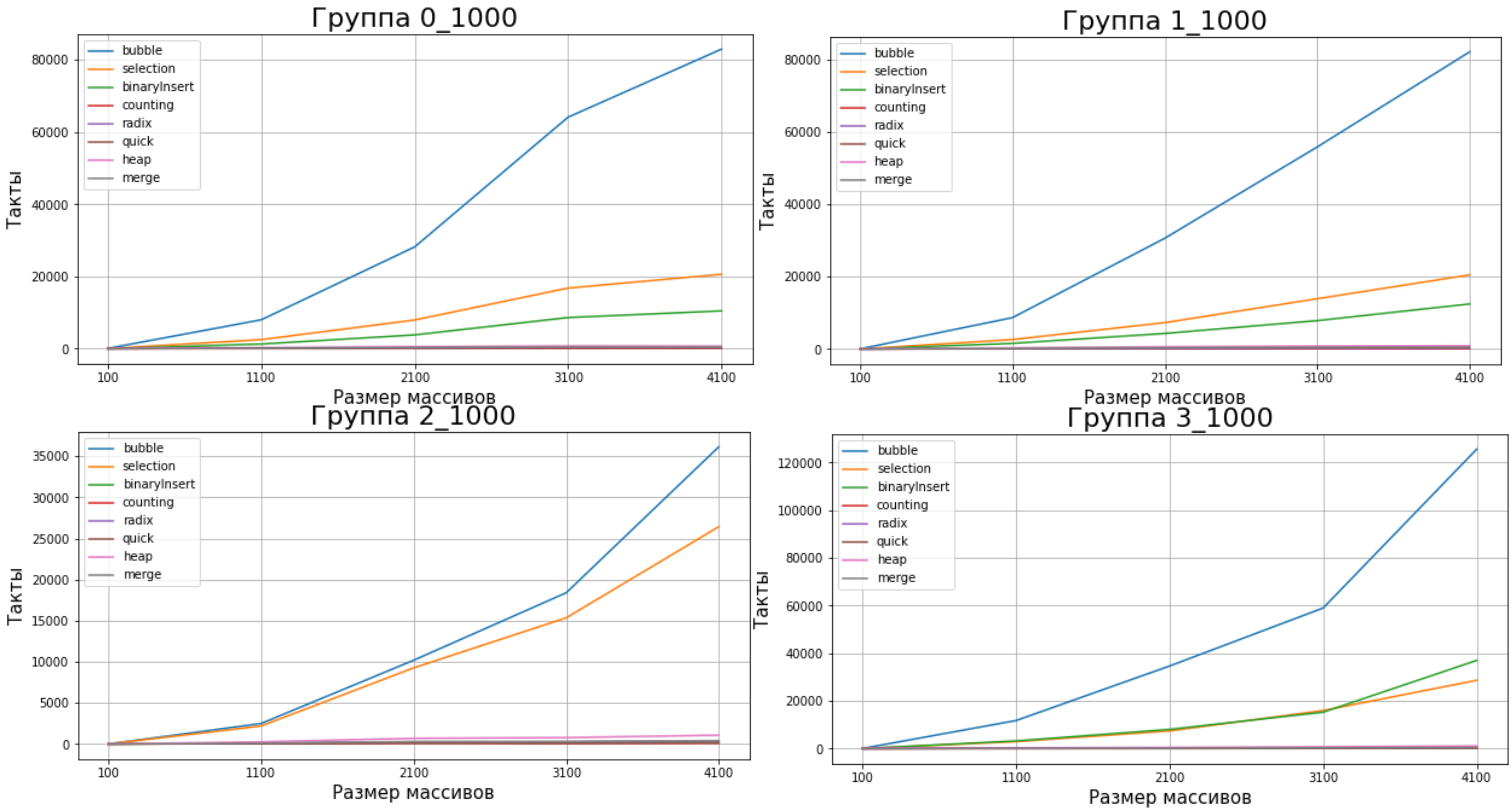
Отчет по сравнению сортировок
Ибрагимов Константин, БПИ192

Сравнение сортировок по каждой группе массивов с шагом 50:



