КДЗ №1

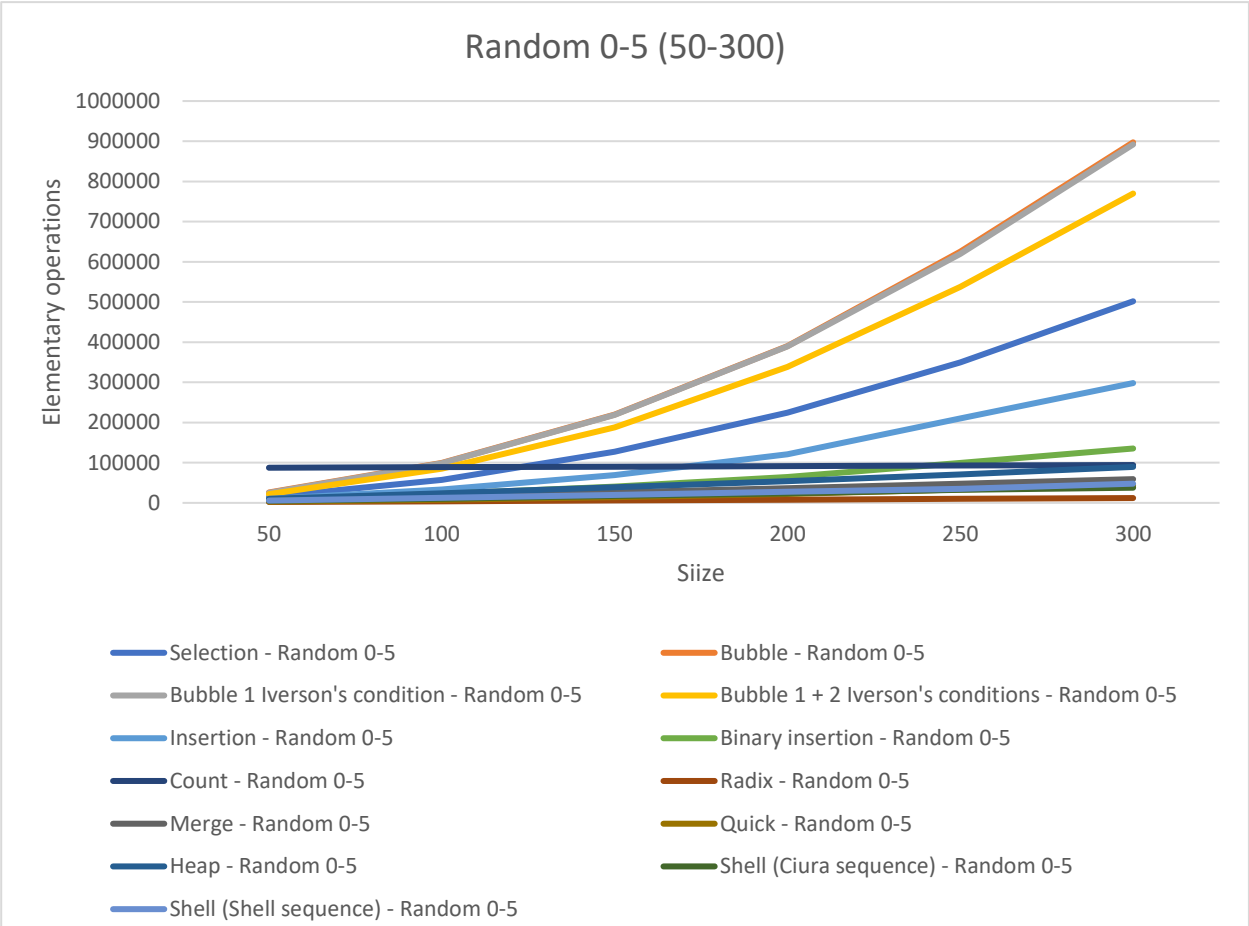
ОТЧЁТ

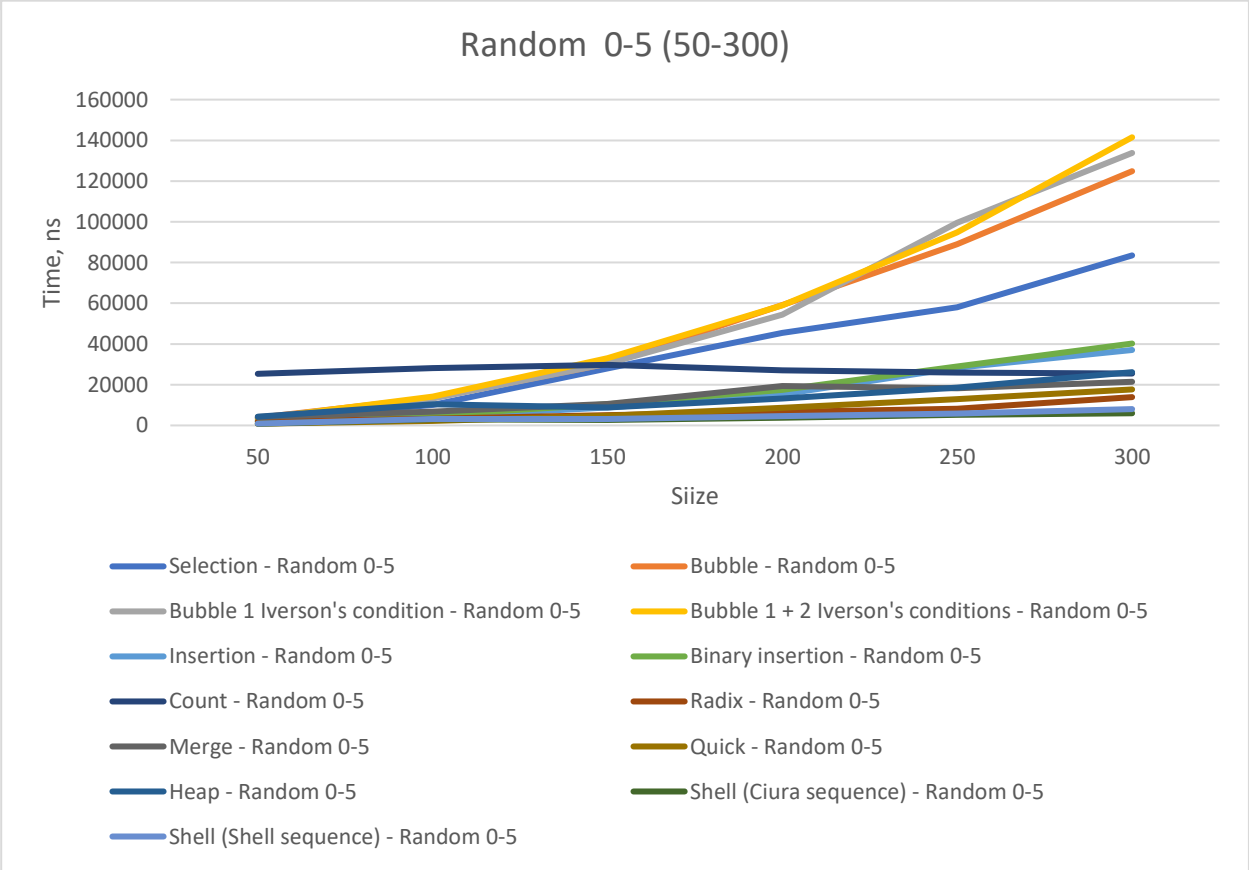
ПО ИССЛЕДОВАНИЮ МЕТОДОВ СОРТИРОВОК

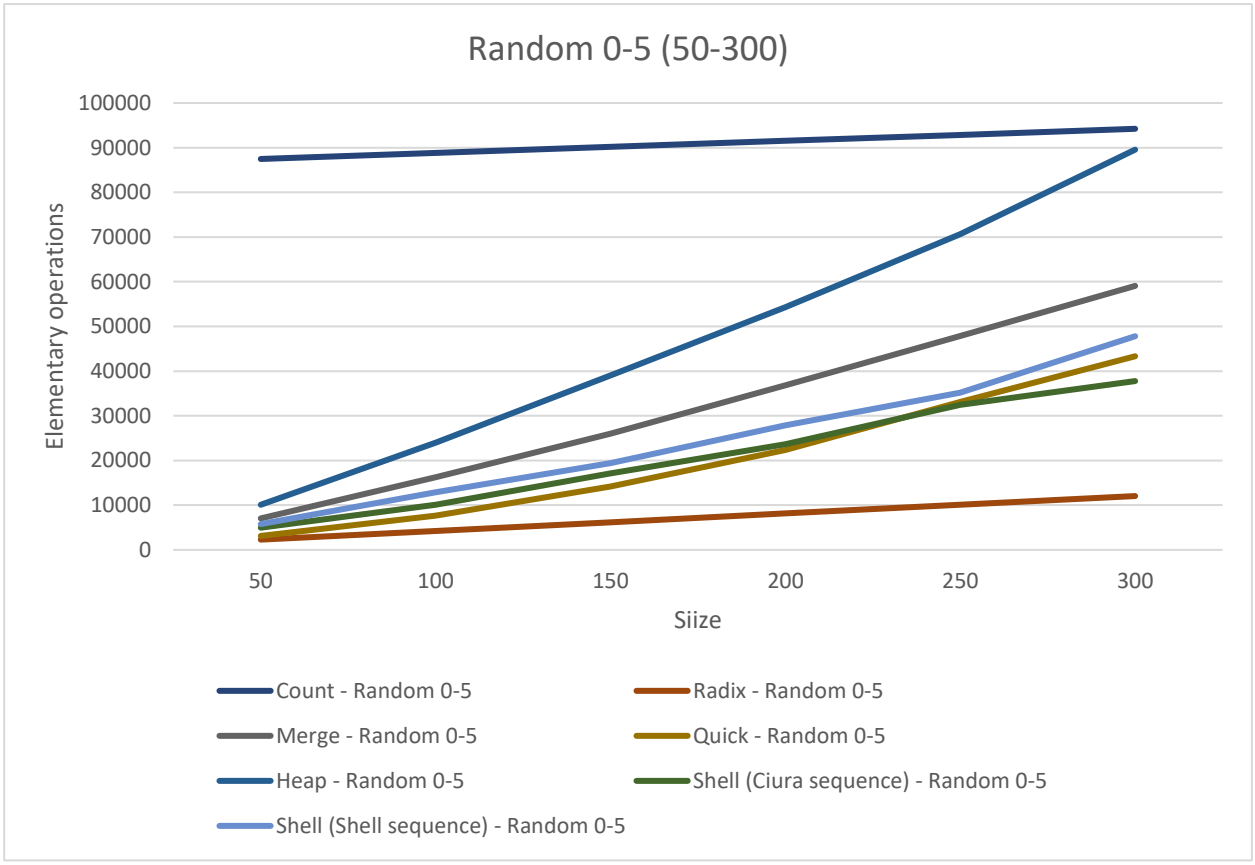
Выполнил:
Студент БПИ219
Гладких Иван

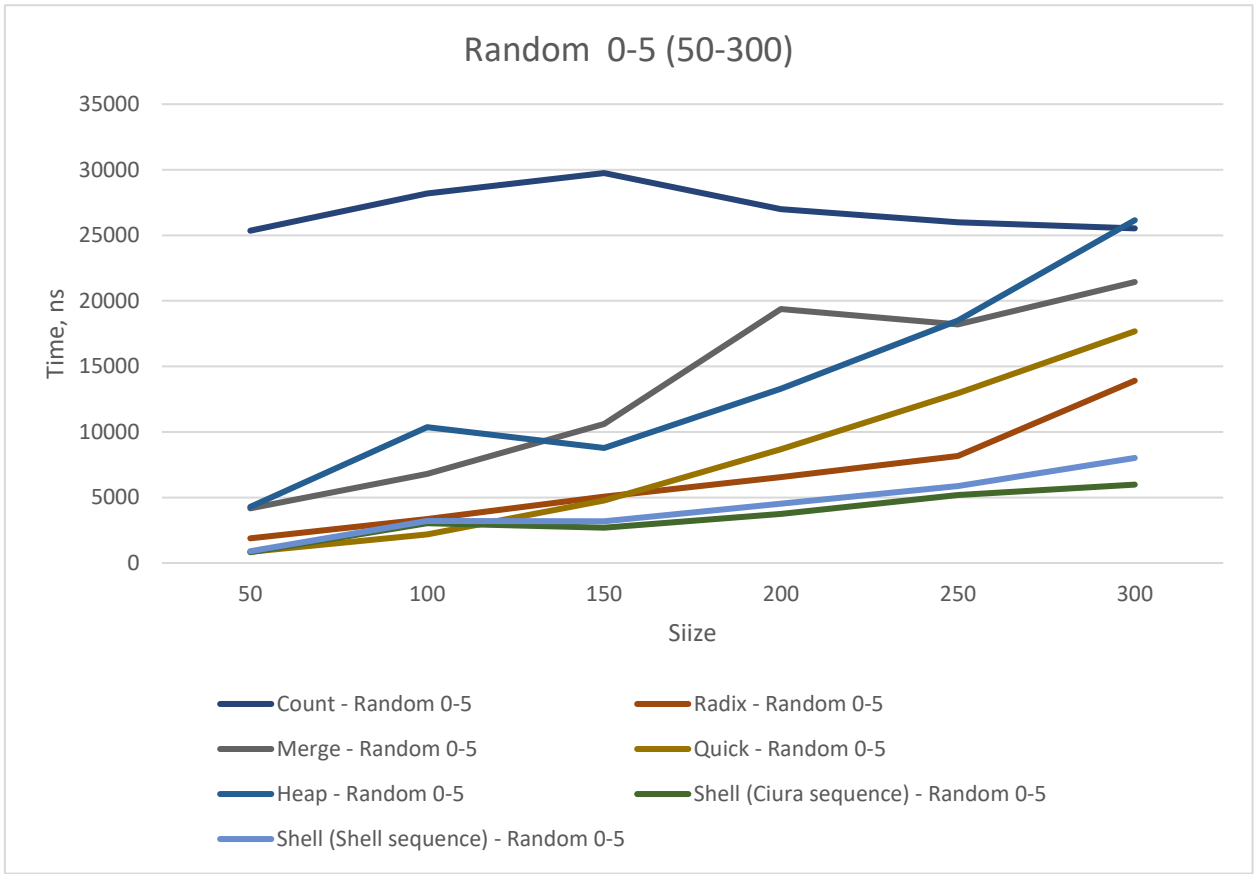
2023

1. Массив размером от 50 до 300, заполненный случайными числами в диапазоне от 0 до 5.






Random 0-5 (50-300)
Elementary operations
100000
90000
80000
70000
60000
50000
40000
30000
20000
10000
0
50
100
150
200
250
300
Siize
Count - Random 0-5
Radix - Random 0-5
Merge - Random 0-5
Quick - Random 0-5
Heap - Random 0-5
Shell (Ciura sequence) - Random 0-5
Shell (Shell sequence) - Random 0-5


Random 0-5 (50-300)
Time, ns
35000
30000
25000
20000
15000
10000
5000
0
50
100
150
200
250
300
Siize
Count - Random 0-5
Radix - Random 0-5
Merge - Random 0-5
Quick - Random 0-5
Heap - Random 0-5
Shell (Ciura sequence) - Random 0-5
Shell (Shell sequence) - Random 0-5

На данном массиве хуже всего себя показывают пузырьковые сортировки, т.к. в среднем случае сложность алгоритма составляет $O(n^2)$ при любой оптимизации. Сортировке с условиями Айверсона 1 и 2 нужно меньше операций, однако по времени все три реализации примерно одинаковые (иногда оптимизированные даже хуже оригинальной).

Затем идет сортировка выбором. Хотя она имеет такую же асимптотическую сложность, сортировка выбором показывает результат лучше, чем пузырьковая, т.к. делает минимальное количество обменов (имеет меньшую константу).

Затем сортировки обычными и бинарными вставками. Сортировка бинарными вставками требует меньше элементарных операций, т.к. находим место для элемента не за O(n), а за O(log n). Но размер массива слишком маленький, константа играет слишком большую роль, поэтому по времени они работают примерно одинаково.

Далее – сортировка подсчётом. Она линейна, но не оптимизирована под маленький диапазон значений, поэтому константа достаточно велика (относительно), из-за чего показывает себя чуть хуже некоторых других сортировок.

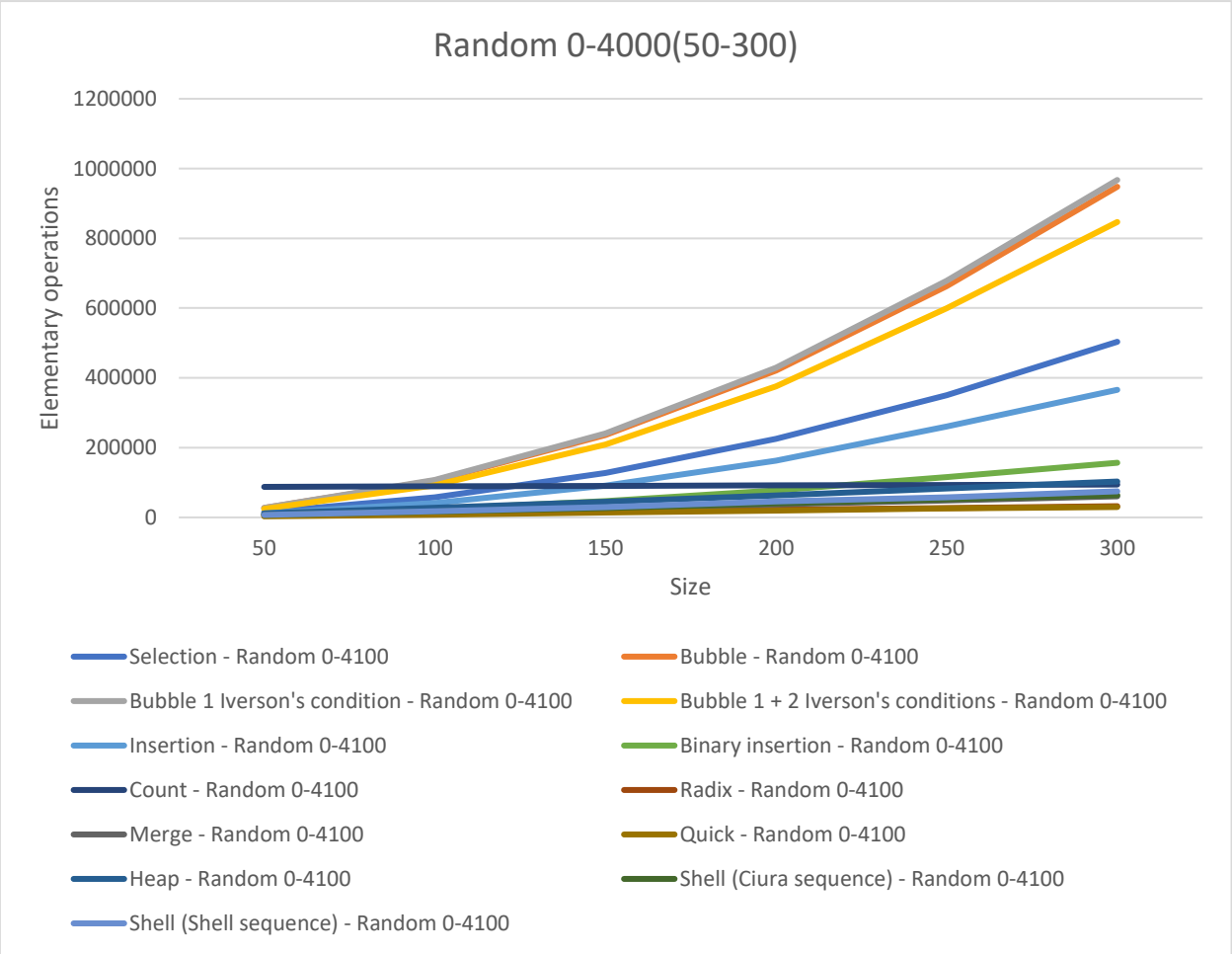
Как по времени, так и по числу элементарных операций дальше идут пирамидальная сортировка и сортировка слиянием. Они имеют одинаковую сложность в худшем и среднем случаях – O(n log n), но у пирамидальной сортировки в среднем выше константа.

