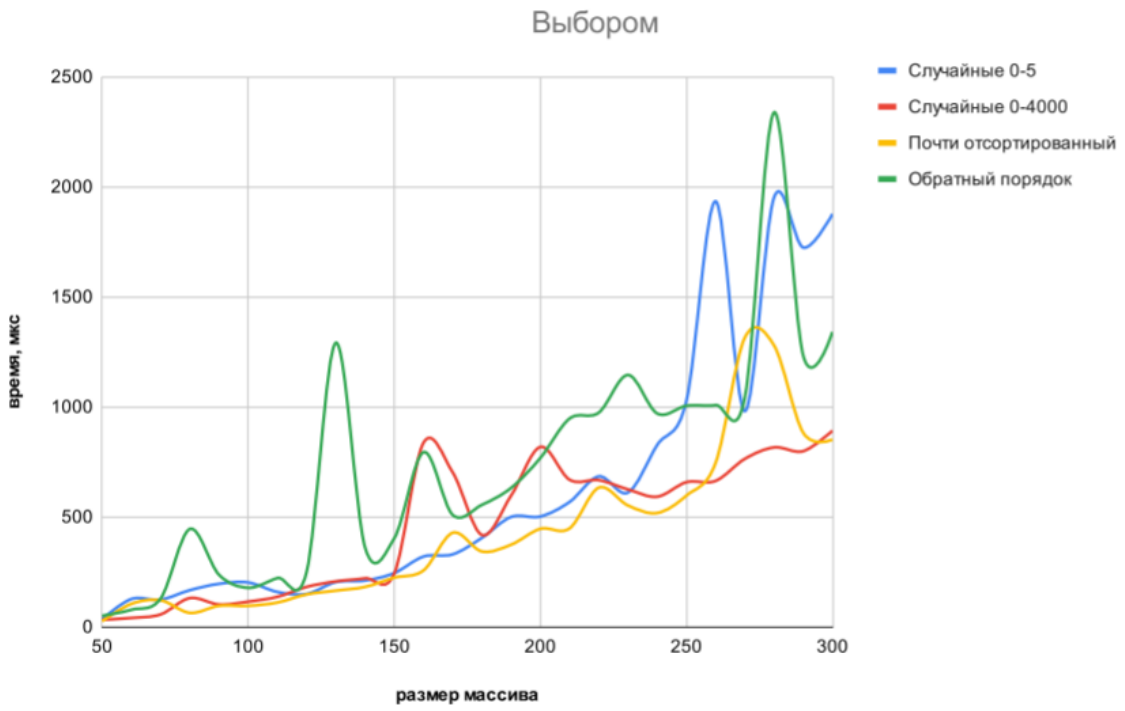
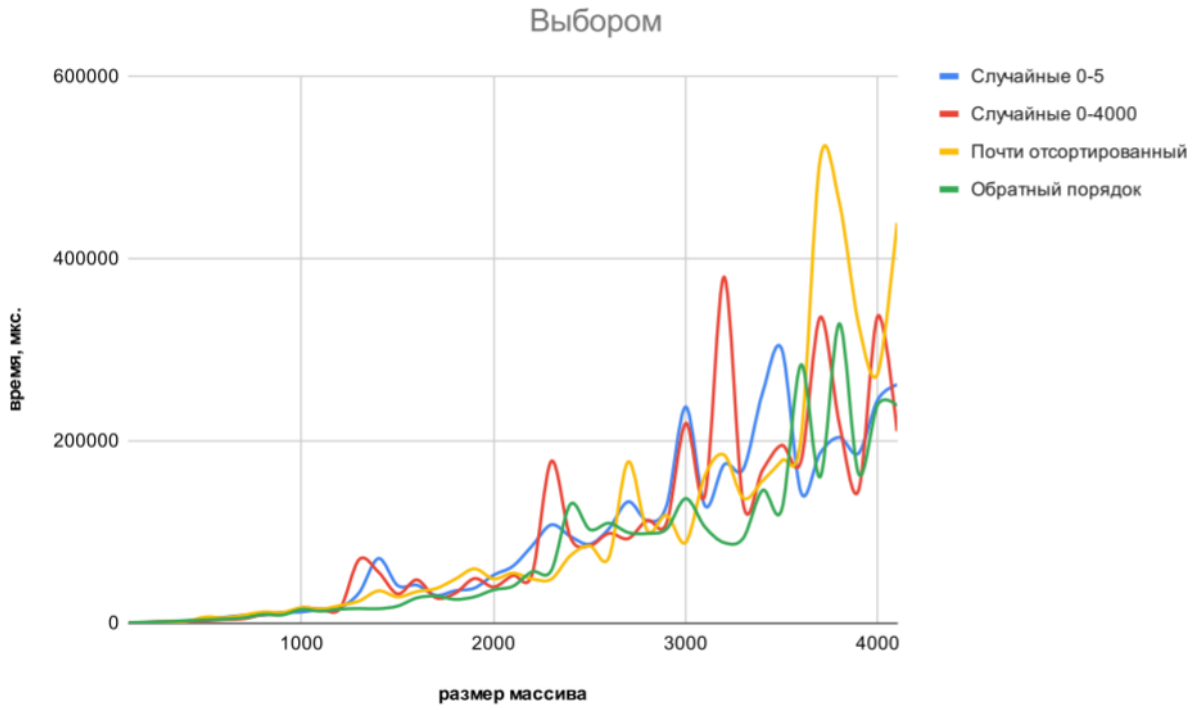
Отчет. Обрубов Илья. БПИ208

	Худший случай	Средний случай	Лучший случай	Доп. Память
Выбором	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(1)$
Пузырьком	$O(n^2)$	$O(n^2)$	$O(n)$	$O(1)$
Пузырьком с условием Айверсона 1	$O(n^2)$	$O(n^2)$	$O(n)$	$O(1)$
Пузырьком с условием Айверсона 1+2	$O(n^2)$	$O(n^2)$	$O(n)$	$O(1)$
Простыми вставками	$O(n^2)$	$O(n^2)$	$O(n)$	$O(1)$
Бинарными вставками	$O(n^2)$	$O(n^2)$	$O(n)$	$O(1)$
Подсчетом (устойчивая)	$O(n+k)$	$O(n+k)$	$O(n+k)$	$O(k)$
Цифровая	$O(nk)$	$O(nk)$	$O(nk)$	$O(n+k)$
Слиянием	$O(nlogn)$	$O(nlogn)$	$O(nlogn)$	$O(n)$
Быстрая + Хоара	O(n^2)	$O(nlogn)$	$O(nlogn)$	$O(n)$
Быстрая + Ломуто	O(n^2)	$O(nlogn)$	$O(nlogn)$	$O(n)$
Пирамидальная	$O(nlogn)$	$O(nlogn)$	$O(nlogn)$	$O(1)$

Выбором.