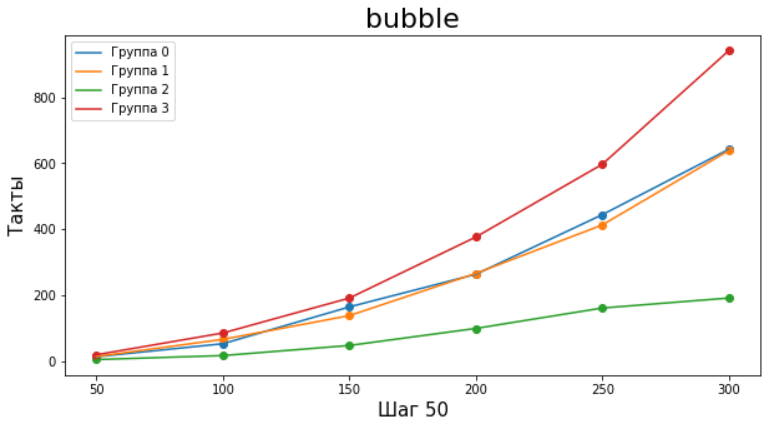
1. На графике 0_50, на котором изображена работа сортировок для массива, заполненного случайными числами от 0 до 5, мы можем видеть преимущество counting sort. При этом quick, heap и merge sort немного ей уступают. Хуже всего себя показал bubble sort.
2. На графике 1_50, на котором изображена работа сортировок для массива, заполненного случайными числами от 0 до 4000, мы можем видеть небольшое преимущество radix sort. При этом quick, counting и merge sort показали себя практически одинаково и очень близко к результату radix. В данном примере bubble sort снова оказался худшим.
3. На графике 2_50, на котором изображена работа сортировок для почти отсортированного массива, мы можем наблюдать преимущество counting sort. Интересно, что binaryInsert sort и quick sort показали очень похожие результаты, а их графики даже пересеклись при переходе от размера 200 до 250. Так же интересно, что counting, quick binaryInsert, merge и radix sort не увеличили время обработки массива после перехода размеров 250-300. Худший в этом случае снова bubble.
4. На графике 3_50, на котором изображена работа сортировок для отсортированного в обратном порядке массива, мы можем наблюдать, что counting, radix и quick sort показали себя одинаково хорошо. Bubble тут снова худший выбор.

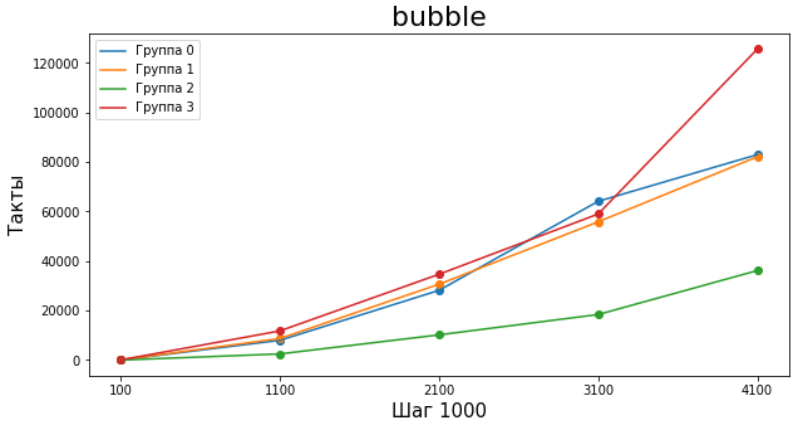
Сравнение сортировок по каждой группе массивов с шагом 1000:



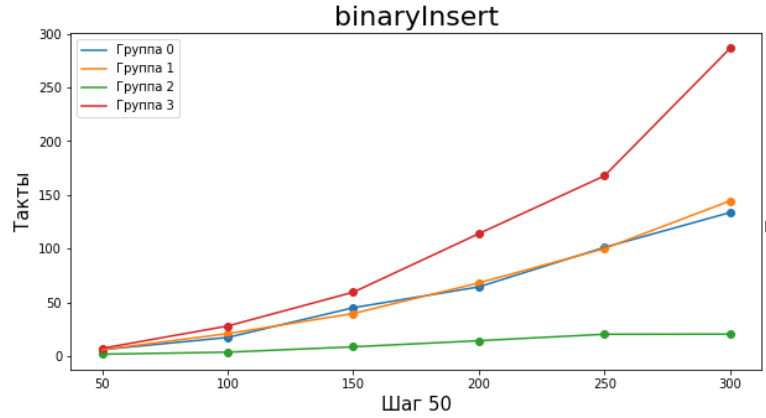
1. На графике 0_1000, на котором изображена работа сортировок для массива, заполненного случайными числами от 0 до 5, мы можем видеть, что все сортировки, кроме bubble, binaryInsert и selection показали себя очень хорошо при большом размере массива. Bubble снова худший.
2. На графике 1_1000, на котором изображена работа сортировок для массива, заполненного случайными числами от 0 до 1000, мы можем видеть ту же самую картину, но при этом график bubble резко взлетает после размера массива 1100. Selection и binaryInsert отрабатывают не так хорошо, как остальные, но при этом держатся более менее стабильно.
3. На графике 2_1000, на котором изображена работа сортировок для почти отсортированного массива, мы видим, что все сортировки, кроме bubble и selection отработали почти одинаково хорошо, немного выделяется только heap sort.
4. На графике 3_1000, на котором изображена работа сортировок для отсортированного в обратном порядке массива, мы видим, что merge, heap, radix, counting, quick sort отработали хорошо. Selection и binaryInsert шли очень близко, но на размере массива 4100 selection sort все-таки выиграл по скорости. Bubble показывает абсолютный антирекорд – более 120млн тактов.