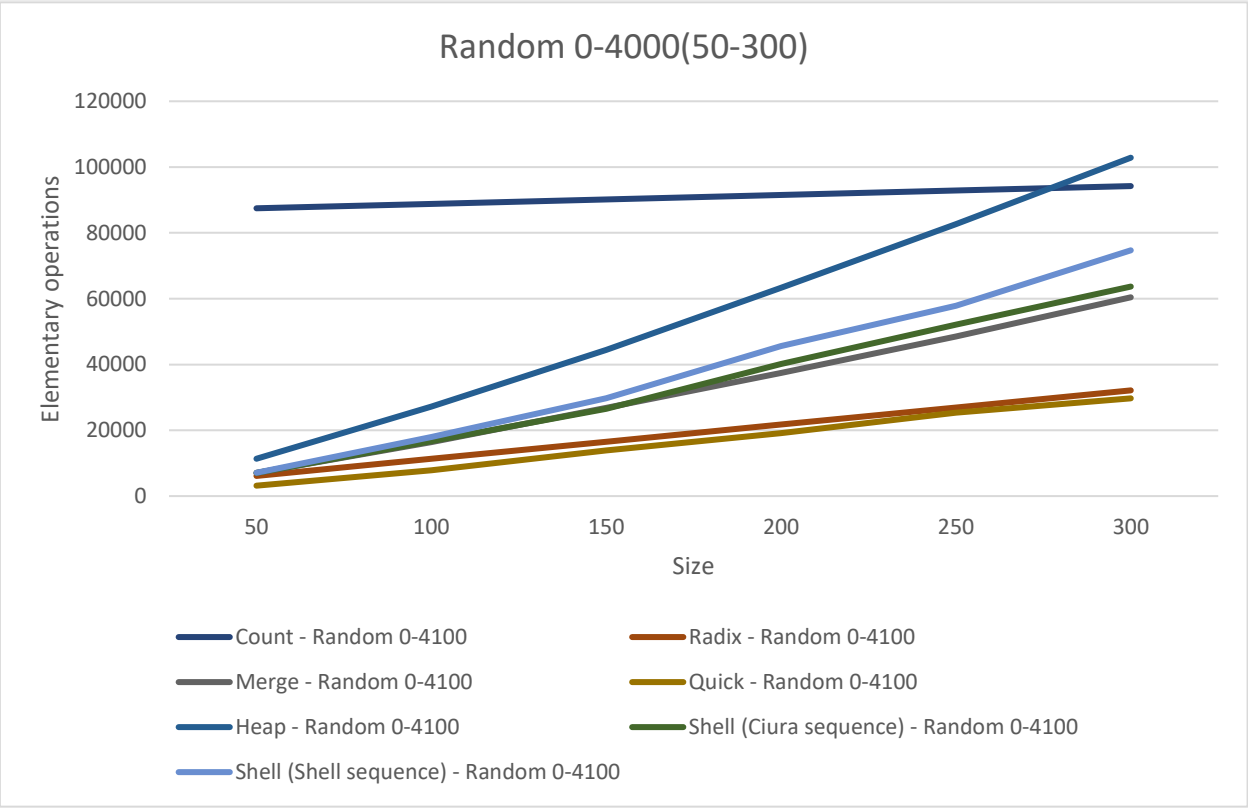
По времени лучше всего себя показывают сортировки Шелла. В среднем случае имеет сложность $O(n^{1,5})$. Причём последовательность Циура показала себя лучше, чем последовательность Шелла. Последовательность Циура была выведена экспериментально и в среднем работает лучше последовательности Шелла (асимптотическая сложность не выведена).

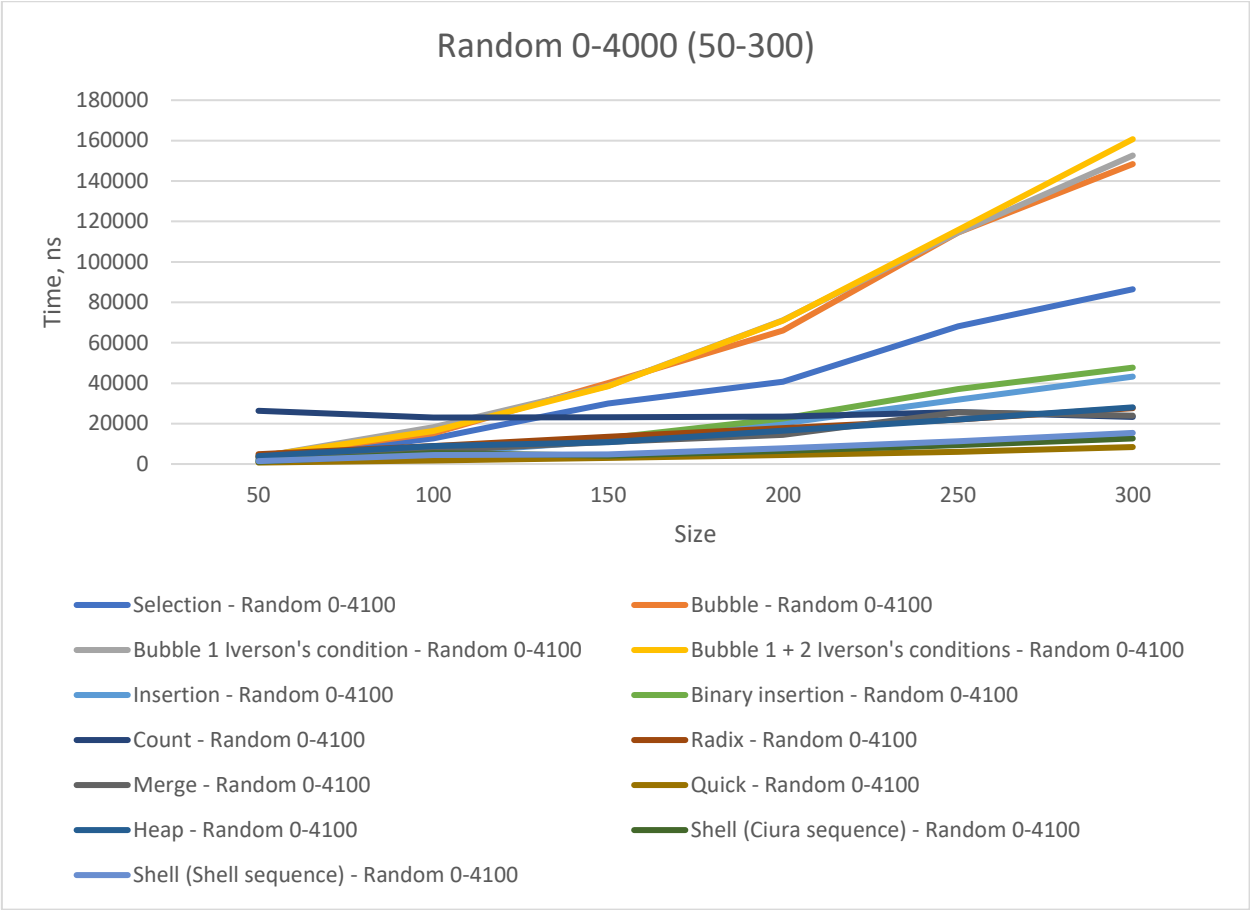
A102549	Unknown (experimentally derived)	1, 4, 10, 23, 57, 132, 301, 701	Unknown	Ciura, 2001[15]

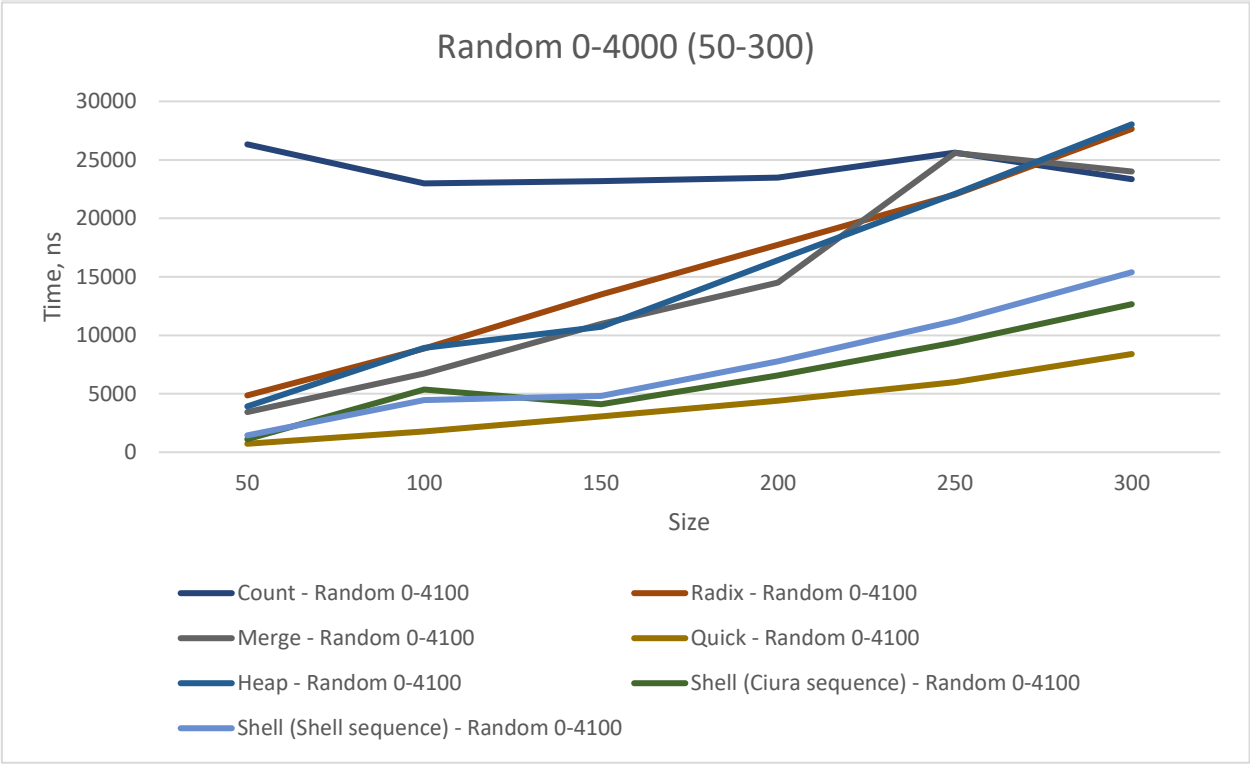
По числу элементарных операций лучший результат показывает цифровая сортировка, т.к. она линейна и зависит от числа разрядов (1 для данного массива). По времени отрабатывает хуже, т.к. неоднократно выделяет и освобождает память.

2. Массив размером от 50 до 300, заполненный случайными числами в диапазоне от 0 до 4000.