Выбором
2500
2000
1500
1000
500
0
время, мкс
50
100
150
200
250
300
размер массива
Случайные 0-5
Случайные 0-4000
Почти отсортированный
Обратный порядок


Выбором
600000
400000
200000
0
время, мкс.
1000
2000
3000
4000
размер массива
Случайные 0-5
Случайные 0-4000
Почти отсортированный
Обратный порядок

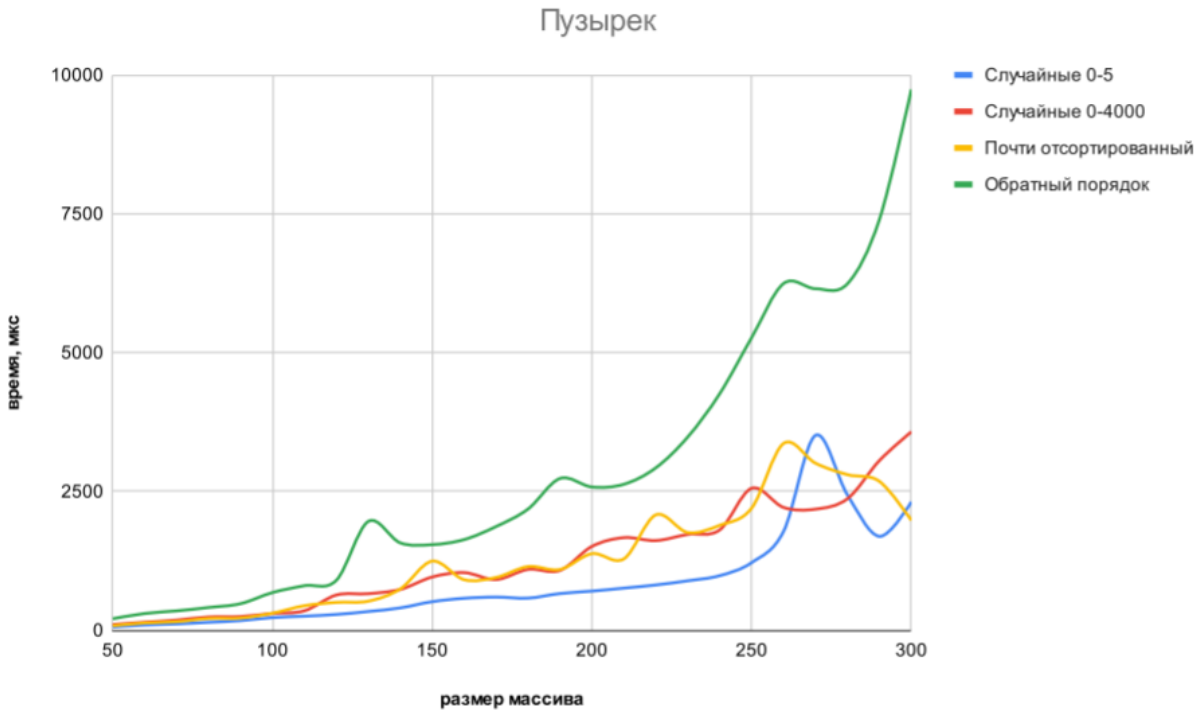
Сортировка выбором - итеративная сортировка, которая в лучшем, худшем и среднем случае показывает ассимптотическое время $O(n^2)$.

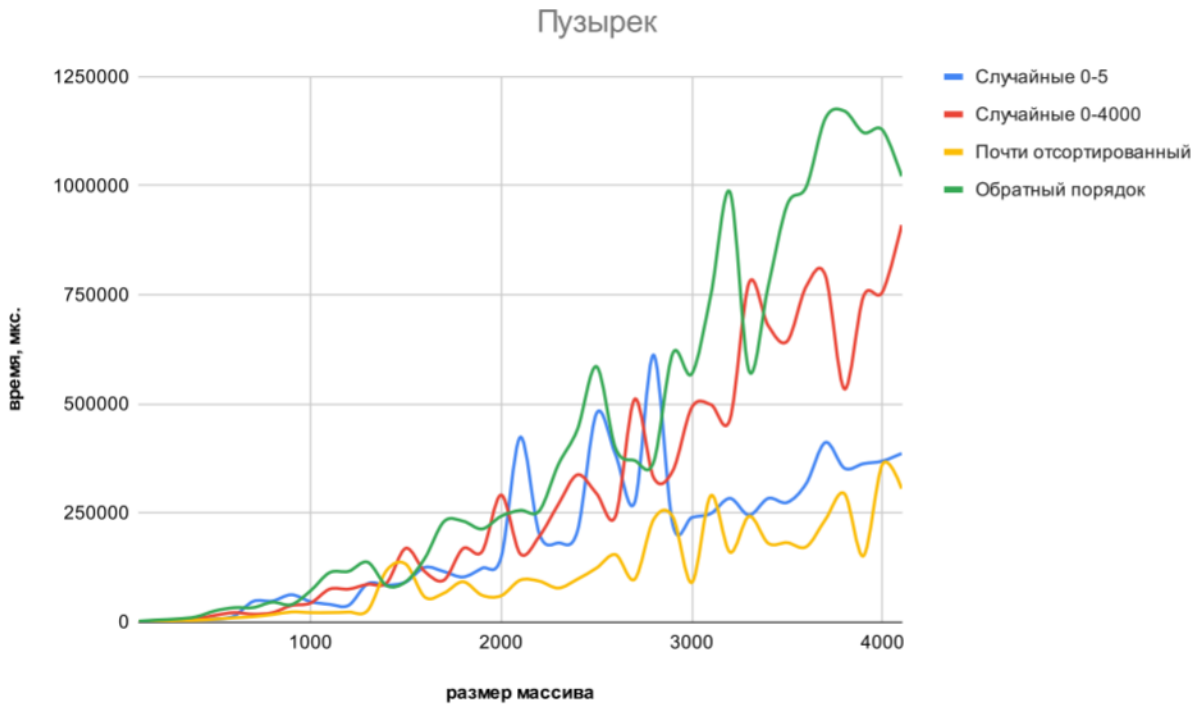
Это можно увидеть и на графиках, итоговое время не сильно отличается друг от друга и возрастает квадратично как на маленьком количестве данных [50; 300], так и на больших [100, 4100].

Вид данных в массиве не оказывает большого эффекта на результирующее время.

То есть, так как алгоритму приходится производить много обменов и поисков минимума неотсортированной части - это не зависит от разброса и сортированности данных.

Пузырьком





Пузырьковая сортировка - также итеративная сортировка, хоторая требует кучу сравнений элементов и проходов по массиву с пошаговыми перестановками элементов.