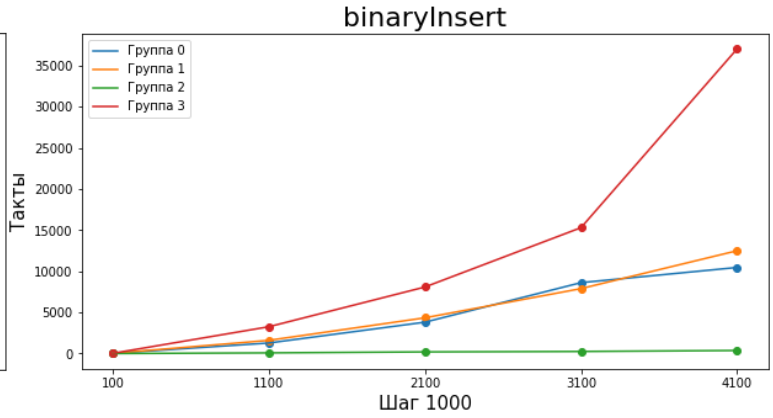
Далее давайте посмотрим на сравнение работы на разных группах массивов каждой из сортировок:



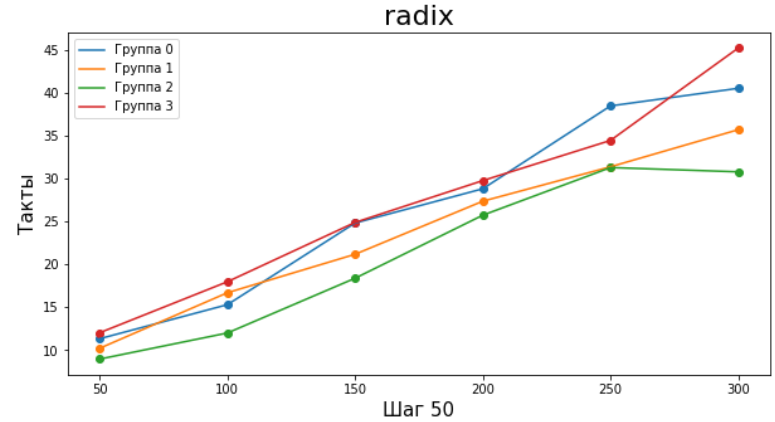


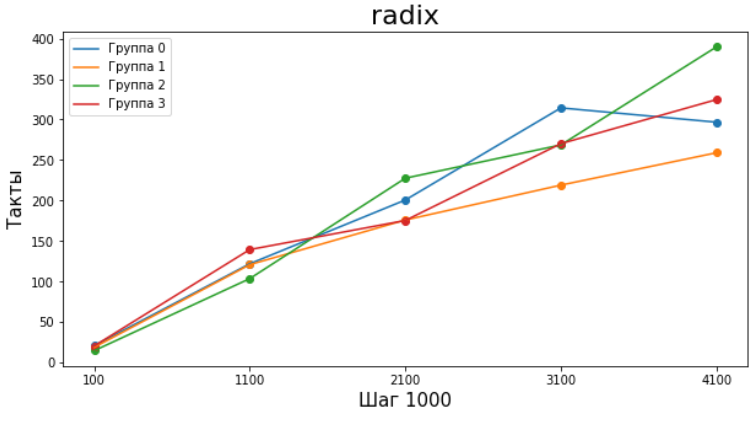
Начнем с bubble sort. На этих графиках мы видим, что bubble лучше всего работает с почти отсортированным массивом и на массивах с небольшой длинной. Все-таки асимптотика bubble sort n^2, и худшим случаем для этой сортировки является отсортированный в обратном порядке массив длинной 4100, на котором bubble поставил антирекорд по скорости.