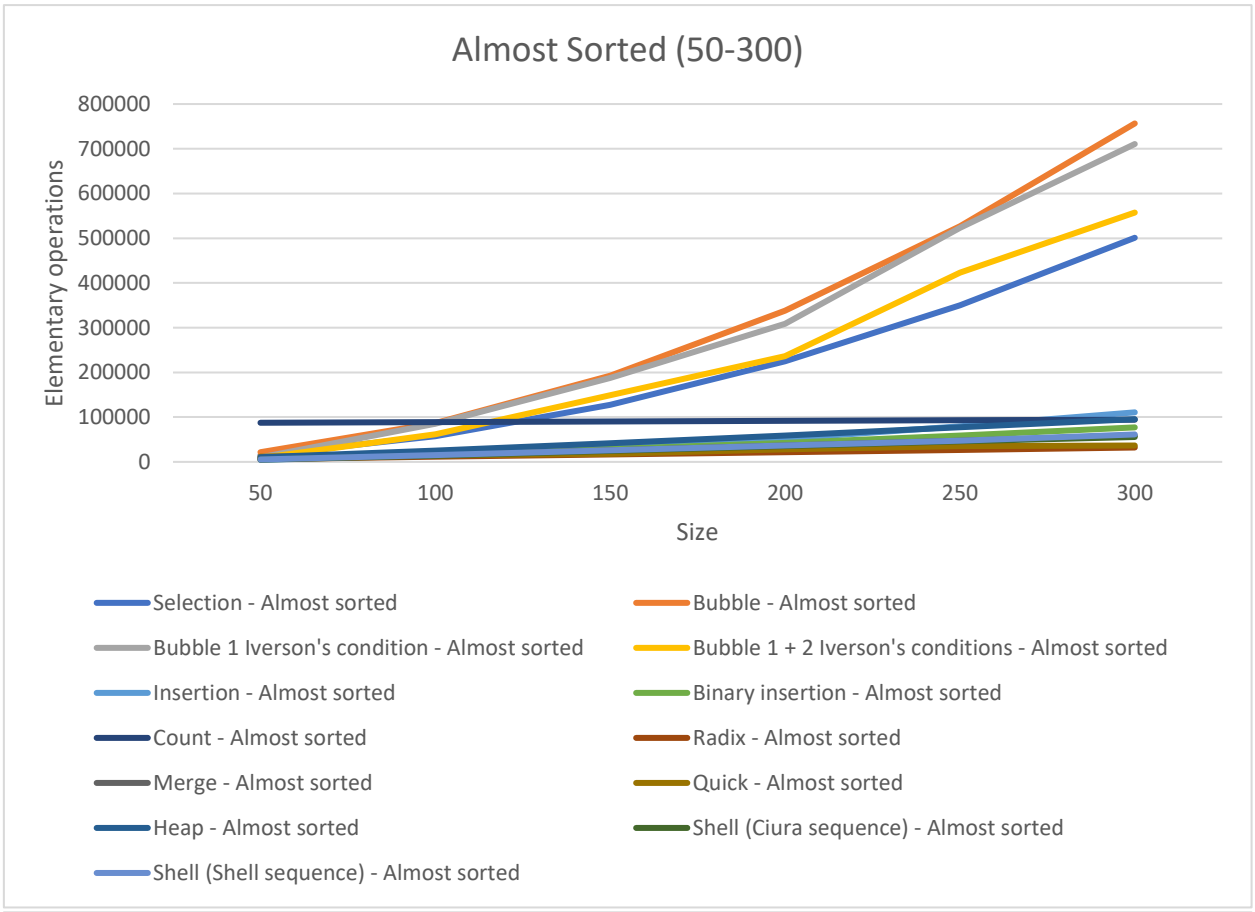


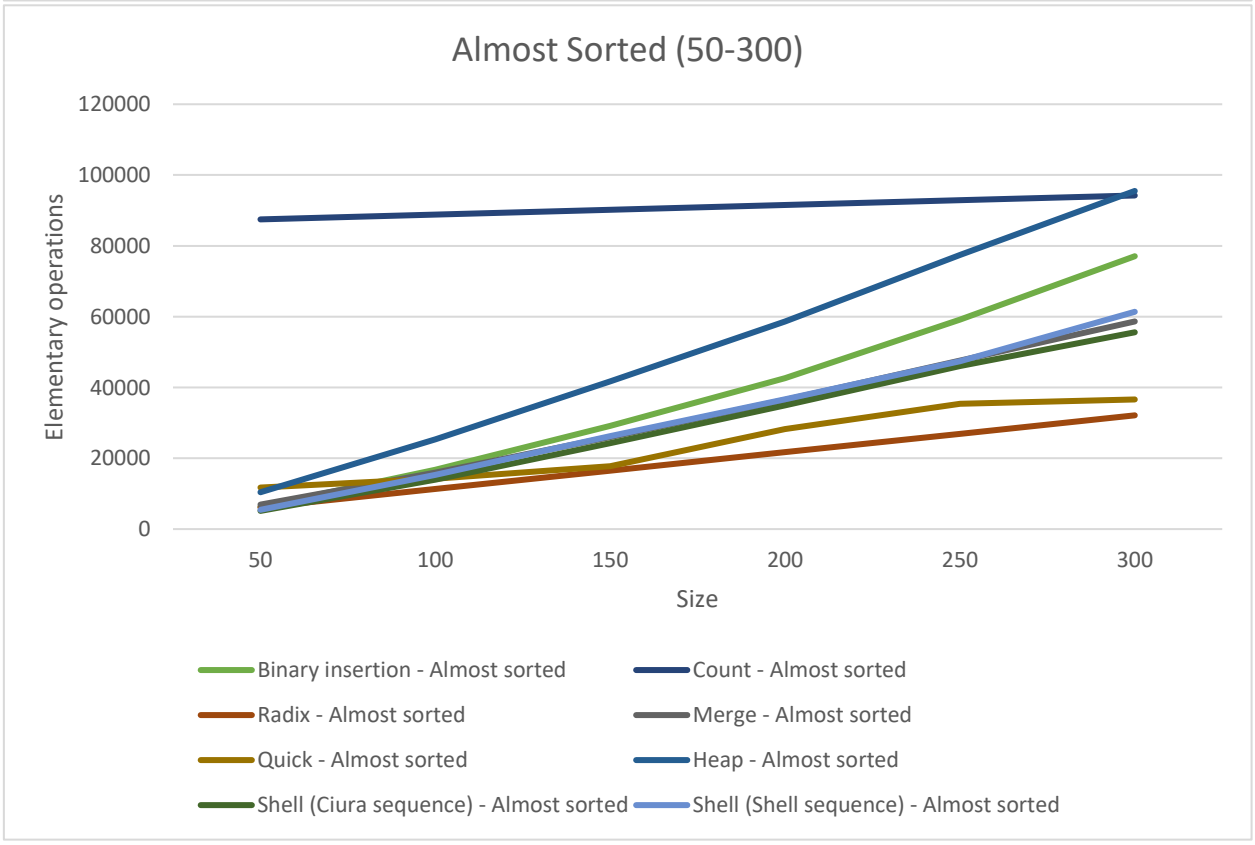


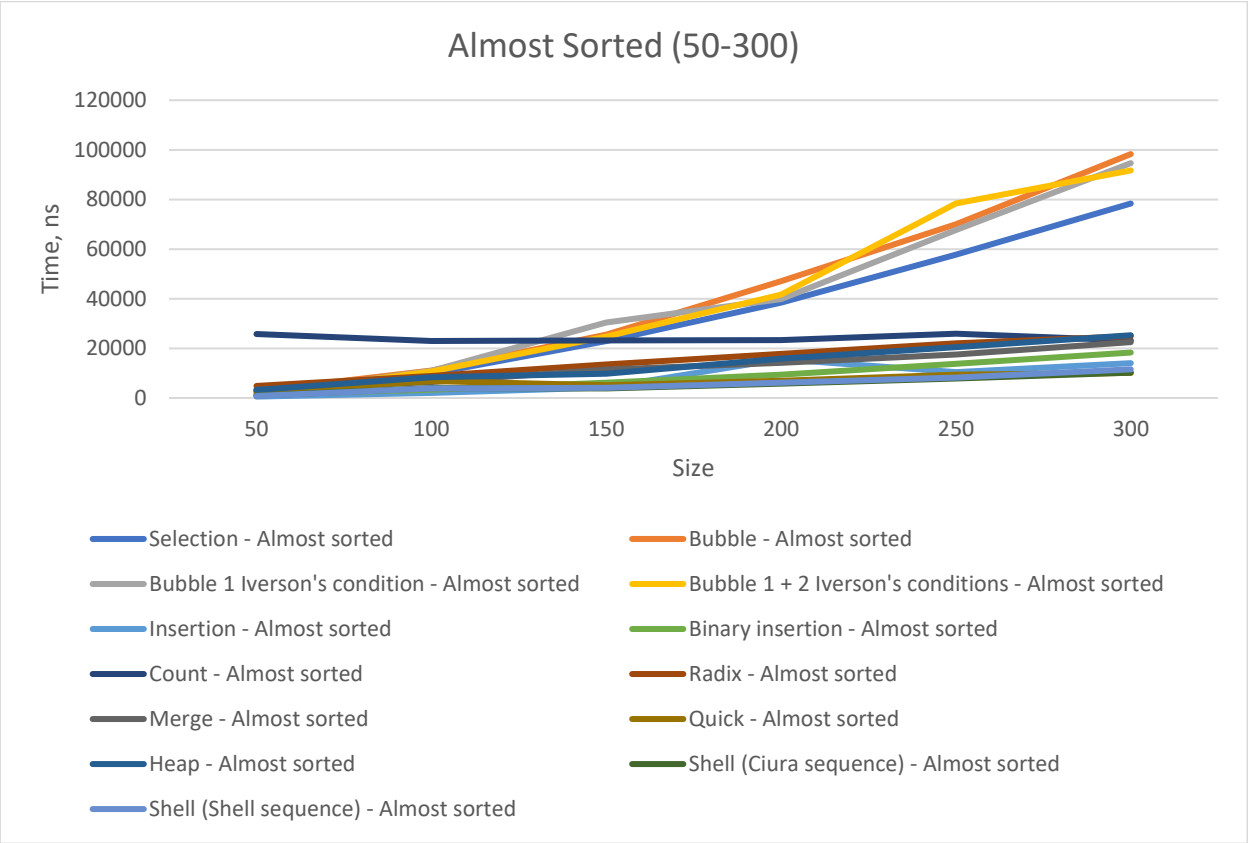
В отличие от предыдущего пункта диапазон значений увеличился - одинаковых элементов стало в среднем меньше – в сортировке Шелла стало производиться больше обменов – сложность алгоритма возросла как по времени, так и по числу элементарных операций.

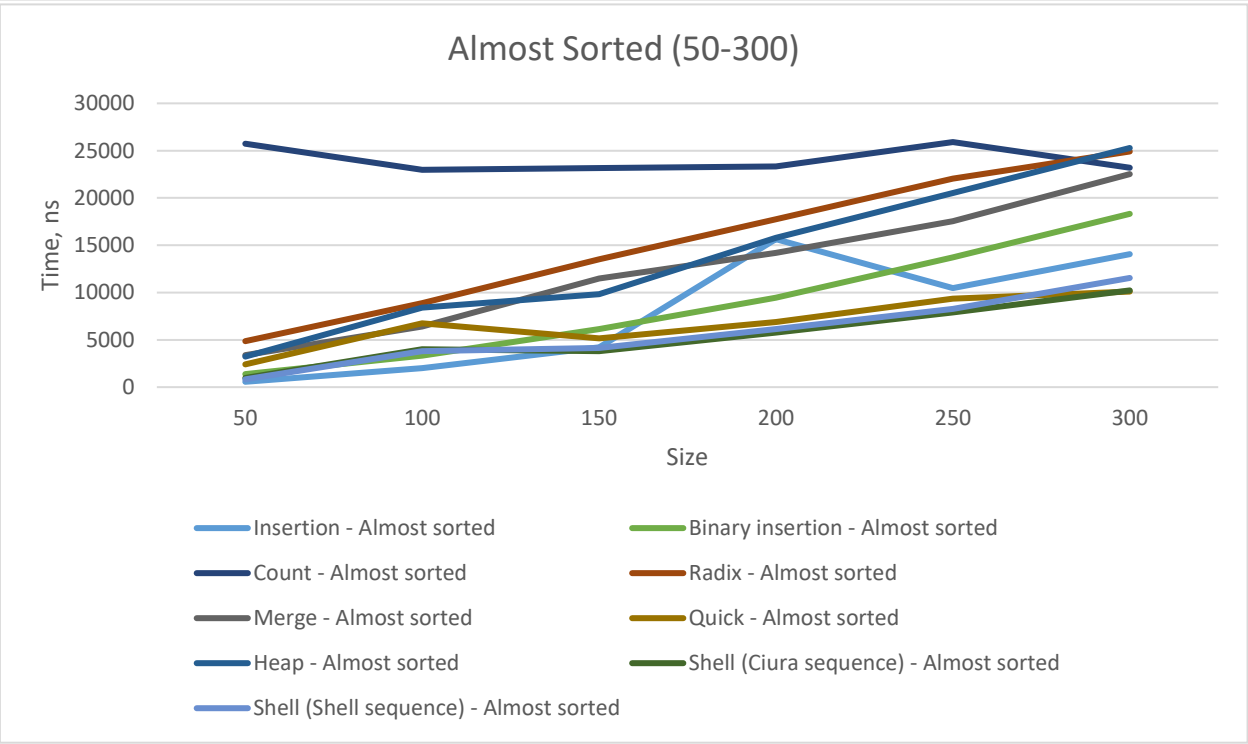
В быстрой сортировке число обменов также возросло (реализация in-place), но в меньшем количестве, поэтому и сложность увеличилась в меньшем масштабе.

3. Массив размером от 50 до 300, почти отсортированный.





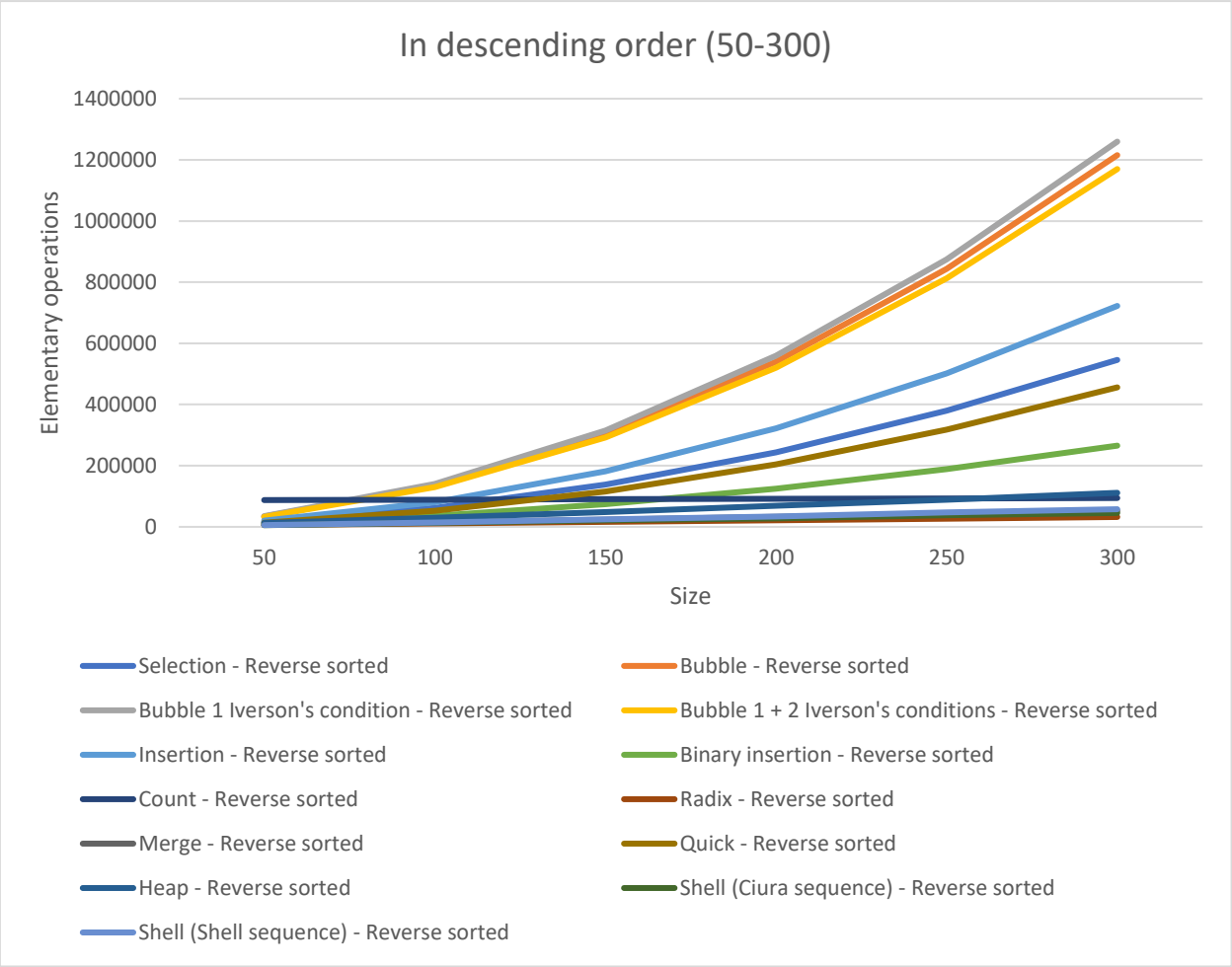


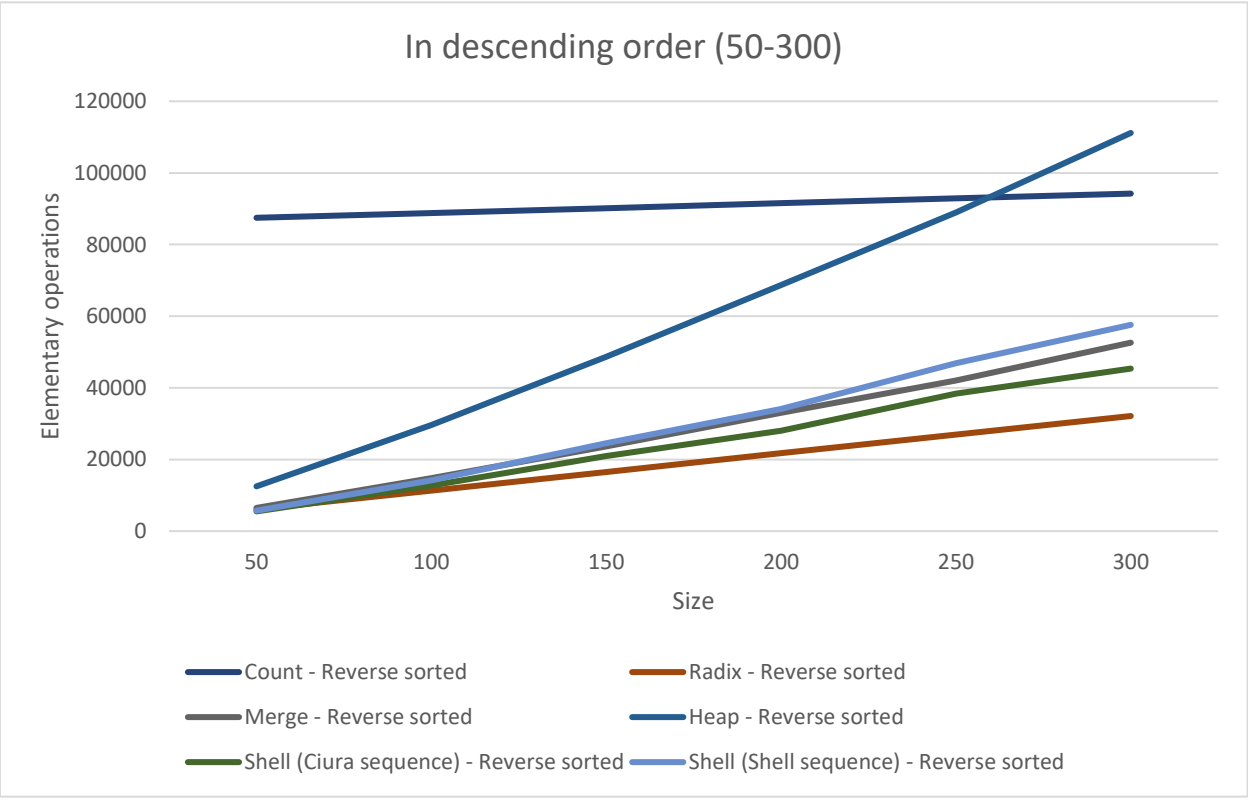


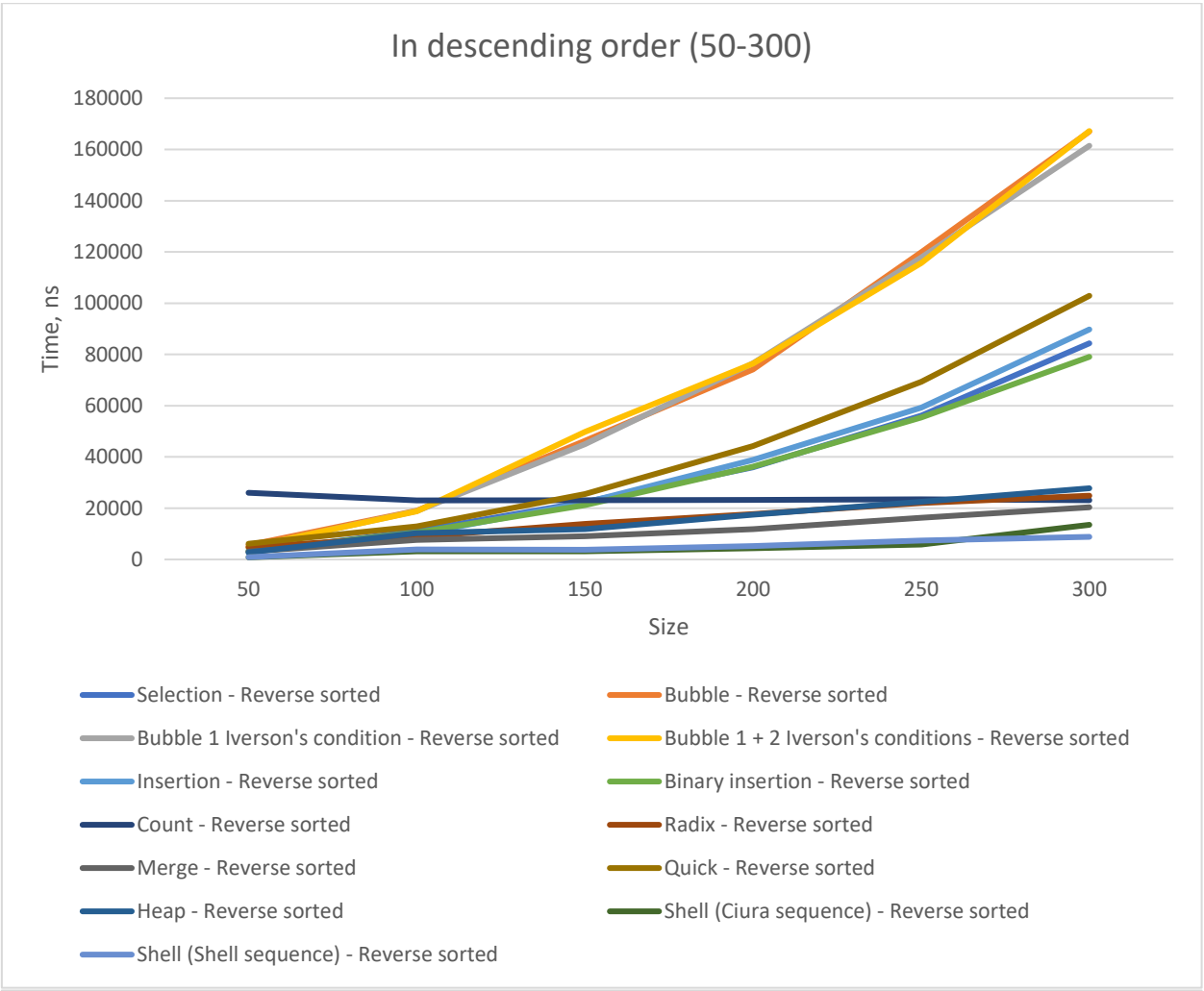
Отличие от предыдущих графиков заключается в том, что сортировки вставками показали результат лучше. Это произошло, из-за того что почти все элементы находились на своих местах, а число сравнений (и переопределний элементов массива) зависит от числа элементов не на своих местах – сложность стремится к лучшей – O(n).