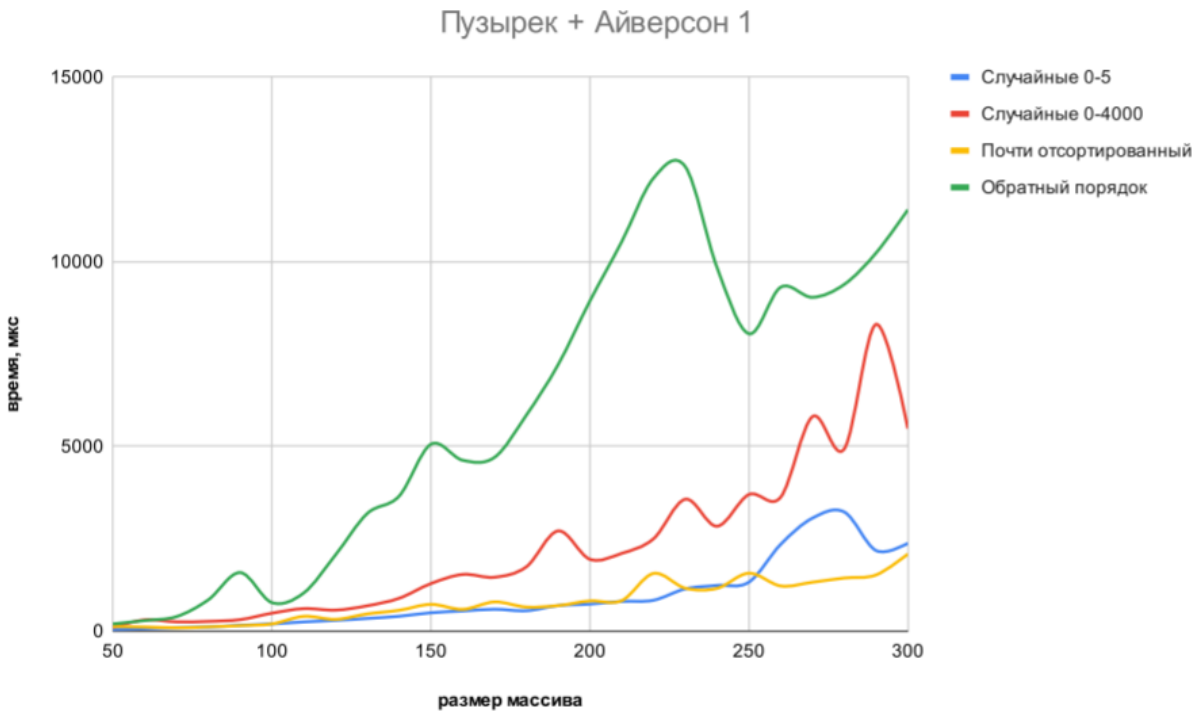
Ассимптотическая сложность - $O(n^2)$, что особенно видно если взглянуть на графики, где массив отсортирован в обратном порядке, что является худшим случаем для сортировки пузырьком, время исполнения вырастает в разы, в сравнеии с другими видами массивов.

Отсортированный массив является лучшим случаем для “пузырька”, сложность остается $O(n)$. Поэтому почти отсортированный массив показывает лучшее время, чем остальные.

Если сравнить графики случайных чисел в диапазоне [0; 5] и [0; 4000], массив меньшего разброса показывает лучшее время на больших данных, так как гораздо чаще будут встречаться элементы не требующие перестановки.

Так как время увеличивается квадратически, алгоритм хоть как то эффективен только на массиве меньшего размера.

Пузырьком с условием Айверсона 1



Условие Айверсона 1 означает следующее: “Если после выполнения внутреннего цикла не произошло ни одного обмена, то массив уже отсортирован, и продолжать что-то делать бессмысленно.”.

О никак не улучшает ассимптотическую сложность алгоритма, а также ассимптотику лучшего, среднего и худшего случая, но уменьшает количество проходов внешнего цикла.

Эффект от этой оптимизации нельзя заметить на массиве отсортированном в обратном порядке, так как это остается худшим случаем и время примерно тоже что и без нее. Но можно увидеть улучшение времени на почти отсортированном массиве при большом размере массива, так как в какой то момент сравнения заканчиваются раньше чем в алгоритме без оптимизации.

Остальные выводы совпадают с предыдущими рассуждениями об алгоритме “пузырьком”.

Пузырьком с условием Айверсона 1+2

К алгоритму с первым условием Айверсона можно добавить еще одну оптимизацию, т.е. условие Айверсона 2, которое говорит о том, что если использовать идекс последнего обмена на предыдущей итерации внешнего цикла и использовать его как верхнюю границу просмотра массива, то это уменьшит количество проходов внутреннего цикла, так как алгоритм закончится когда индекс станет равен 0.

Также никак не улучшает ассимптотическую сложность и не меняет крайние случаи, но позволяет выигрывать во времени на некоторых массивах.

Но можно заметить, что массив отсортированный в обратном порядке на малом количестве данных сортируется быстрее, используя условие Айверсона 2, улучшение с 11000 мкс. до 3000 мкс. Также менее значительные улучшения есть и на других массивах.

Но глядя на разницу между графиками с условием Айверсона 2 и без него на массиве большого размера улучшения по времени обратно отсортированного массива практически нет и время увеличивается все также квадратически. Наоборот, если обратить внимание на почти отсортированный массив у этих графиков, алгоритм с двумя оптимизациями, в сравнении с одной, улучшает

работу с 126000 до 58000 мкс, в сравнении с алгоритмом без оптимизаций так вообще, с 305000 мкс. до 58000 мкс.

Эти же улучшения по времени заметны на массиве разброса [100; 4000] и большого размера, где без второго условия время составляет 740000 мкс., а используя вторую оптимизацию 339000 мкс. Улучшение в сравении с сортировкой без использования условий Айверсона аналогично больше.

То есть можно заметить что использование двух оптимизаций Айверсона гораздо лучше чем без них и примерно в два раза лучше чем только с одной на большом количестве данных.

Это достигается за счет уменьшения количества проходов внутреннего цикла, т.е. лишних сравнений.

Простыми вставками

Сортировка простыми вставками - алгоритм с тем же квадратичным временем, но с ассимптотикой $O(n)$ в лучшем случае, где массив отсортирован и обменов не производится.

Эта ассимптотика особенно заметна если взглянуть на график с массивом большого размера, где время работы алгоритма с почти отсортированным массивом не превышает 9000 мкс, так как обменов производится гораздо меньше.

В то время как работа сортировки на других видах массивов квадратична и требует множества самих вставок и поиска необходимого индекса в массиве.

На небольшом количестве данных сортировка вставками работает быстрее сортировки пузырьком более чем в 3 раза. Если посмотреть на массив со случайными числами от 0 до 5 время значительно сокращается, так как поиск о котором говорилось выше произваодится быстрее так как дисперсия меньше.

Обратно отсортированный массив - худший случай для алгоритма, поэтому и имеет наиболее худшее время, например, при большом размере массива.

Бинарными вставками


Бинарными вставками
2500
2000
1500
1000
500
0
50
100
150
200
250
300
время, мкс
размер массива
Случайные 0-5
Случайные 0-4000
Почти отсортированный
Обратный порядок


Бинарными вставками
600000
400000
200000
0
1000
2000
3000
4000
время, мкс.
размер массива
Случайные 0-5
Случайные 0-4000
Почти отсортированный
Обратный порядок

Сортировка бинарными вставками является тем же алгоритмом сортировки вставками, но за тем исключением, что используется не линейный поиск, а бинарный. Ассимптотика та же, лучший, худший и средний случаи не меняются.