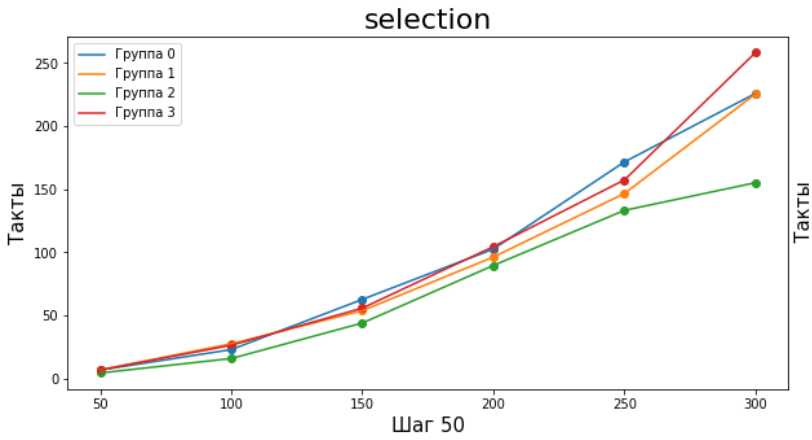


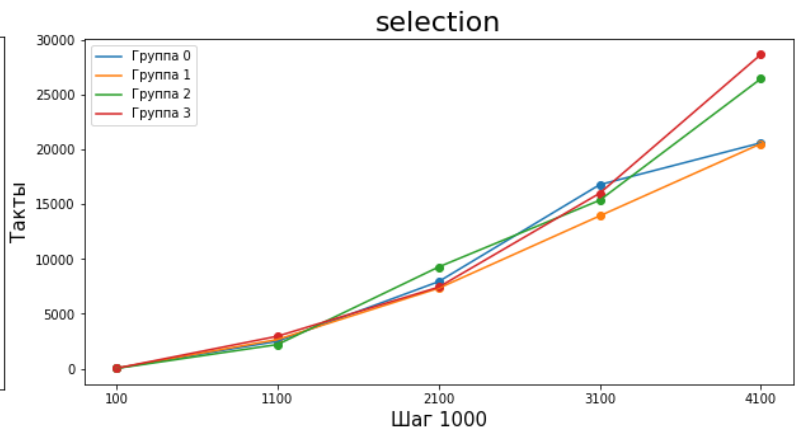
Binary insertion sort так же лучше всего отработал на почти отсортированном массиве, причем очень стабильно, независимо от размера массивов. Так же для массивов, заполненных случайными числами от 0 до 5 и от 0 до 4000 binary insertion показал очень близкие результаты.