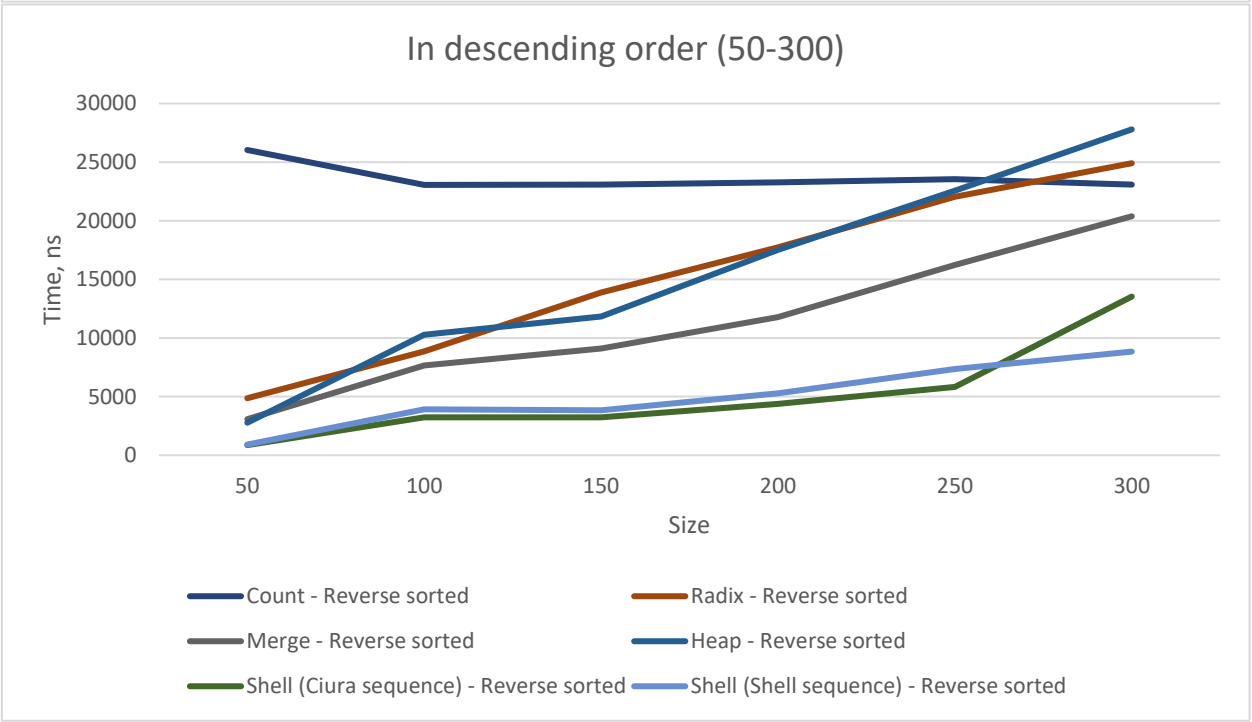
Ещё одним изменением является ухудшение времени работы быстрой сортировки. Т.к. почти все элементы находятся на своих местах, а опорным выбирается всегда первый элемент, это практически худший случай, однако это не так критично из-за того, что практически не происходило обменов (особенности реализации).

4. Массив размером от 50 до 300, обратно отсортированный.



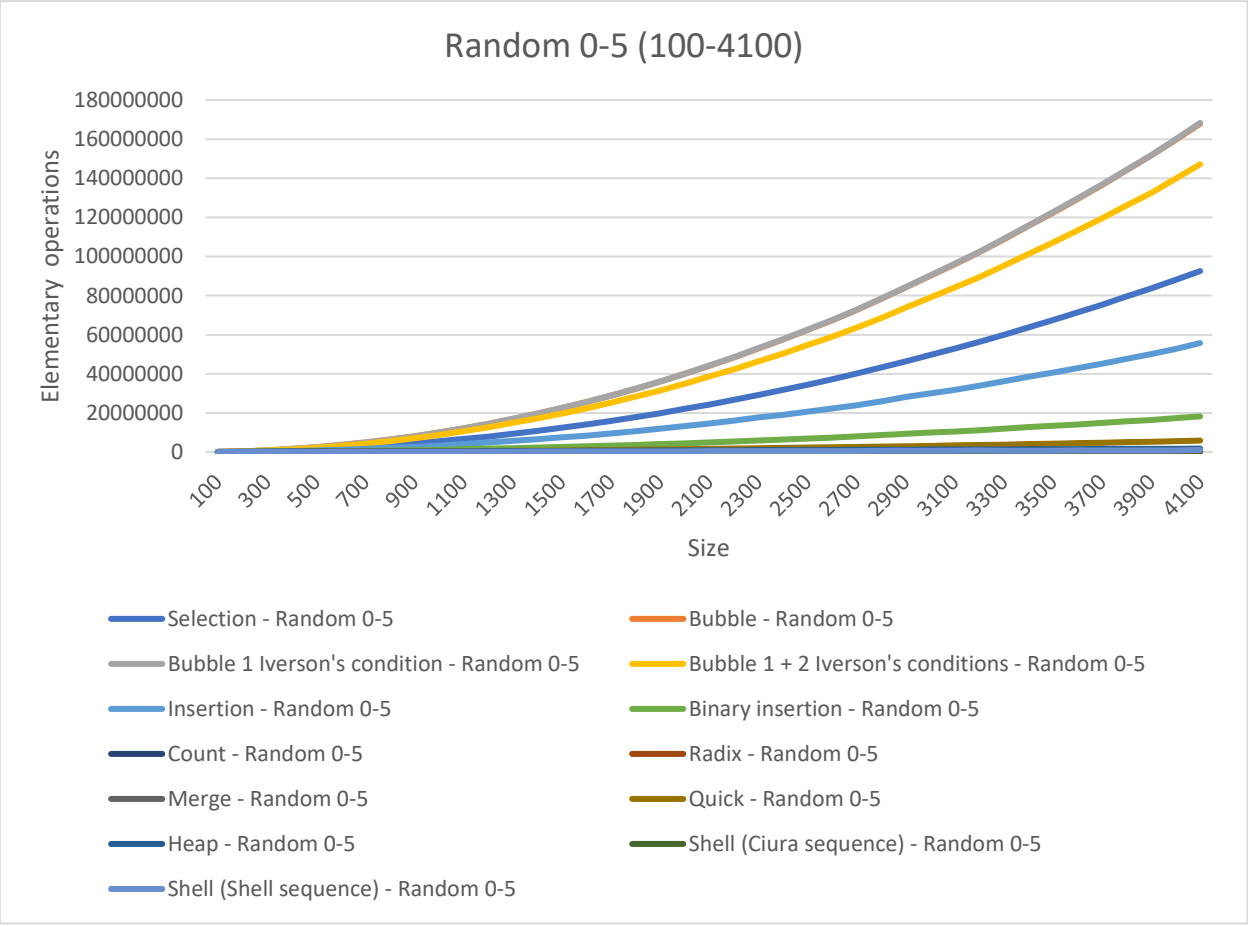


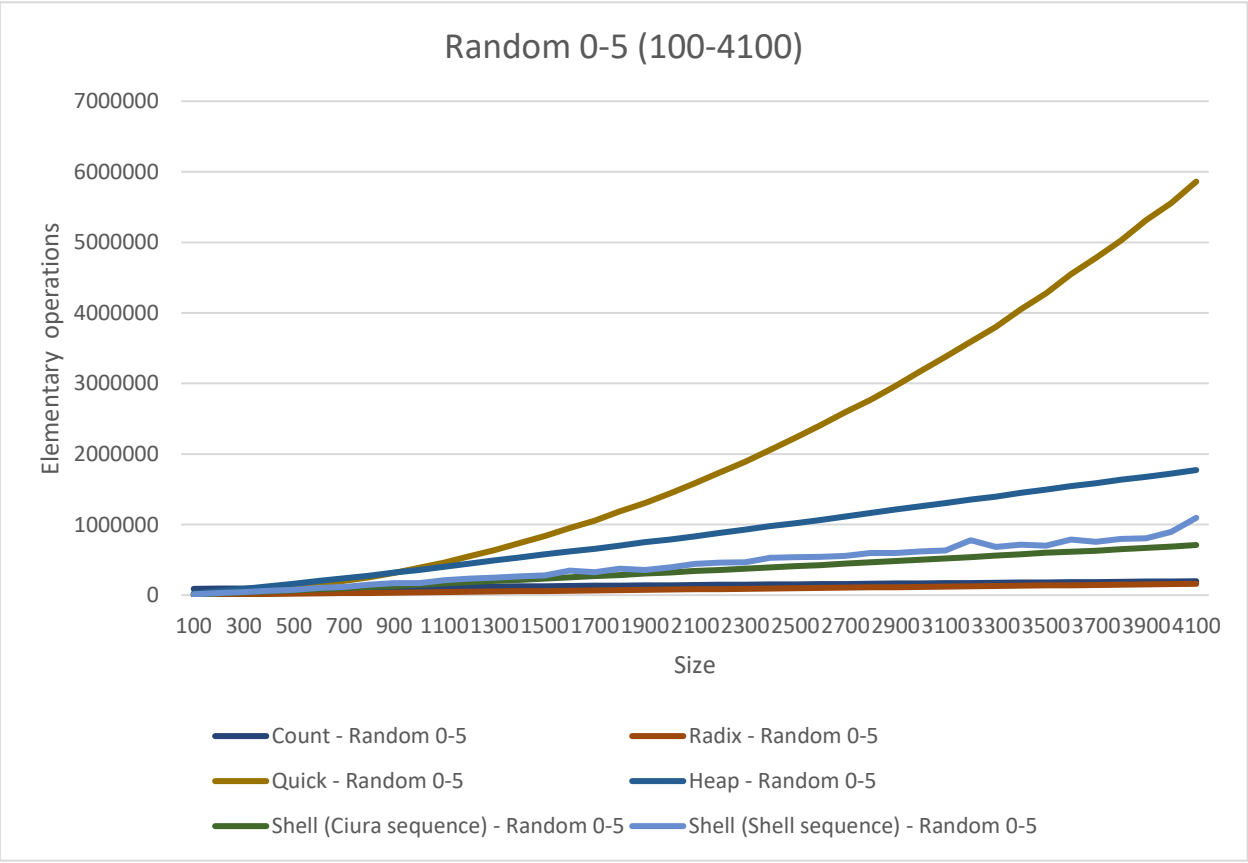


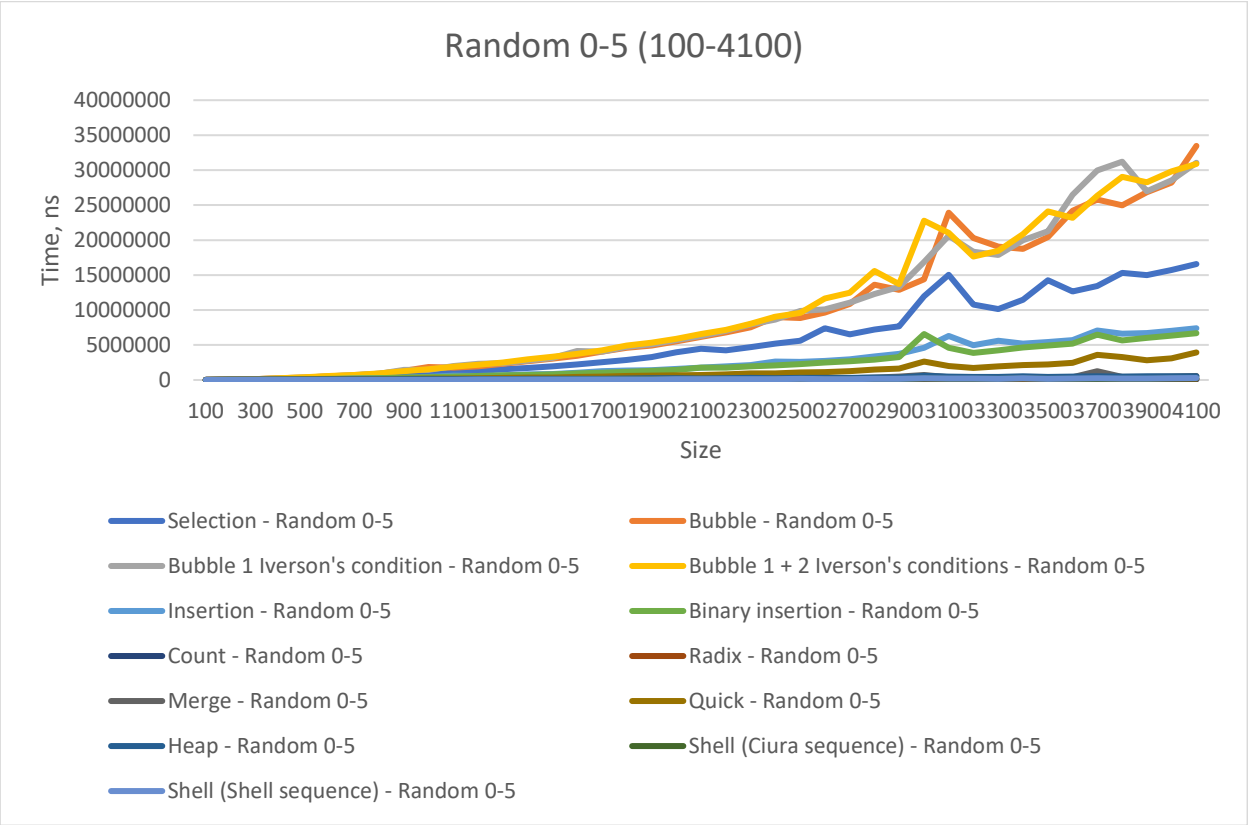


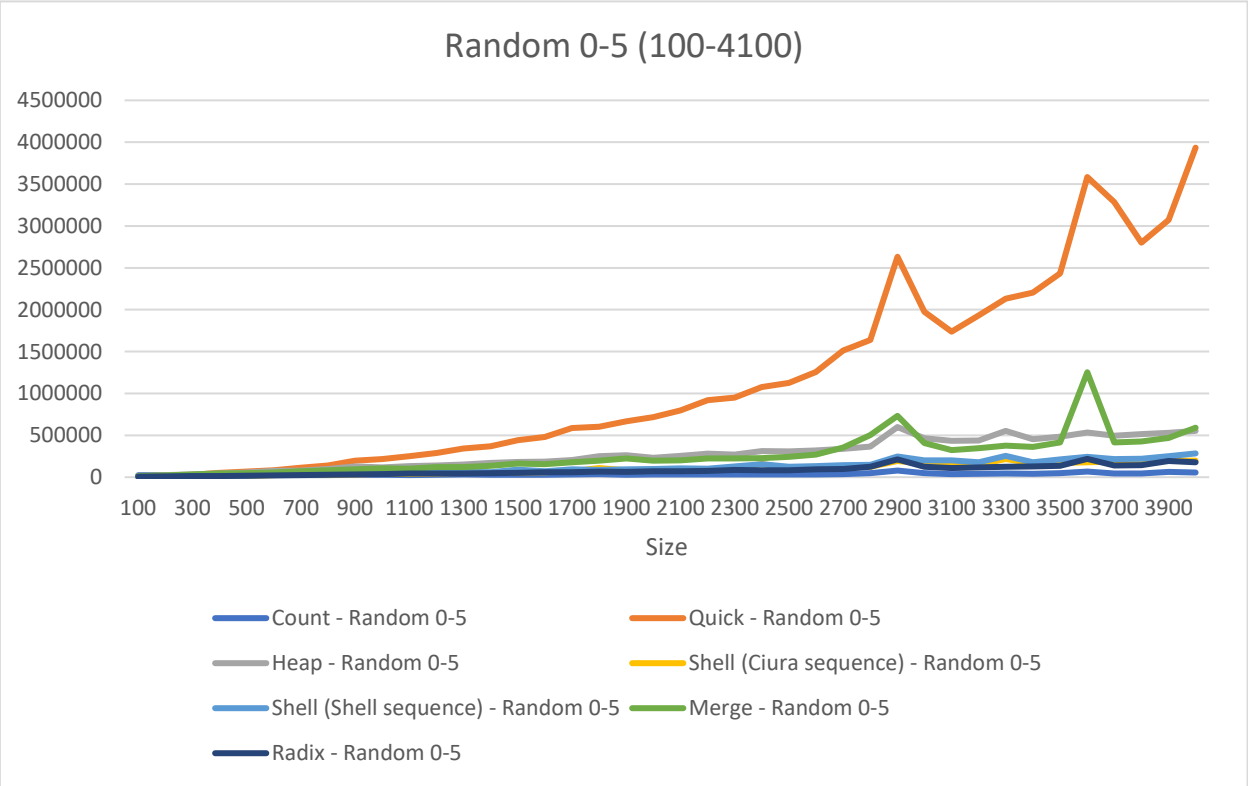
Сложность быстрой сортировки и сортировок вставками сильно возрасла в связи с тем, что теперь обмен происходит на каждой итерации.