Более эффективен, в сравнении с простыми вставками, только на массиве большого размера, но как видно не всегда. Так как почти отсортированный массив все еще близок к лучшему случаю, время работы практически то же.

Как мы видим на массиве небольшого размера выйгрыша по времени, в сравнении с простыми вставками нет, а на данных с небольшим разбросом время чуть хуже. На обратно отсортированном массиве большого размера, аналогично время не улучшается, а время работы при остальных массивах примерно одинаково с простыми и бинарными вставками.

Время работы на других массивах различается то в пользу алгоритма с простыми вставками, то с использованием бинарных вставок. Поэтому анализировать их и сравнивать их достаточно сложно. И бинарные вставки не являются ультимативной оптимизацией алгоритма сортировки простыми вставками.

Подсчетом


Подсчетом
1250
1000
750
500
250
0
50
100
150
200
250
300
время, мкс
размер массива
Случайные 0-5
Случайные 0-4000
Почти отсортированный
Обратный порядок


Подсчетом
5000
4000
3000
2000
1000
0
1000
2000
3000
4000
время, мкс.
размер массива
Случайные 0-5
Случайные 0-4000
Почти отсортированный
Обратный порядок

Сортировка подсчетом - алгоритм сортировки в котором используется диапазон чисел для подсчета совпадающих элеметов.

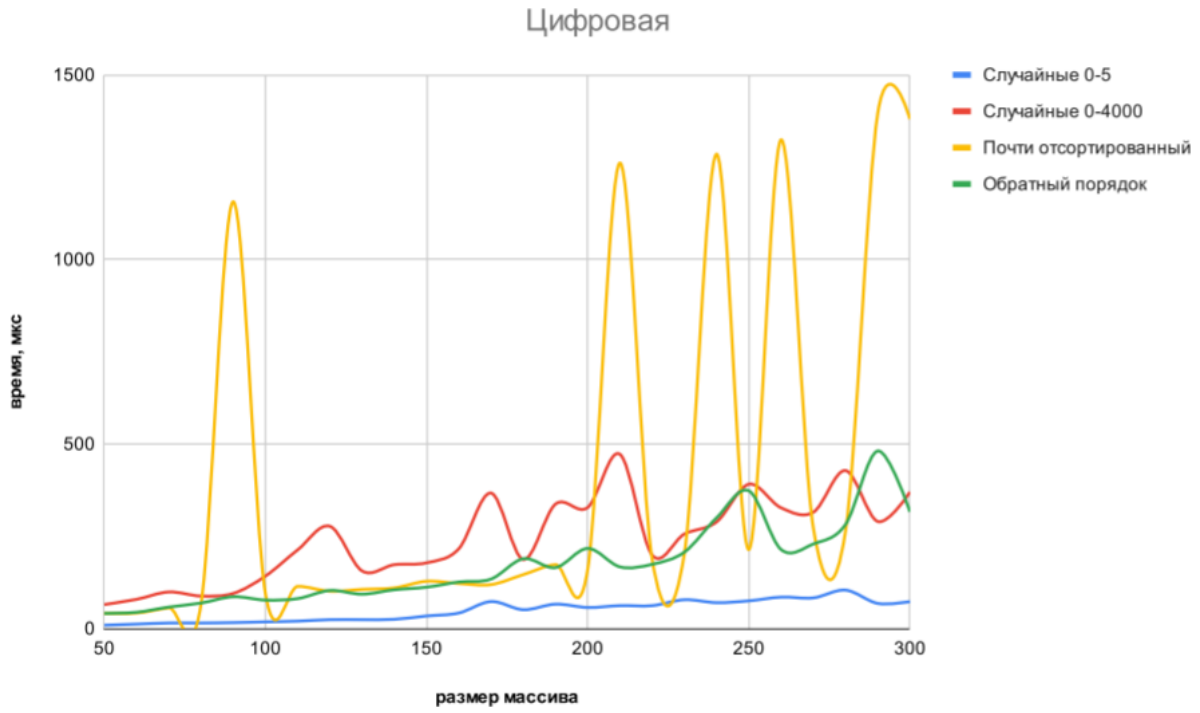
Очевидно эффективен когда разброс этих чисел не большой и чем меньше тем лучше. Это заметно на графиках, время работы на массиве случайных чисел от 0 до 5 выигрывает у массива с большим разбросом чисел, вне зависимости от размера массивов.

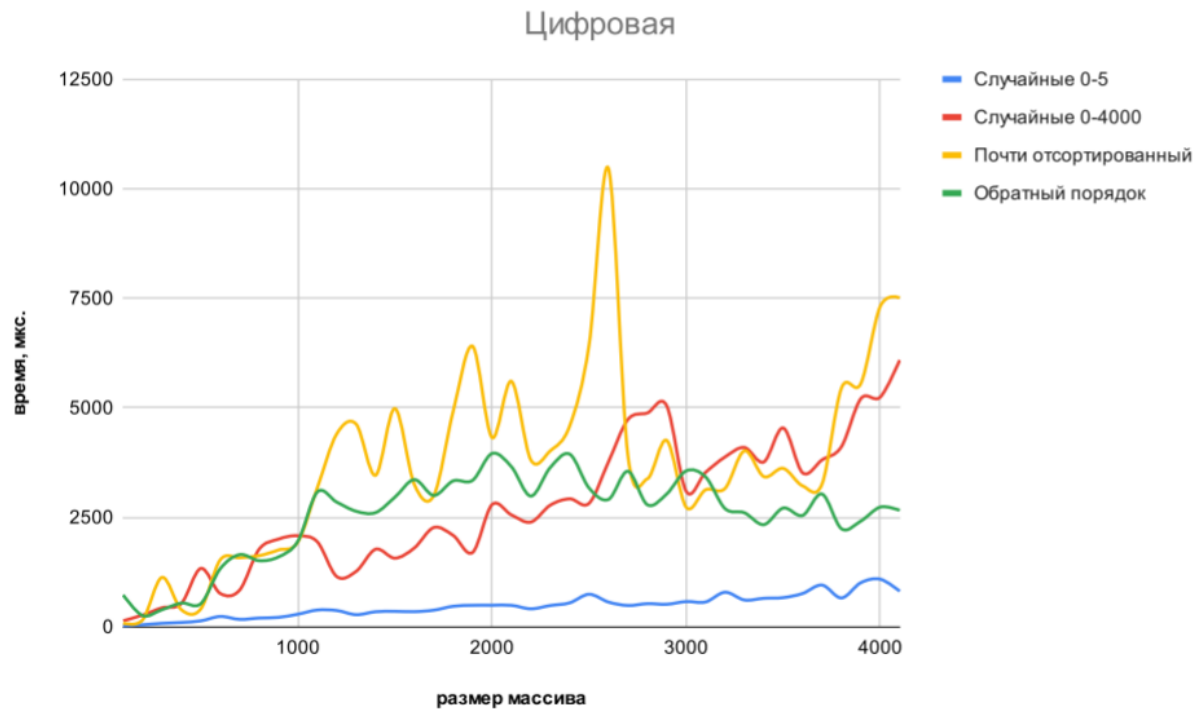
Так как даже в худшем случае ассимптотика не скатывается в квадратную, а лишь зависит от разброса чисел в массиве мы видим, что время работы не превышает 5000 мкс.