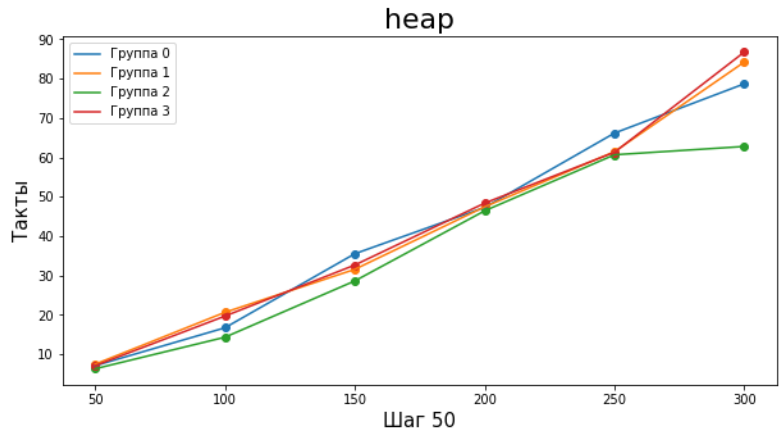


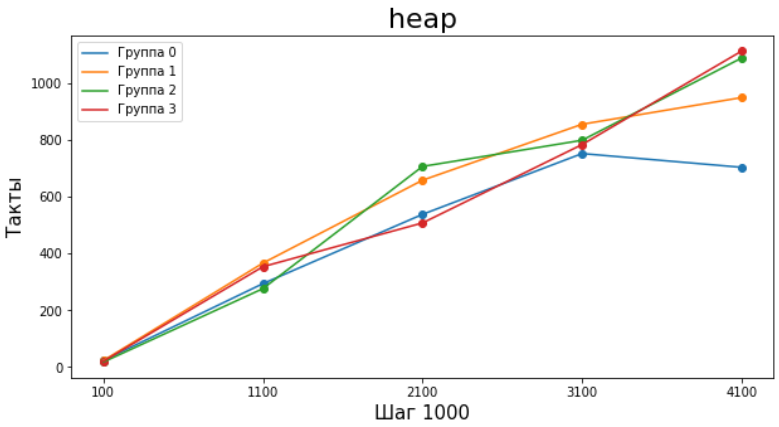
Radix sort оказалась одной из самых стабильных сортировок. Для нее лучше всего подошли небольшие массивы с почти отсортированным содержимым. Примечательно, что те же массивы, но с большим размером оказались худшим случаем для radix. На массиве длинной 4100 radix лучше справился со случайным содержимым от 0 до 4000.



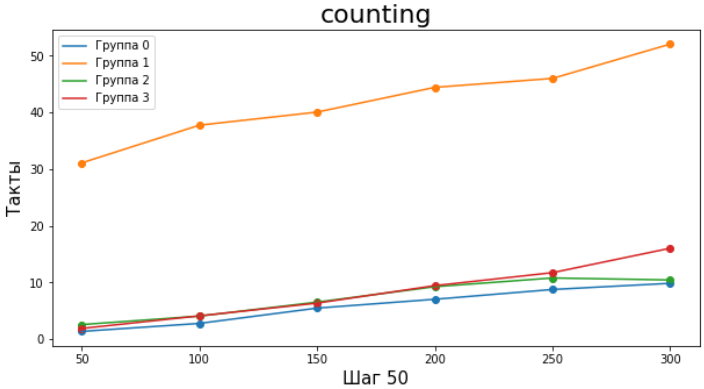


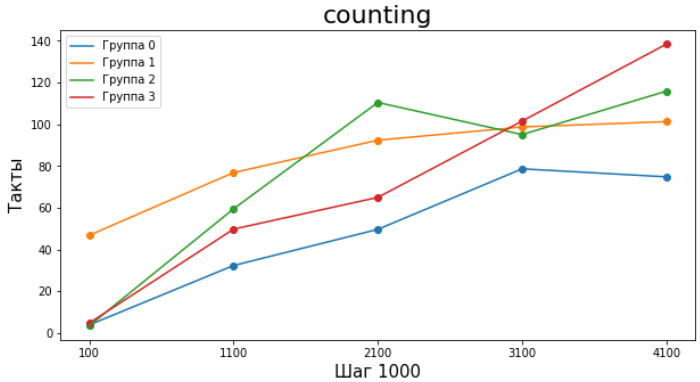
Selection sort тоже оказалась достаточно стабильной в показанных результатах сортировкой. Лучшим случаем для нее является почти отсортированный короткий массив. А худший случай – отсортированный в обратном порядке массив длинной 4100