5. Массив размером от 100 до 4100, заполненный случайными числами в диапазоне от 0 до 5





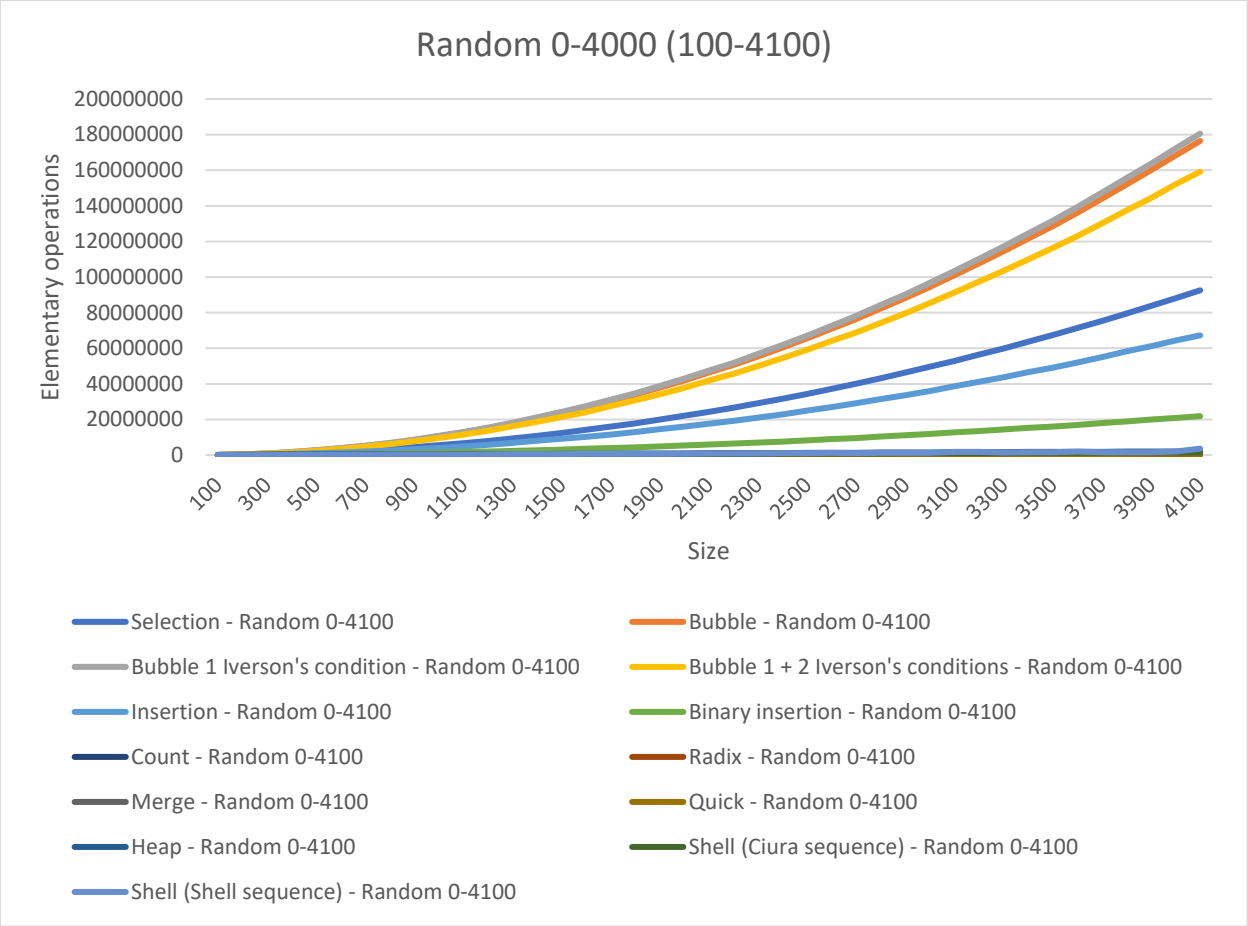


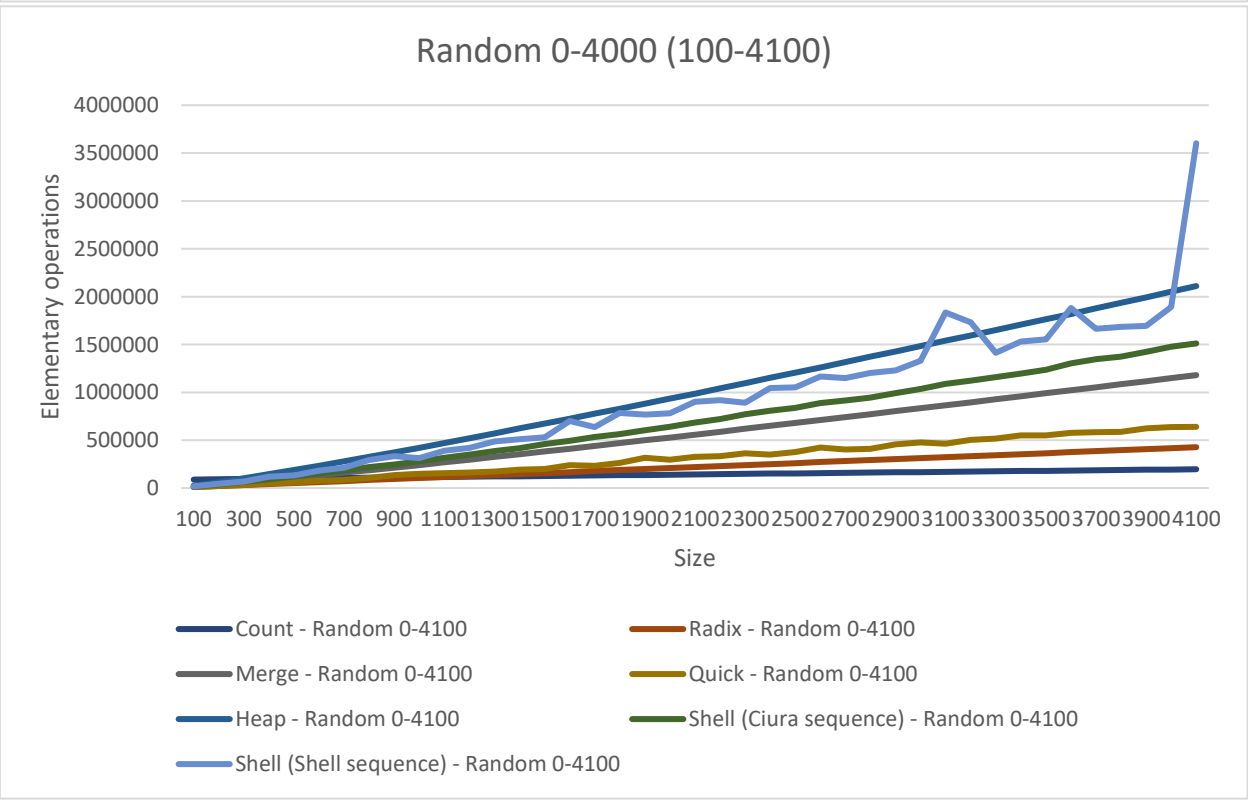


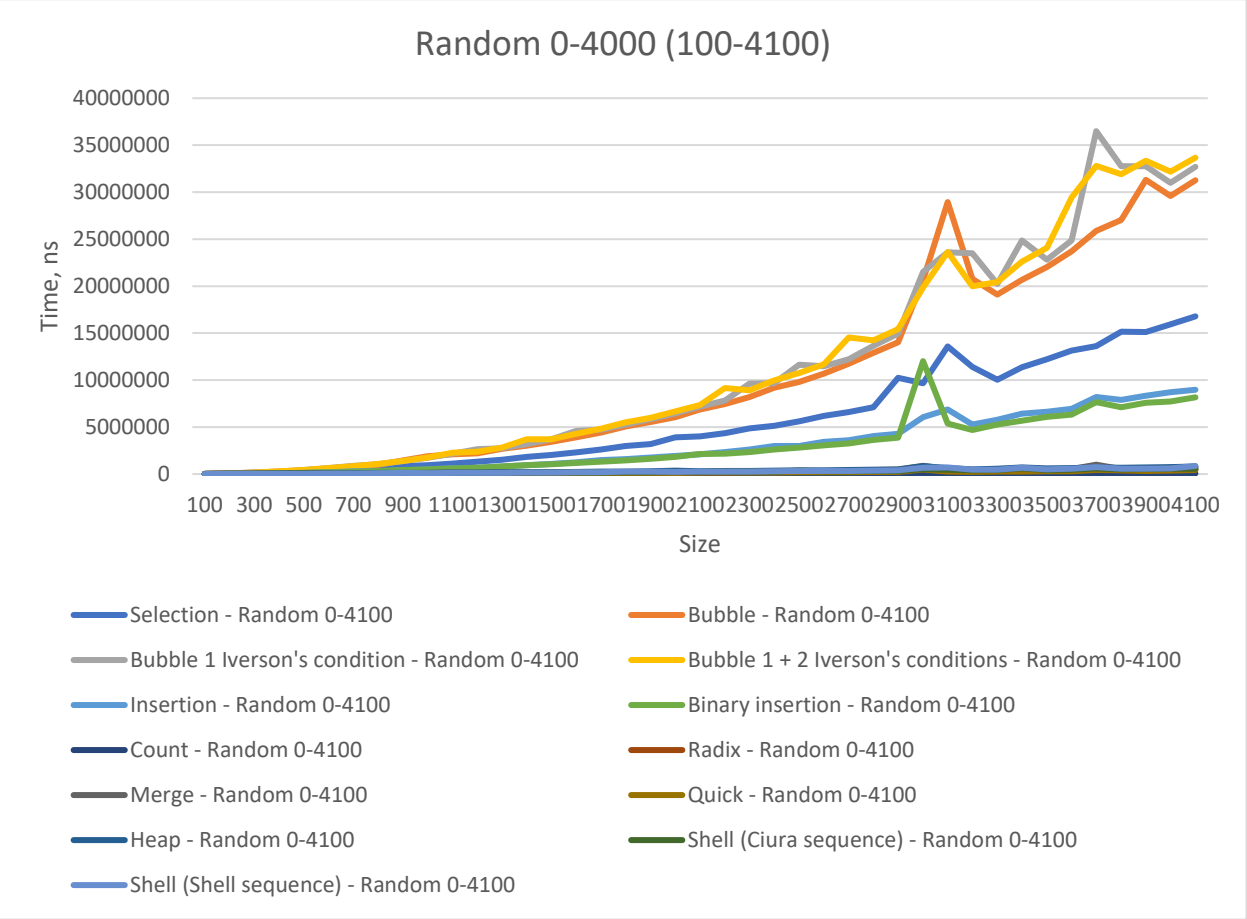
В сопоставлении с графиками из п. 1:

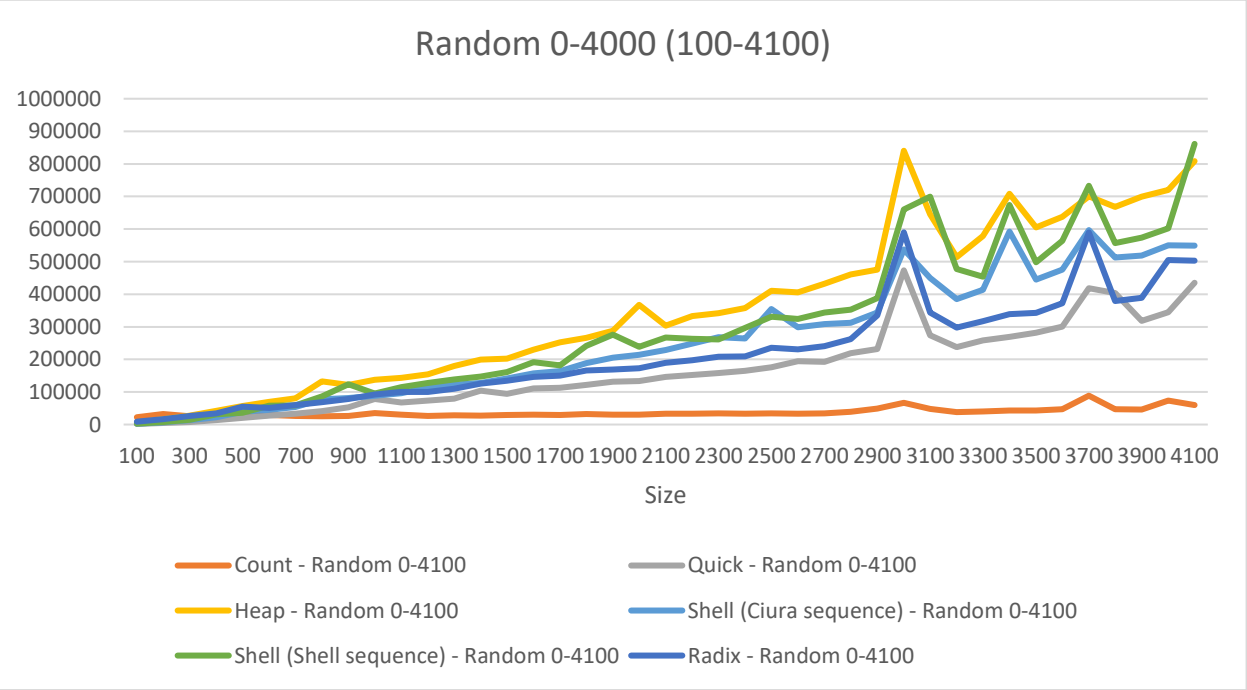
- Достаточно сильно возрасла сложность быстрой сортировки относительно других, т.к. многократное повторение элементов и увеличение размера массива приводят к тому, что вероятность попасть в максимальный элемент на любой из итераций становится очень высокой;
- Сортировка подсчётом и цифровая сортировка примерно сравнялись и показали лучший результат среди прочих сортировок, т.к. они линейны.

6. Массив размером от 100 до 4100, заполненный случайными числами в диапазоне от 0 до 4000.