Также стоит отметить, что при уже отсортированном массиве время зависит исключительо от разбросачисел , что видно на графике с почти отсортированным массивом, разброс которого практически равен 4100 и на большом размере массива алгоритм имеет худшее время, в сравнении с остальными массивами.

Интересно отметить, что обратно отсортированный массив на больших данных работает также быстро как на массиве с небольшим разбросом, видимо это связано с устойчивостью сортировки.

Цифровая


Цифровая
1500
1000
500
0
время, мкс
50
100
150
200
250
300
размер массива
Случайные 0-5
Случайные 0-4000
Почти отсортированный
Обратный порядок


Цифровая
12500
10000
7500
5000
2500
0
время, мкс.
1000
2000
3000
4000
размер массива
Случайные 0-5
Случайные 0-4000
Почти отсортированный
Обратный порядок

Цифровая сортировка основывается на сортировке подсчетом и использовании для нее разрядов чисел, где сами числа представляются как последовательности разрядов в некоторой системе счисления.

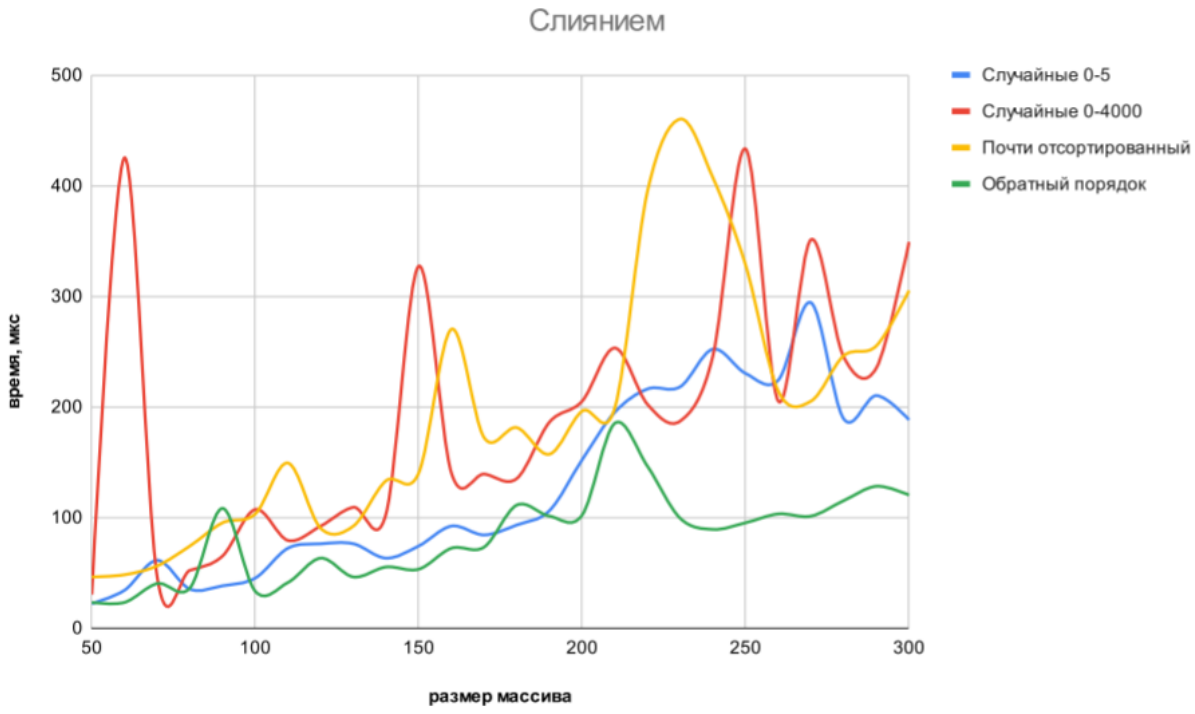
Ассимптотическая сложность $O(nk)$. То есть аналогично с сортировкой подсчетом чем меньше разброс чисел в массиве, тем эффективнее алгоритм и время работы зависит только от этого разброса.

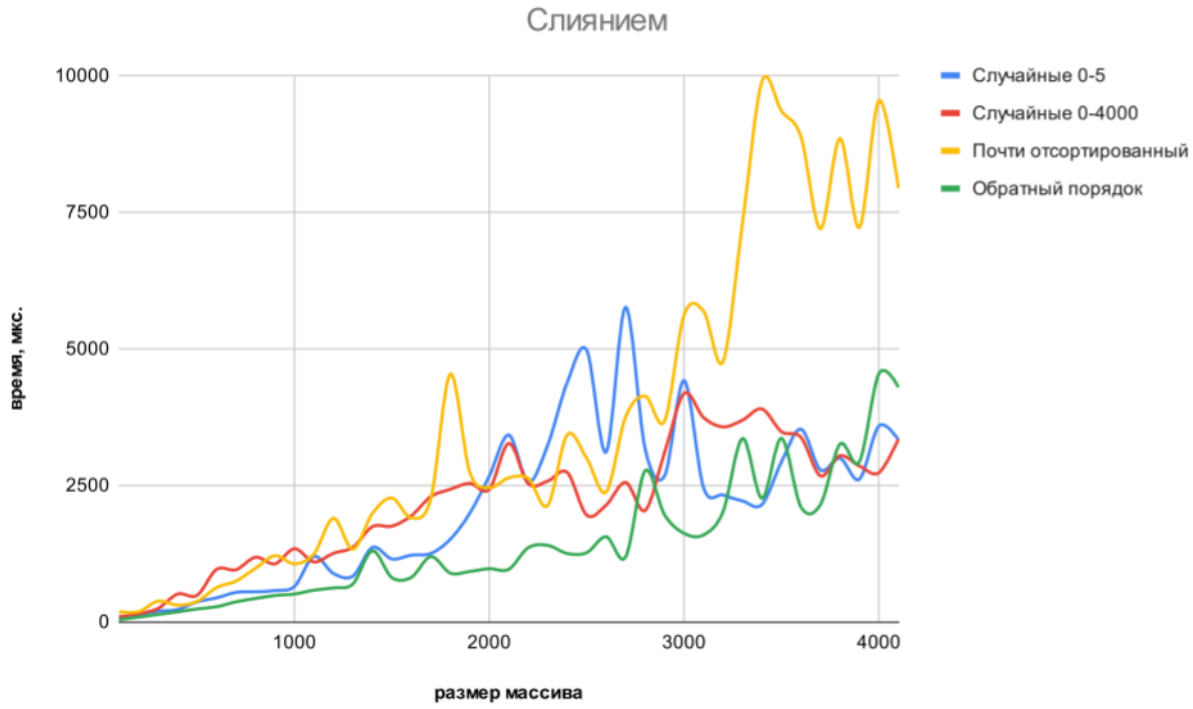
Что видно и на графиках, на случайных числах от 0 до 5 сортировка достаточно эффективна, а на почти отсортированном массиве и на содержащем числа от 0 до 4000 время значительно увеличивается как а массиве большого размера, так и на небольшом.