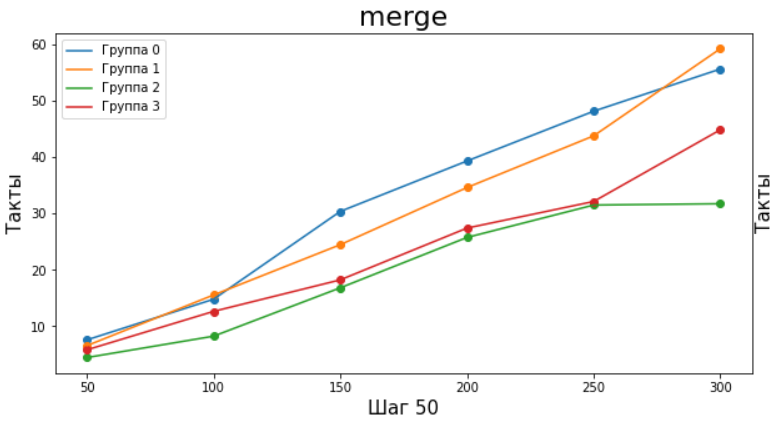


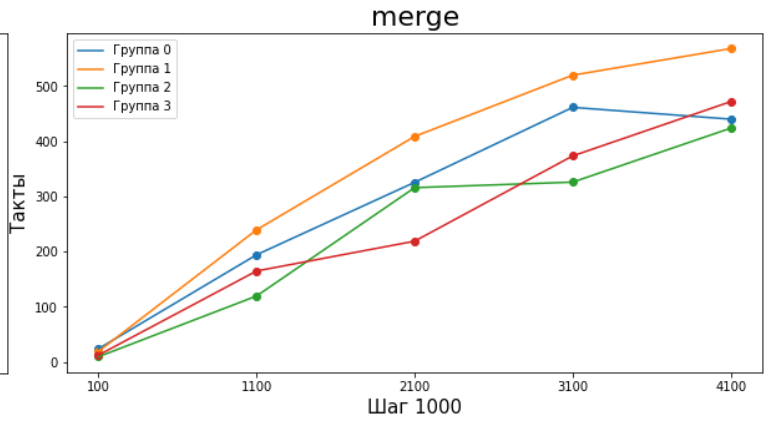
Heap sort показал очень близкие значения во всех группах для короткий массивов. Худшим случаем для данной сортировки является отсортированный в обратном порядке массив большОй длинны, при этом результат почти отсортированного массива той же длинны оказался очень близок к предыдущему.



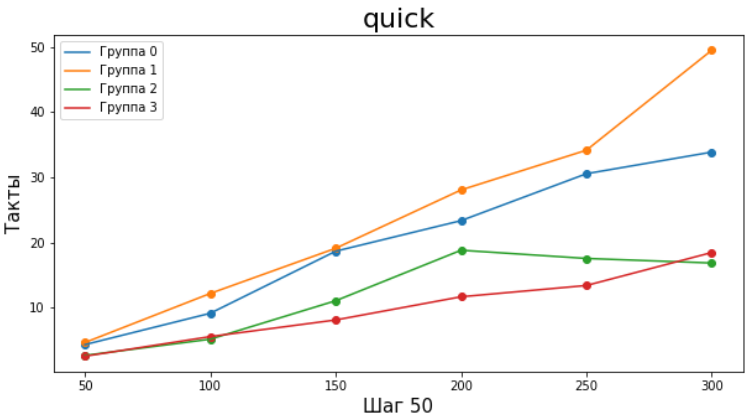


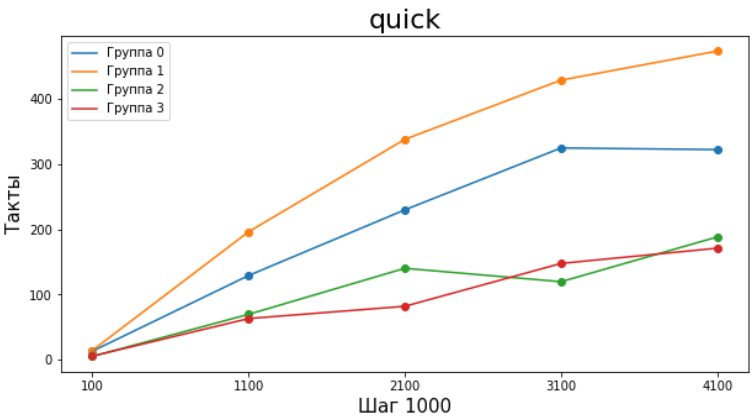
Результаты counting sort оказались достаточно интересными. Практически на всех графиках из первой части отчета мы видели, что данная сортировка была одной из лучших. Counting sort – прекрасный выбор для массивов небольшой длинны почти, при этом, также и для маленьких чисел. Для больших чисел radix sort оказался немного быстрее. Если говорить о больших размерах массивов, то здесь counting sort оказался рекордсменом. И все же на таких размерах лучше всего ему даются массивы со случайными числами от 0 до 5, а хуже всего отсортированные в обратном порядке массивы.





Merge sort лучше всего себя показал на почти отсортированном массиве небольшой длинны, а хуже всего на массив, заполненном случайными числами от 0 до 4000 длинной 4100. При этом увеличение размера массива влияет на график практически равномерно.





Quick sort оказалась достаточно quick на массивах с небольшим размером. Отсортированный в обратном порядке массив и массив, почти отсортированный показали себя достаточно хорошо на массивах малой длинны. Если говорить о массивах с большой длинной, то результаты quick sort достаточно похожи на radix sort и являются одними из лучших среди представленных сортировок. На больших размерах массивов в тех же группах, что и раннее, получились лучшие результаты.