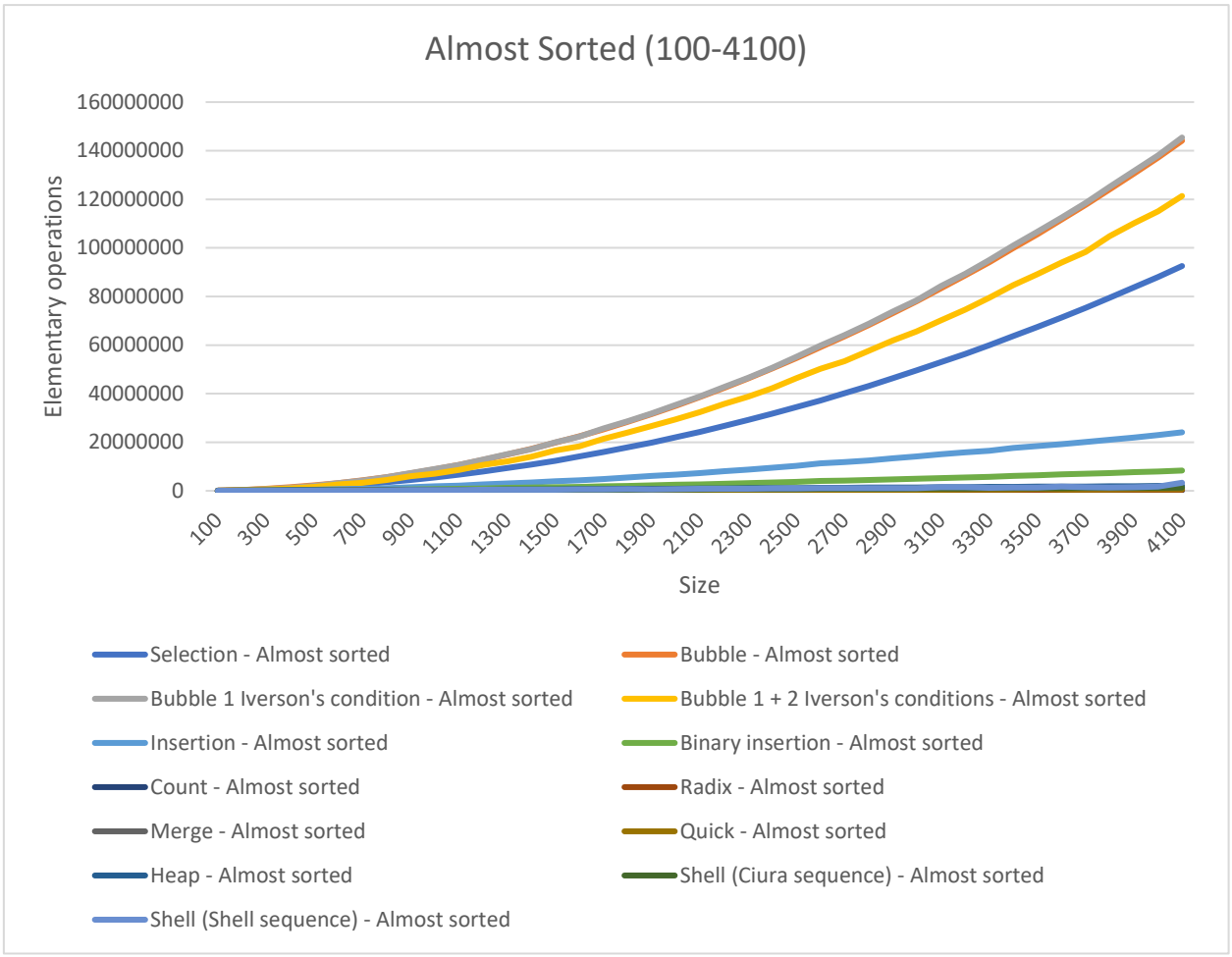


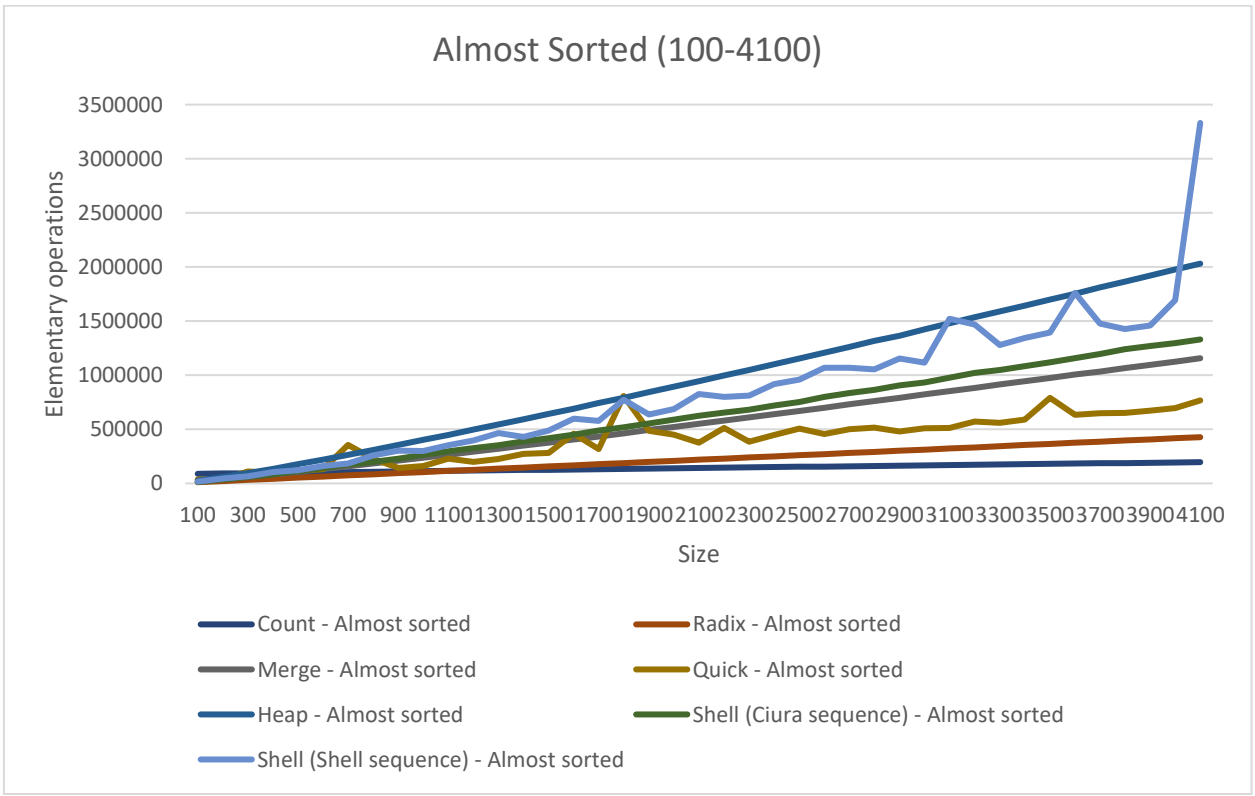


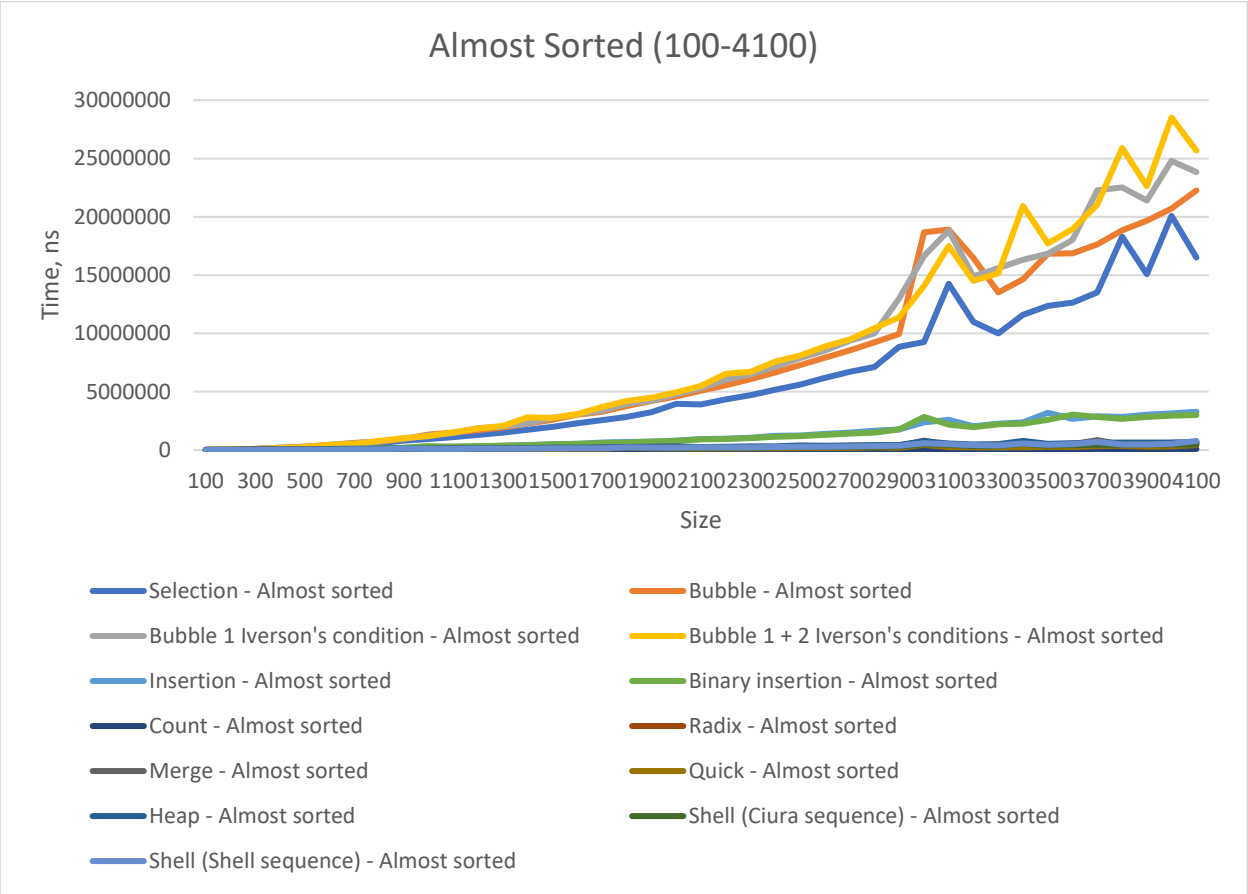
В отличие от предыдущего пункта диапазон значений увеличился - одинаковых элементов стало в среднем меньше – распределение элементов стало более равномерным – части, на которые делится массив в ходе быстрой сортировки, стали более равными. Поэтому сложность быстрой сортировки относительно других понизилась.

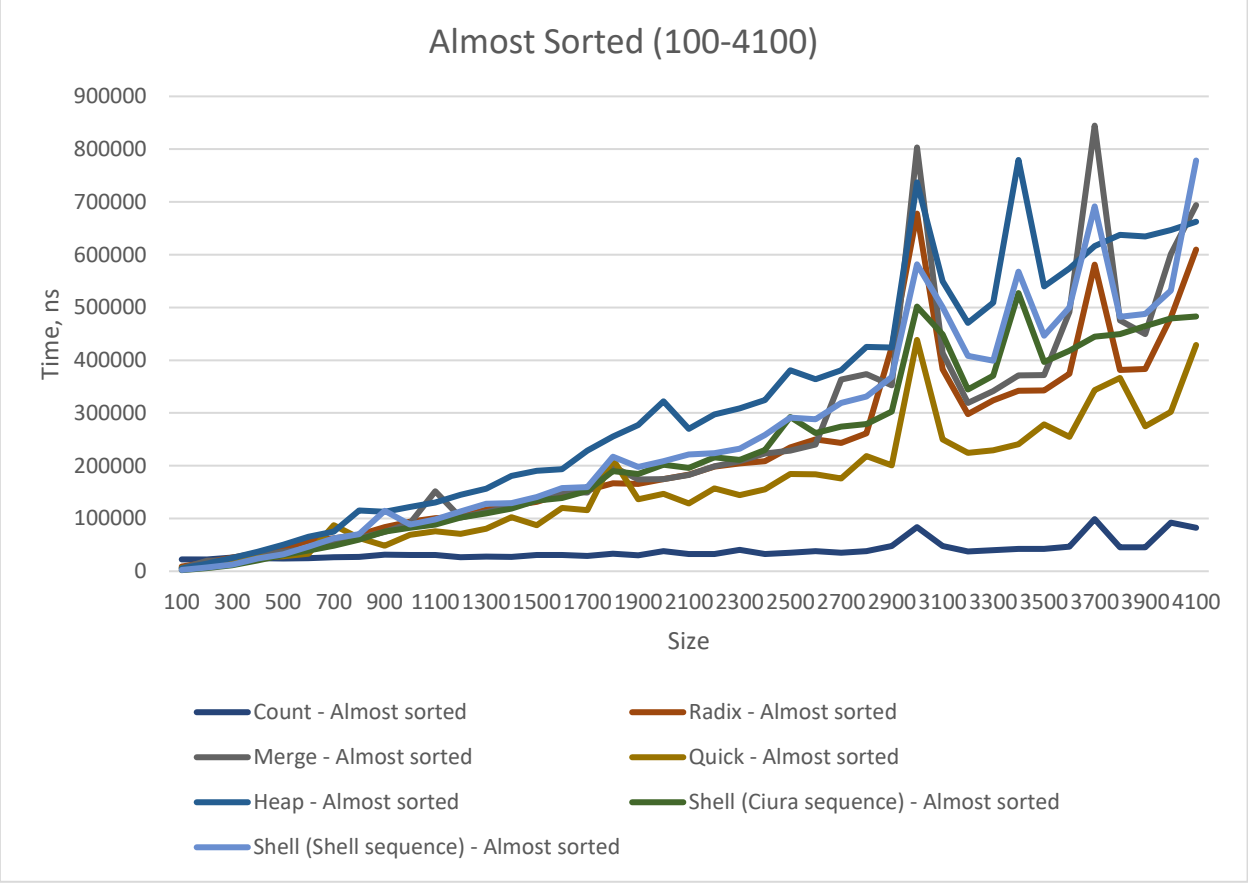
От увеличения диапазона значений также пострадала цифорвая сортировка, т.к. она зависит от числа разрядов.

7. Массив размером от 100 до 4100, почти отсортированный.



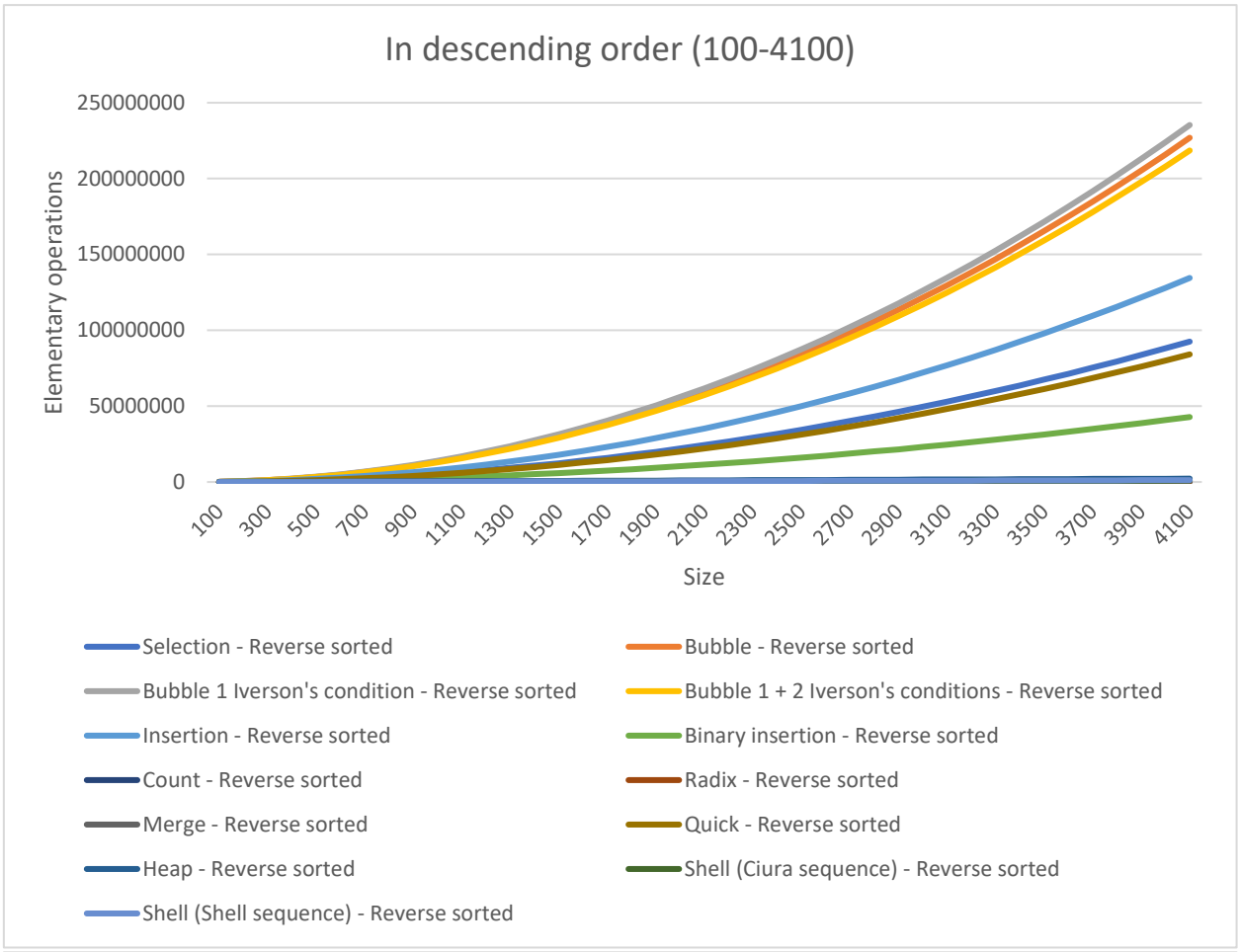


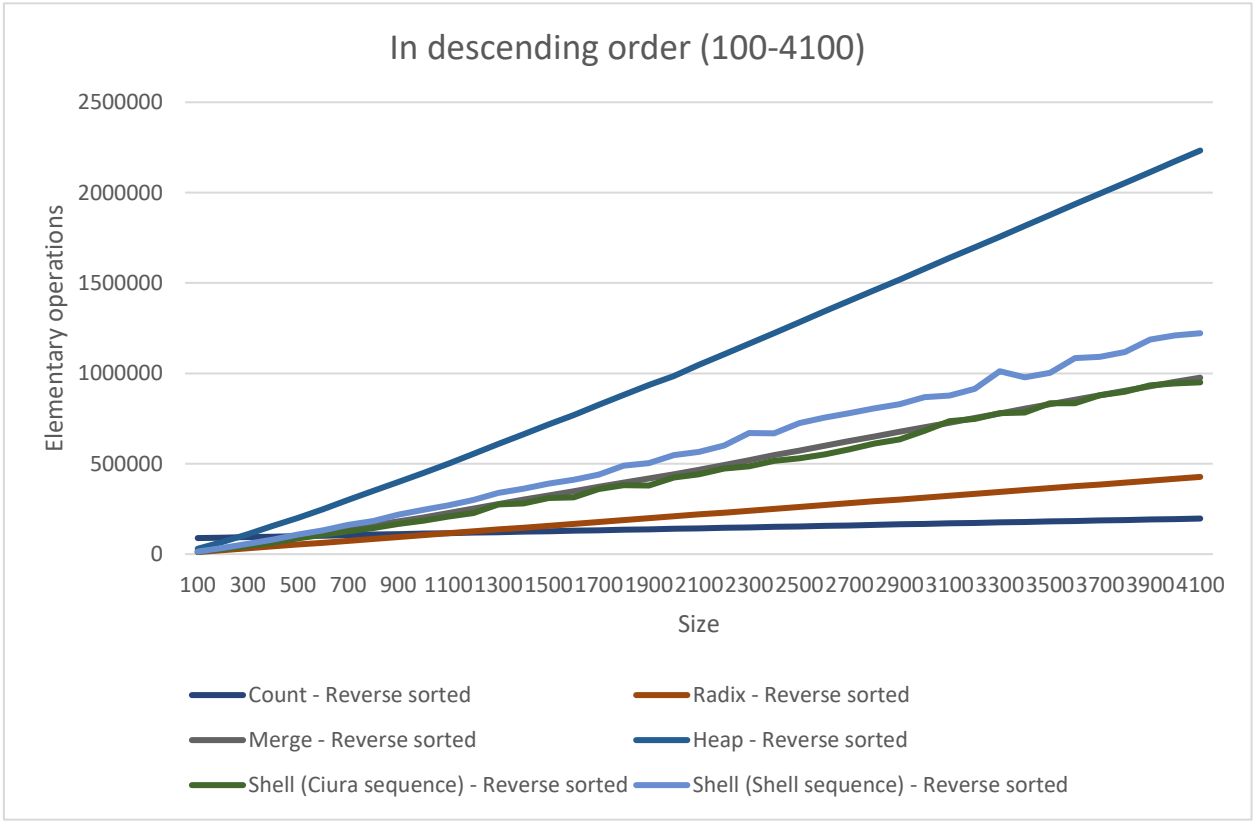


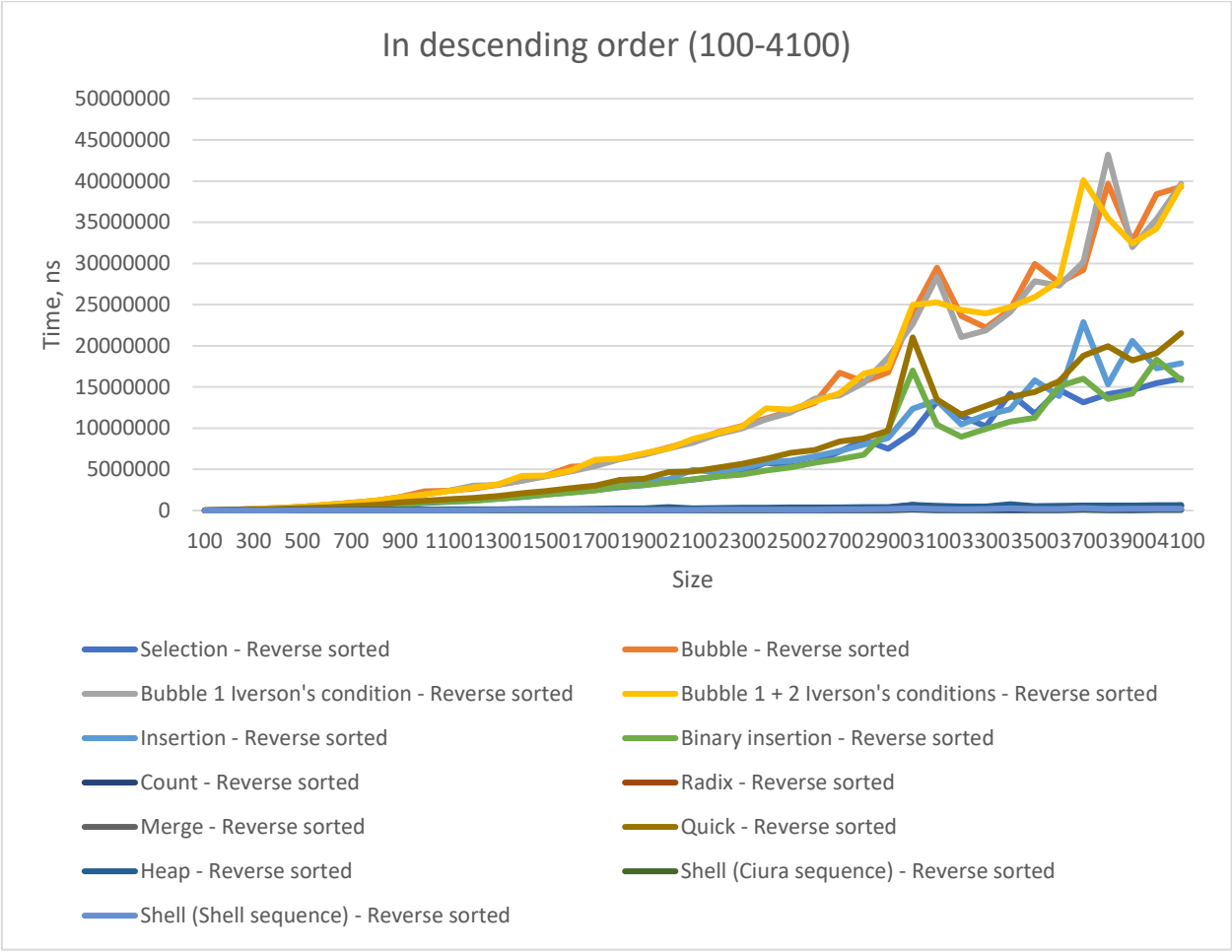


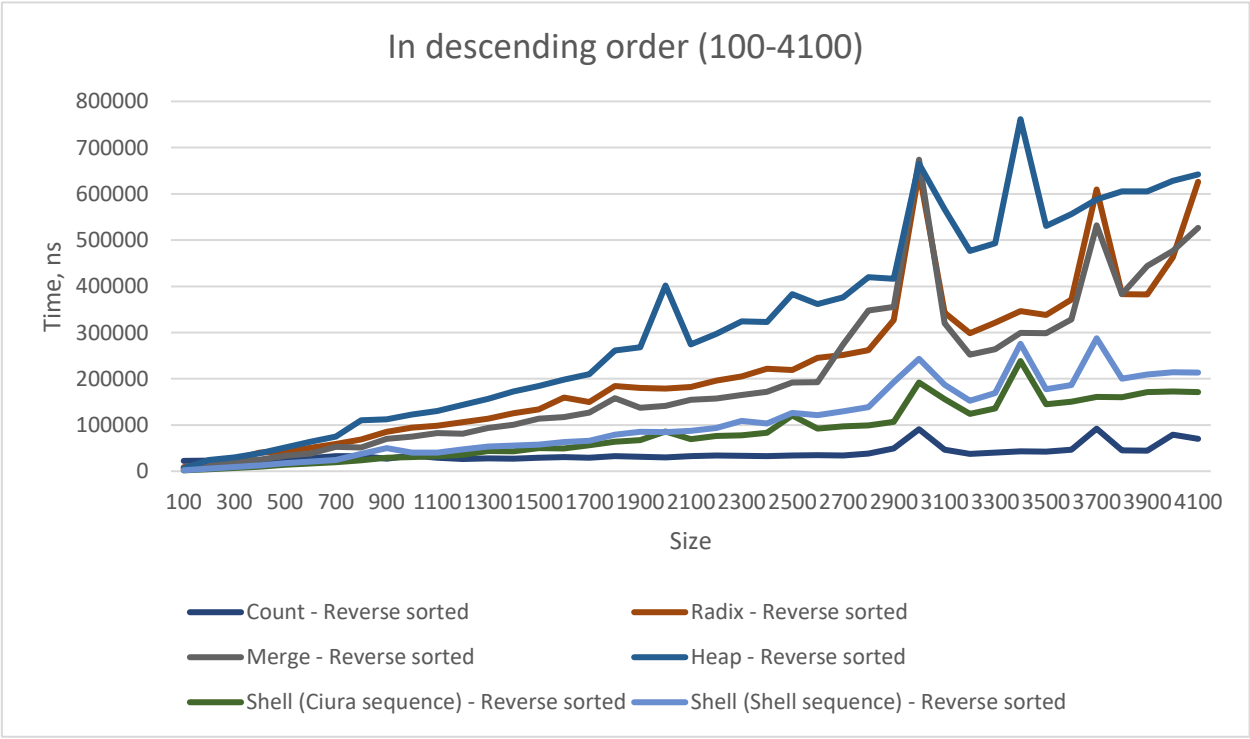
Отличается от предыдщуего только ускорением сортировок вставками (уменьшилось число сравнений и сдвигов подмассива).