Также можно отметить, что на время работы алгоритма влияет и выбор системы счисления, в данном случае это десятеричная система. В зависимости от нее последовательность разрядов будет меняться.

В прочем рассуждения схожи с предыдущей сортировкой.

Слиянием





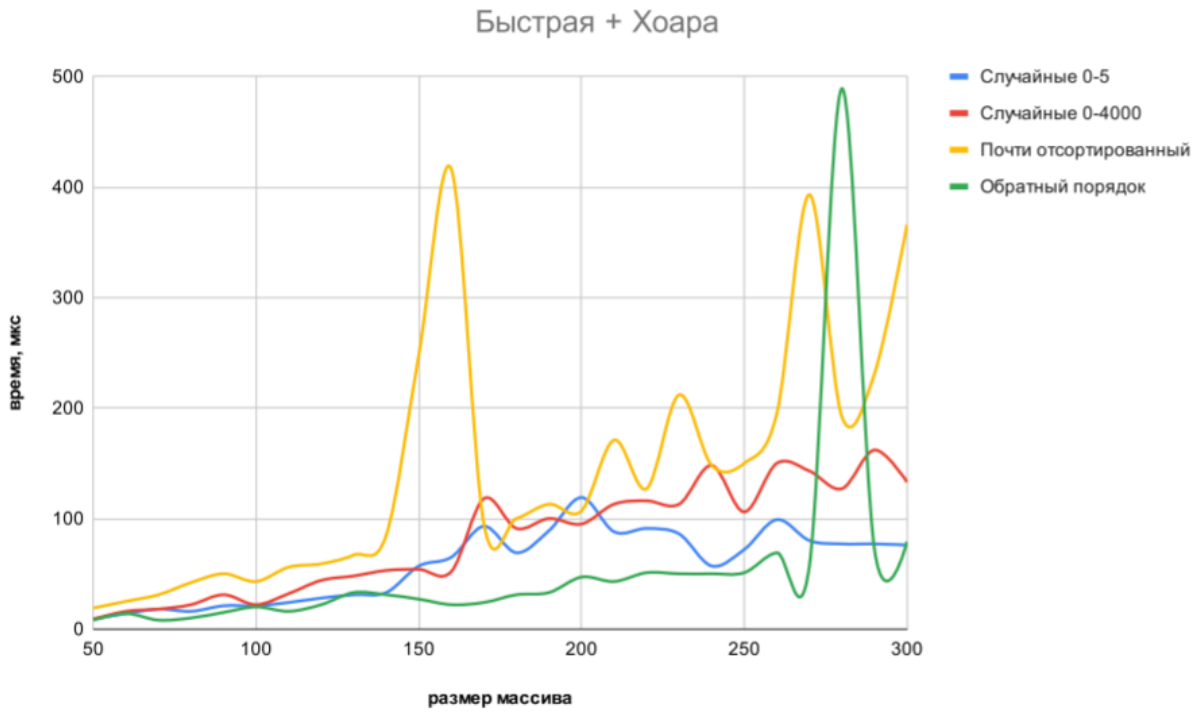
Сортировка слиянием осованна на принципе разделяй и властвуй, т.е. рекурсивном разделении массива на две части, которые сортируются отдельно, а затем склеиваются. Требует дополнительной памяти, но во всех крайних случаях ассимптотика $O(nlogn)$.

Это заметно и на графиках. Например, на небольших данных обратно отсортированный массив сортируется быстрее остальных, которые работают примерно одинаково по времени.

Если смотреть на график, где массив размера от 100 до 4100, почти отсортированный массив не позволяет алгоритму работать быстрее, а наоборот, хуже, чем остальные массивы.

То есть можно заметить что в целом на любых можно ожидать время $O(nlogn)$, что и отображают графики, но нужо помнить что требует достаточного количества дополнительной памяти, что не делает алгоритм универсальным.

Быстрая + Хоара


Быстрая + Хоара
500
400
300
200
100
0
50
100
150
200
250
300
время, мкс
размер массива
Случайные 0-5
Случайные 0-4000
Почти отсортированный
Обратный порядок


Быстрая + Хоара
4000
3000
2000
1000
0
1000
2000
3000
4000
время, мкс.
размер массива
Случайные 0-5
Случайные 0-4000
Почти отсортированный
Обратный порядок

Быстрая сортировка с разбиением Хоара заключается в том, что есть опорный элемент на середине массива и рекурсивно сортируются элементы слева от него, которые меньше либо равны опорному и справа, большие или равные опорому элементу.

В худшем случае работает за $O(n^2)$, при котором опорный элемент всегда максимум или минимум, что не представлено на графиках.

В лучшем же опорный элемент всегда делит по полам уже отсортированные части. Например, на графиках видно, что алгоритм с обратно отсортированным массивом работает лучше остальных как на больших, так и небольших данных, так как делит данные пополам.

Быстрое время работы также заметно на данных с небольшим разбросом чисел. И чем он больше тем вероятнее, что сортировка будет происходить дольше, что видно на графике при увеличеии количества данных.

Почти отсортированный массив не настолько близок к лучшему случаю, потому как разделение данных непредсказуемо, и работа с этим типом массива не настолько быстра как с предыдущими.

Дополнительная память $O(n)$ только в худшем случае, когда происходит n рекурсивных вызовов, а в среднем $O(logn)$

Быстрая + Ломуто

Рассмотрим ту же быструю сортировку, но теперь с разбиением Ломуто, которое состоит в том, что в качестве опорного элемента используется первый или

последний элемент массива.

Сортировка с этим разбиением может легко скатываться в квадратичную. Например, на больших и небольших данных обрато отсортированный массив будет работать неэффективно, так как опорным элементом будет минимум(или максимум) и разбиение не равномерое. Это худший случай сортировки.

Если обратить внимание на массив большого размера, то видно, получилось так, что почти отсортированный масив с увеличением количества данных также квадратично увеличивается, так как количество рекурсивных вызовов мало только если размер массива небольшой.

Лучше всего себя показывает массив со случайными числами от 0 до 4000, так как опорный элемент может неплохо разделять массив, даже на большом количестве данных. (время работы не 0, а меньше 9000 мкс.)