8. Массив размером от 100 до 4100, обратно отсортированный.



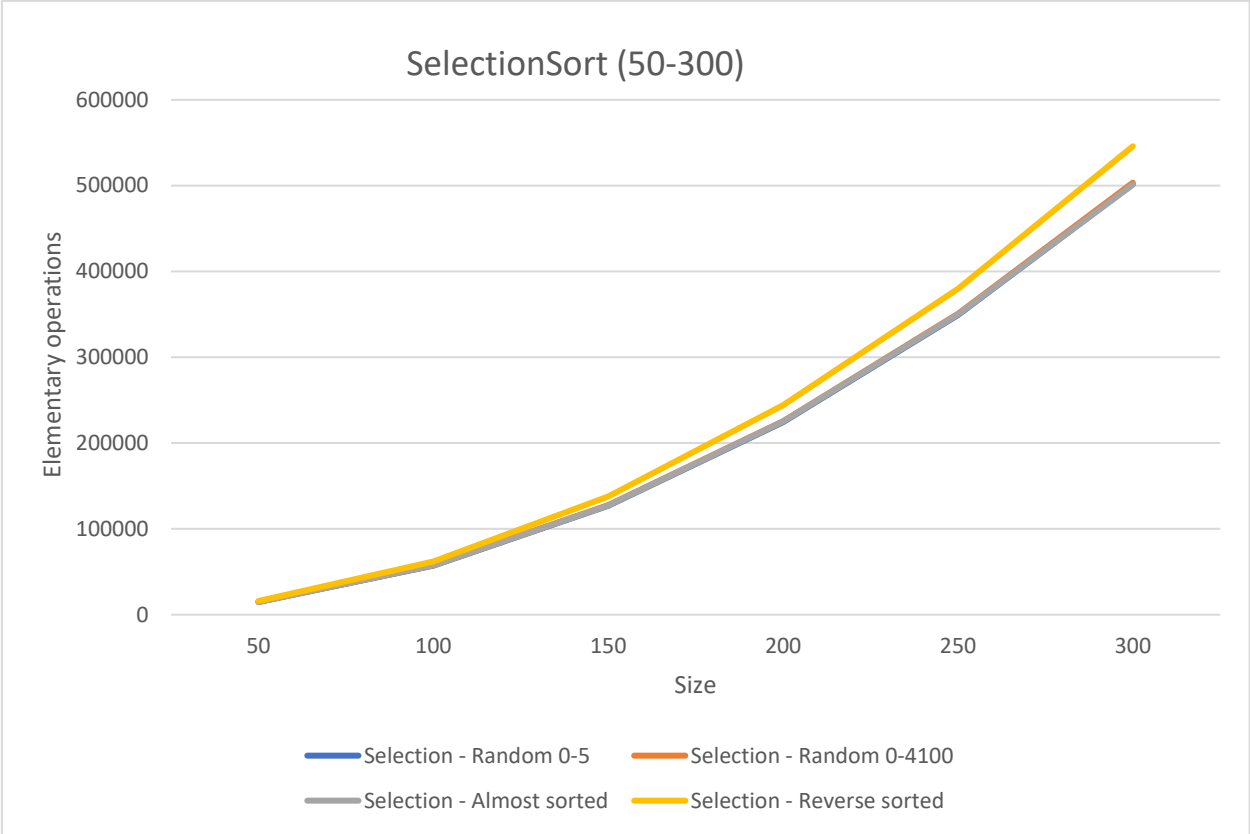


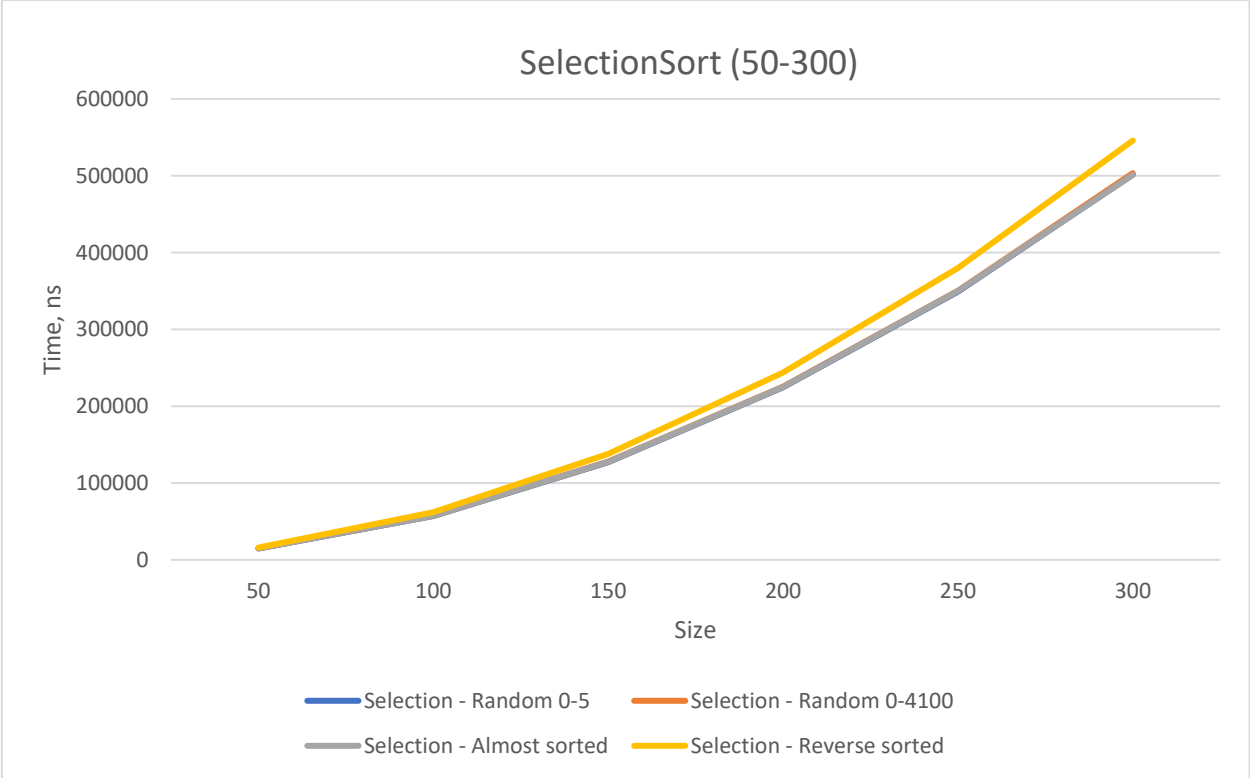


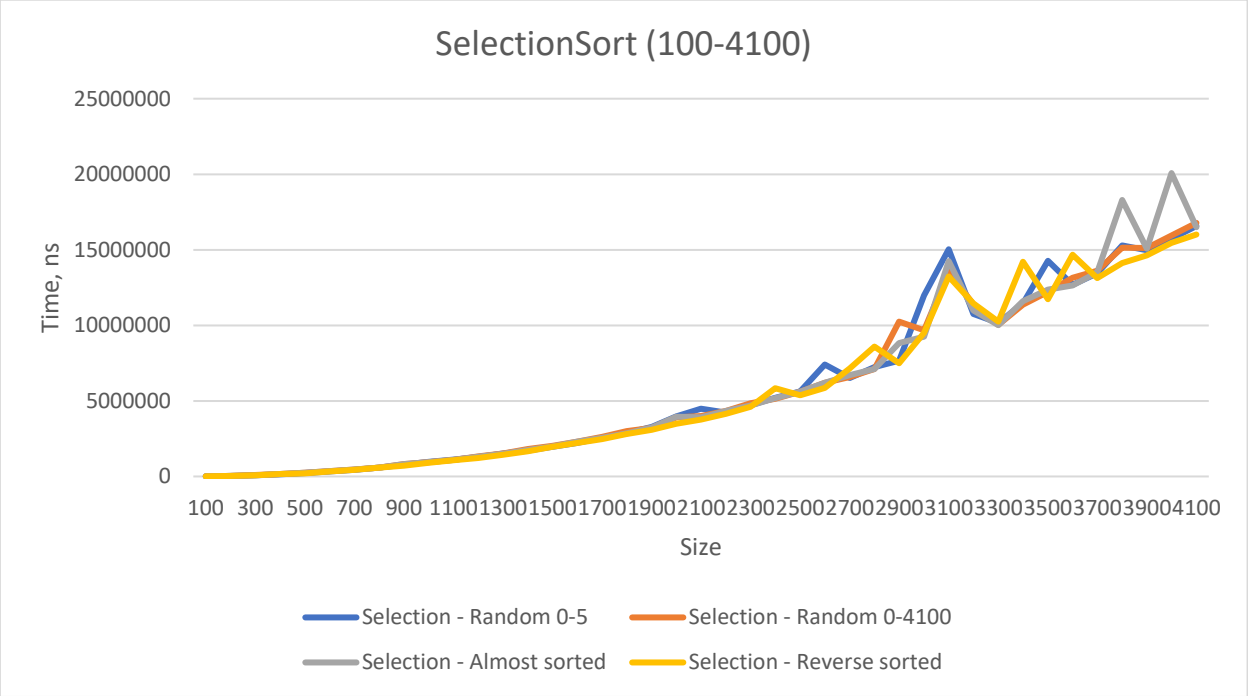


Сложность быстрой сортировки и сортировок вставками сильно возрасла в связи с тем, что теперь обмен происходит на каждой итерации.

9. Сортировка выбором.



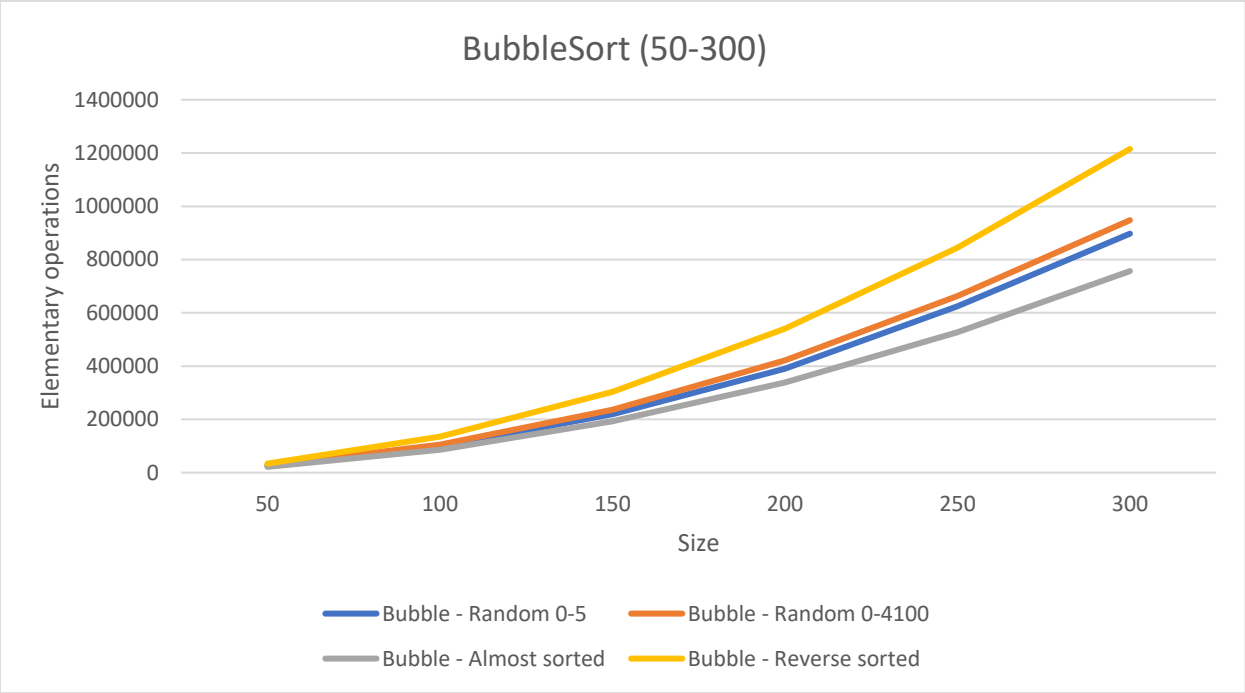


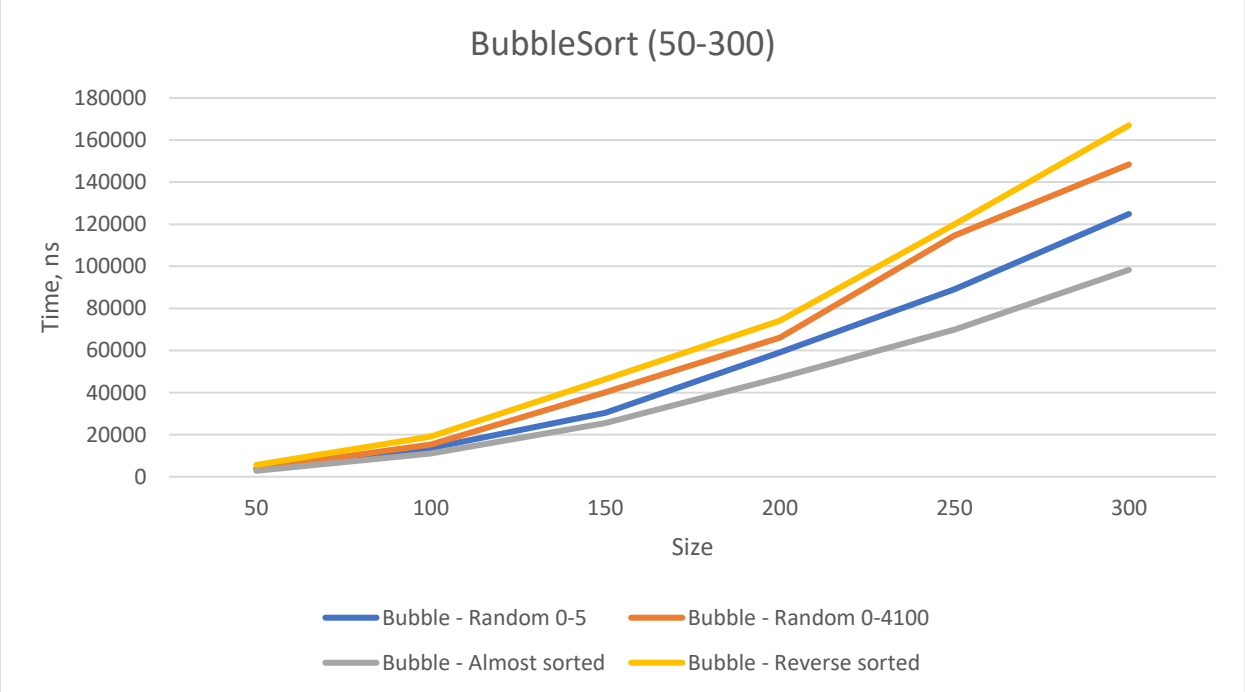


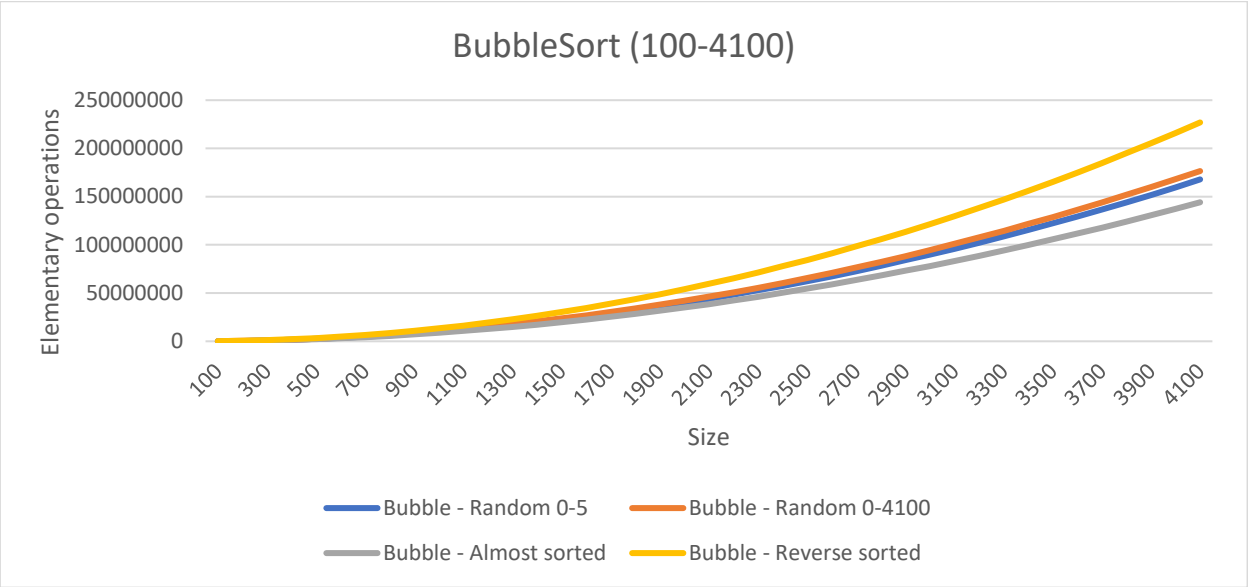
Работает почти одинаково на любых данных ($\Theta(n^2)$).