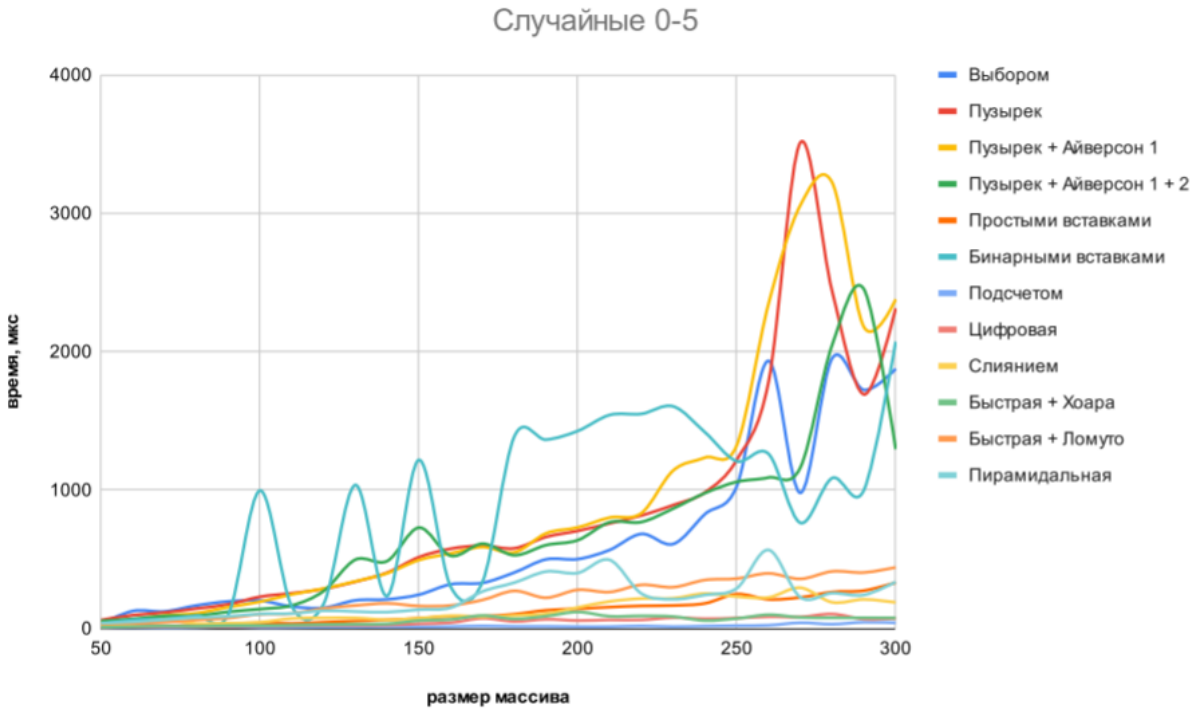
Пирамидальная

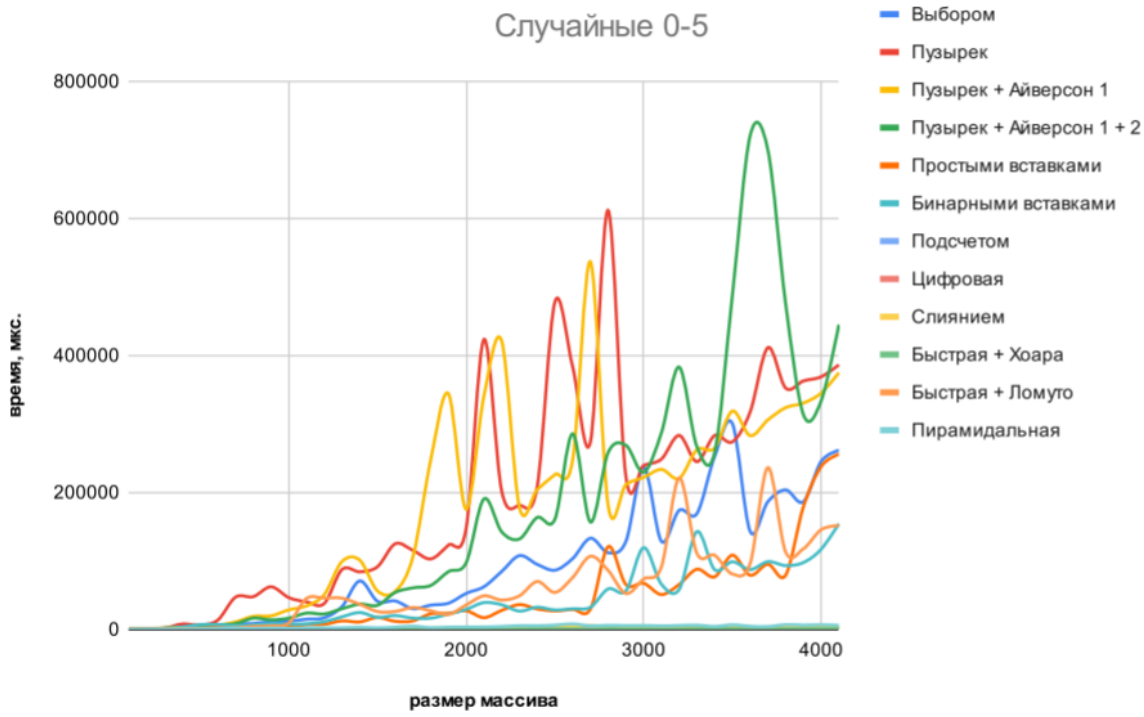
Пирамидальная сортировка использует бинарное сортирующее дерево, где кучи min-heap всплывает, а max-heap тонет.

На почти отсортированных массивах работает столь же долго, как и на хаотических данных, что видно и на графиках, данный вид массива работает наиболее неэффективно в сравнении с остальными.

Так как ассимптотика во всех случаях $O(nlong)$ работа на остальных массивах примерно одинакова, обратно отсортированный массив работет чуть хуже, что связано с построением сортирующего дерева и куч минимума и максимума.

Случайные 0-5





После отдельного рассмотрения всех перечисленных алгоритмов, можно посмотреть отдельно на используемые виды массивов и какой алгоритм лучше

или хуже на нем работает.

Массив со случайными числами от 0 до 5 имеет небольшой разброс, что на руку таким алгоритмам как сортировка подсчетом и цифровая сортировка, они наиболее эффективны именно на таком виде массивов.

Также стоит отметить что быстрая сортировка с разделением Хоара сортирует массив из 4100 элеметов всего за 1100 мкс. Подсчетом за 500 мкс. Цифровая за 800 мкс.

Пирамидальная также наиболее эффективна на данных с небольшим разбросом, чем на других массивах и справляется за 5800 мкс.

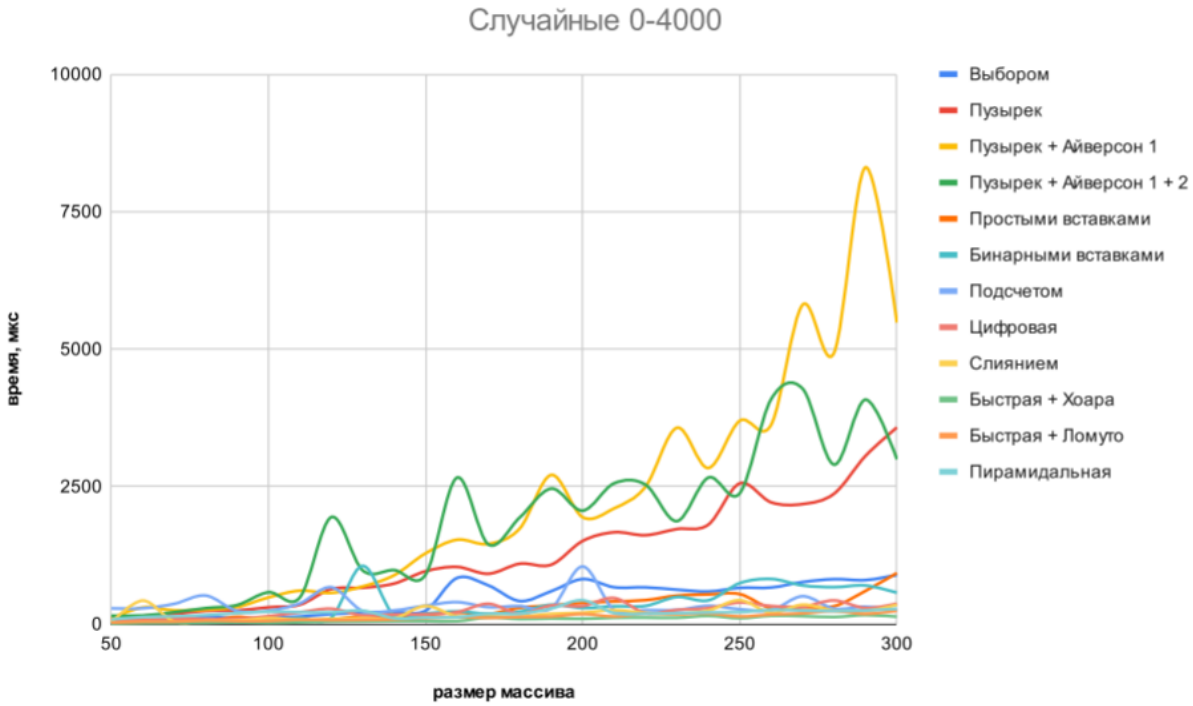
Сортировка вставками работает также как и сортировка выбором за квадратичное время.

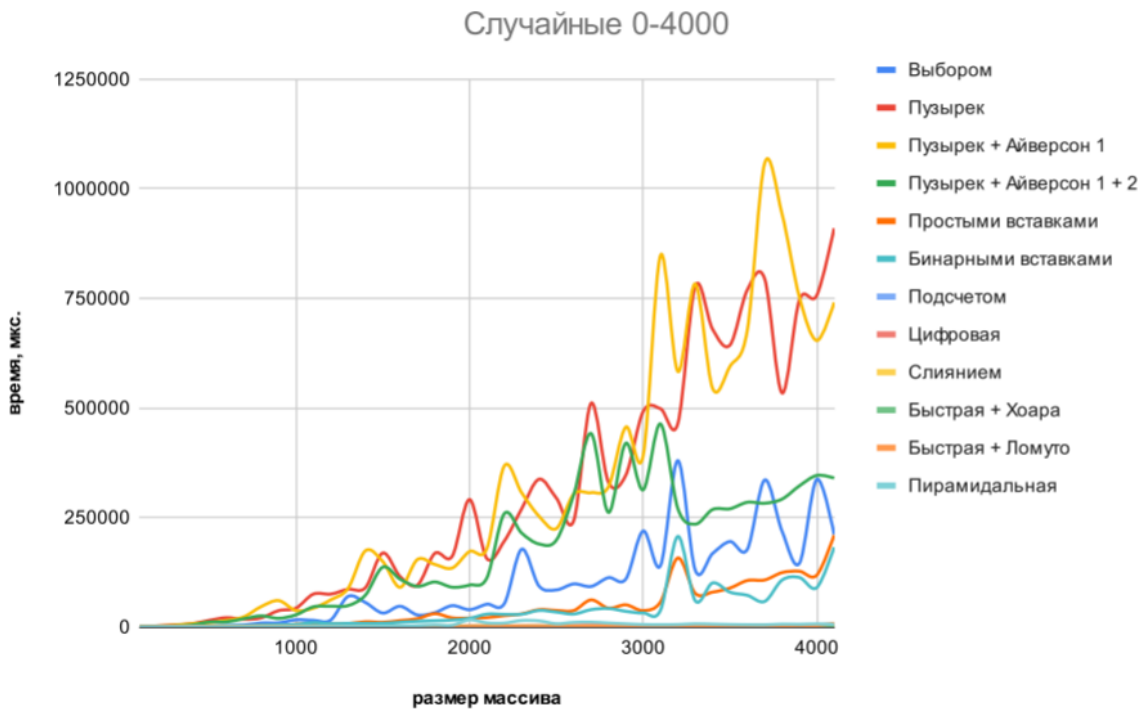
Глядя на графики заметно выделяются квадратичные группы сортировок - пузырьки, на больших данных они работают особенно хуже остальных.

То есть не небольших данных можно явно выделить две группы, одни работают в большинстве своем за квадратичное время, другие более эффективны и время не превышает 500 мкс.

На больших данных 4 группы, которые в целом описаны выше.

Случайные 0-4000





Массив отличающийся большим разбросом чисел.

На этих двух графиках видно, что пузырьковые сортировки работают наиболее неэффективно, но на больших данных сортировка пузырьков с двумя оптимизациями работает гораздо лучше собратьев.

В остальном на массиве размером до 300 сортировки работают неплохо, разве что сортировки вставками и выбором работают не так эффективно, что заметно и на большом массиве.

Стоит отметить, что сортировки подсчетом и цифровая сортировка уже не имеют такого преимущества и работают хуже, что не так заметно на графике большого массива, так как пузырьковые сортировки очень неэффективны.

Почти отсортированный

Далеко не для всех алгоритмов почти отсортированный массив является преимуществом.

На небольших данных все также хуже себя показывают сортировки пузырьком. Но это не настолько относится к большому массиву, где наибольшее неэффективны алгоритмы: быстрая сортировка + Ломуто и сортировка выбором

Лучше всего в данном случае работают: быстрая сортировка с разделением Хоара 1500 мкс. на массиве из 4100 элементов и сортировки вставками, простыми(2600 мкс.) и бинарными(4100 мкс.).

Остальные сортировки работают максимум 7500 мкс., потому как почти отсортированный массив в их случае сортируется с такой же эффективностью как и не настолько отсортированный.

Обратный порядок

Обратно отсортированый массив зачастую является худшим случаем для алгоритмов сортировки, но, например для быстрой сортировки с разделеием

Хоара это наобором неплохо и она работает за 985 мкс. Что наиболее удивительно сортировка подсчетом рабоает за 485 мкс. видимо из за своей устойчивости.

Пузырьковые сортировки работают по преждему хуже остальных, так как это худший случай для них. Как и для быстрой сортировки с разделением Ломуто, которая не так эффективна по сравнению с другими алгоритмами на больших и небольших данных.

Затем по эффективности уступают сортировки вставками и выбором.

Пирамидальная сортировка рабоет на подобных данных с такой же эффективностью как и а хаотических данных.

Вывод

Перечисленные сортировки стоит рассматривать не только с точки зрения ассимптотики и дополнительной памяти, но и глядя на то, какие данные будут сортироваться.

И взависимости от этого выбирать алгоритм.