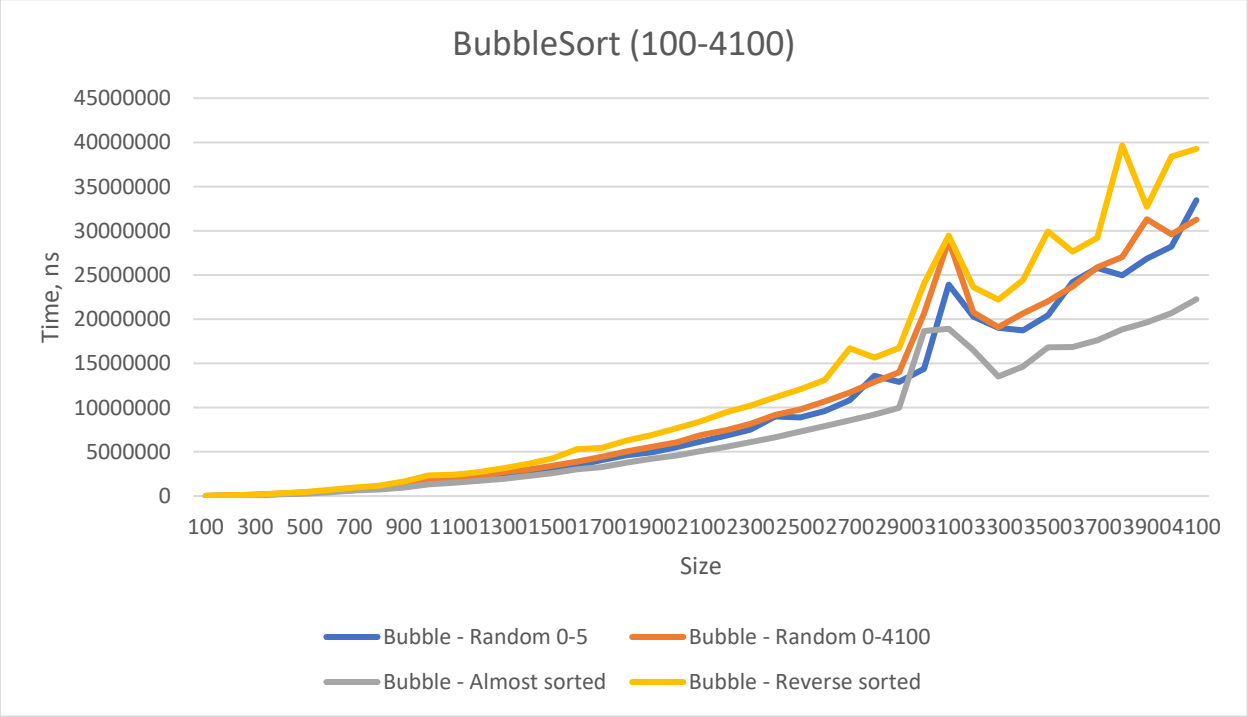
На обратно отсортированном массиве сложность увеличивается, т.к. при линейном поиске минимального элемента его индекс постоянно переопределяется.

10. Сортировка пузырьком (без оптимизаций).





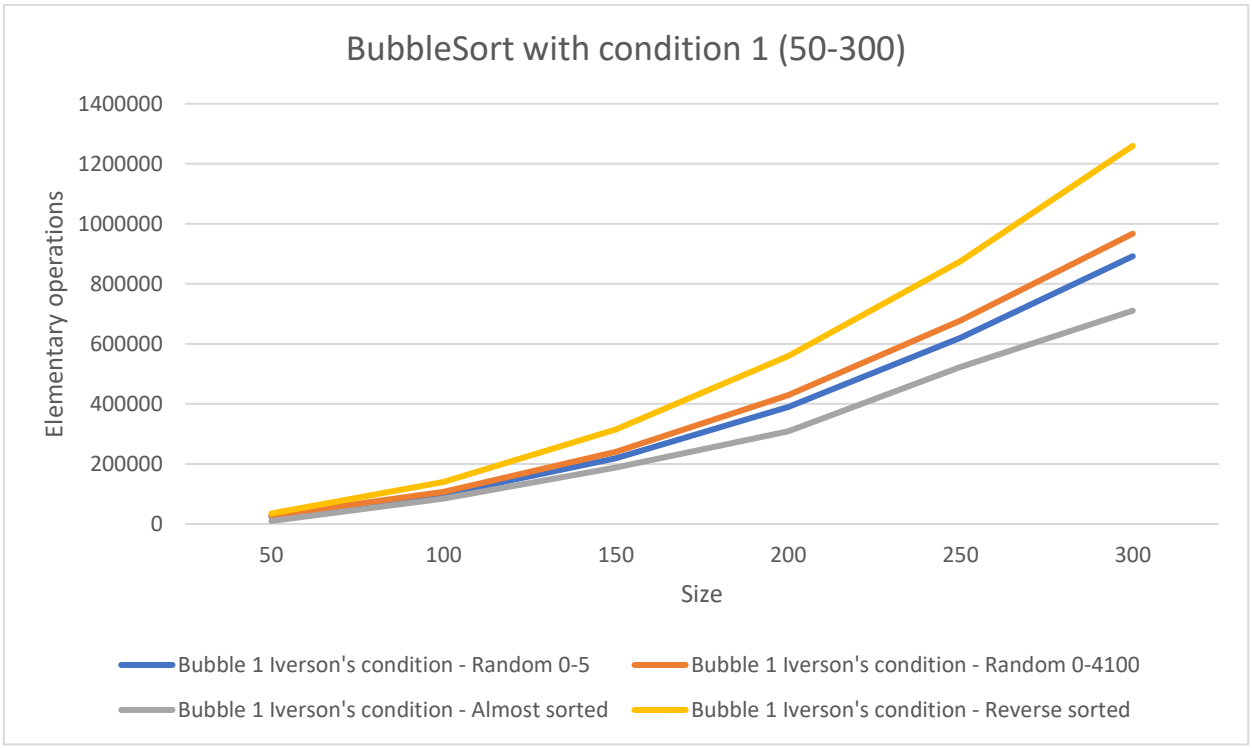


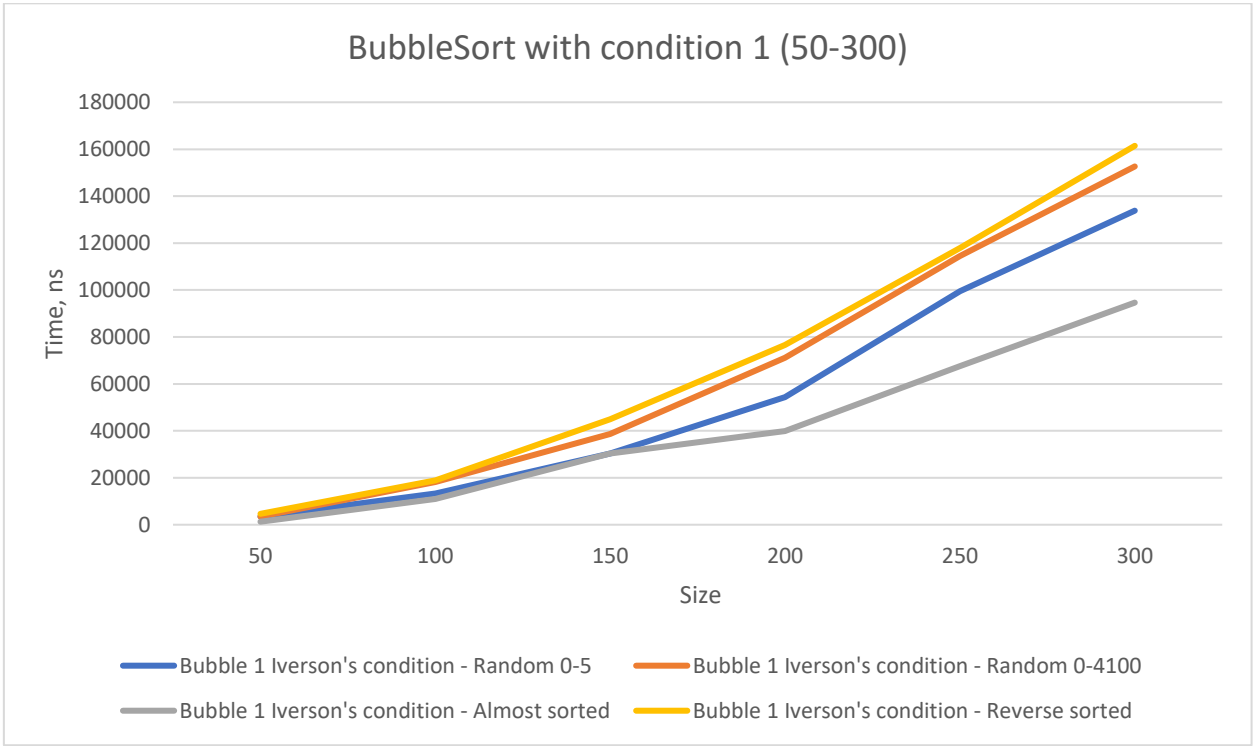


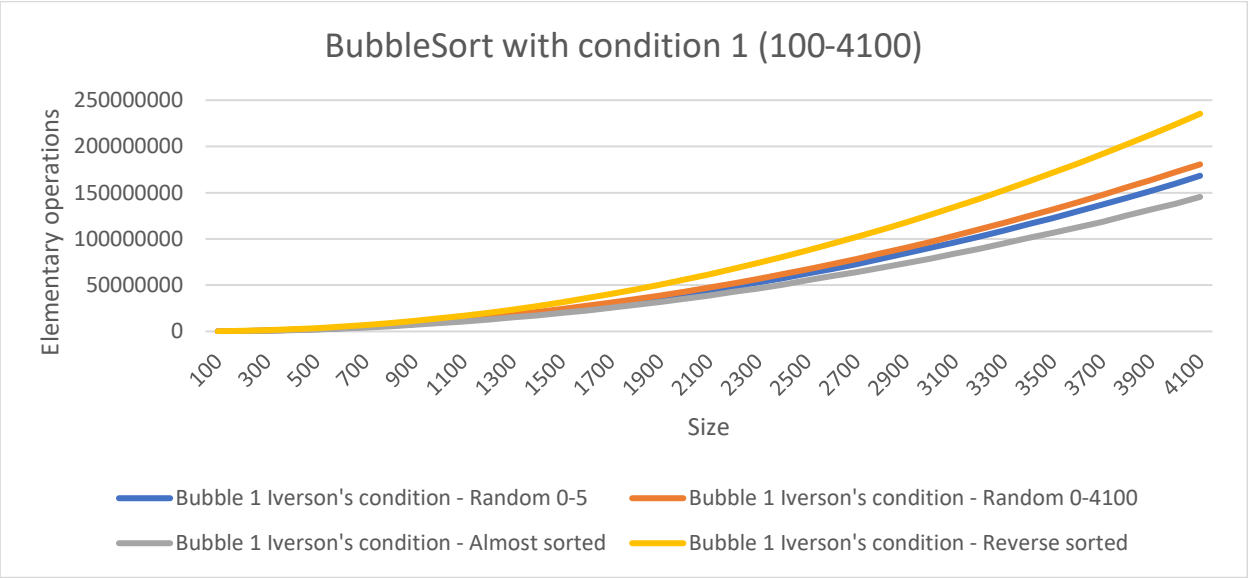
Работает почти одинаково на любых данных ($\Theta(n^2)$).

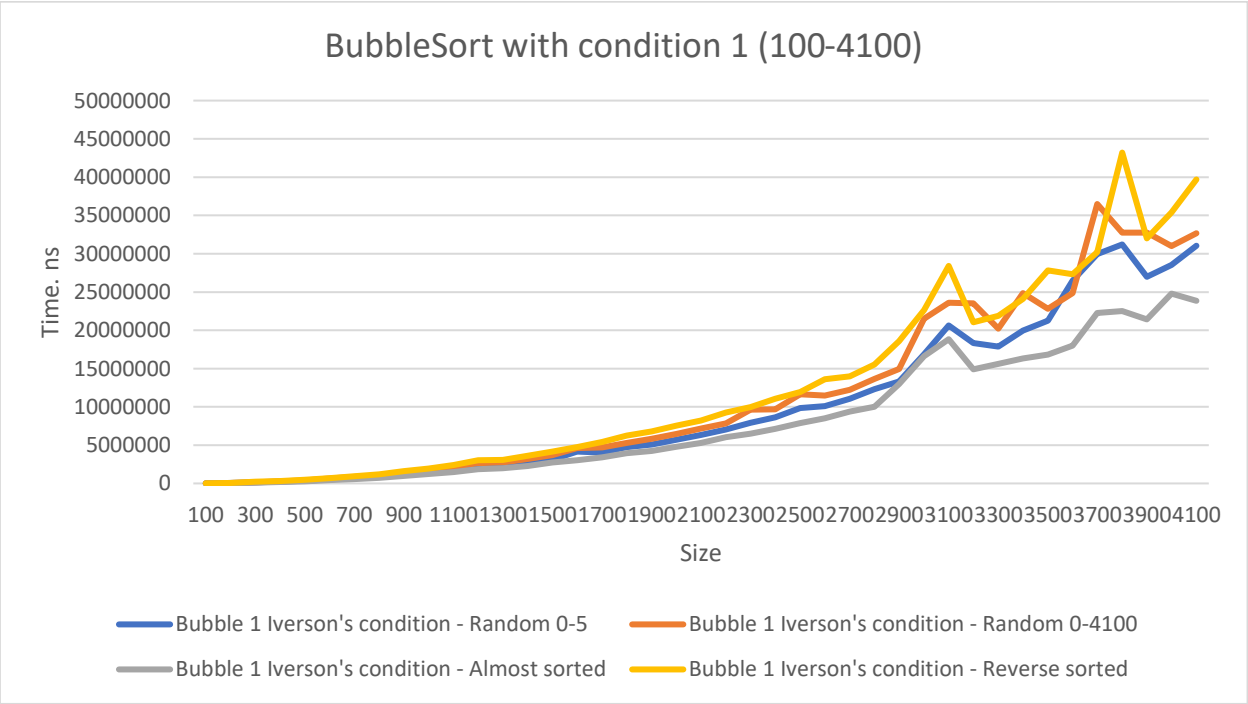
Лучше всего работает на почти отсортированном массиве, т.к. обмен происходит только для элементов, нарушающих порядок. По этой же причине обратно отсортированный массив является худшим случаем, на нем сортировка пузырьком работает хуже всего.

11. Сортировка пузырьком (1-е условие Айверсона).





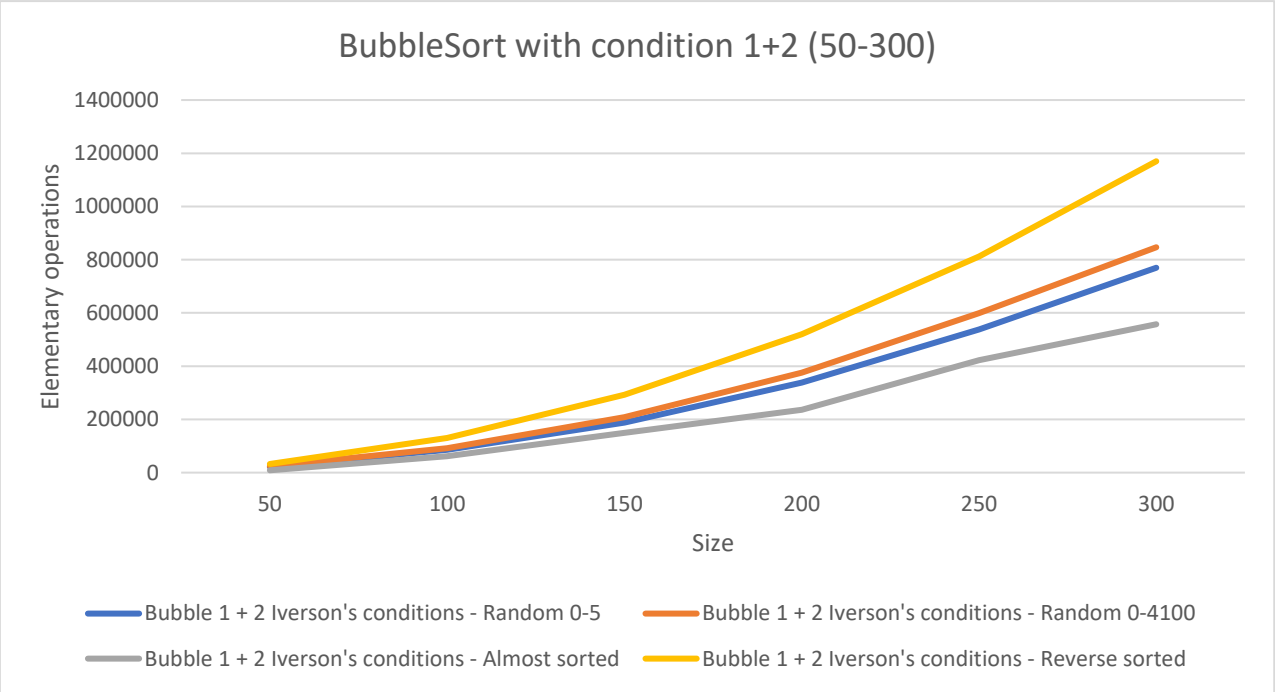


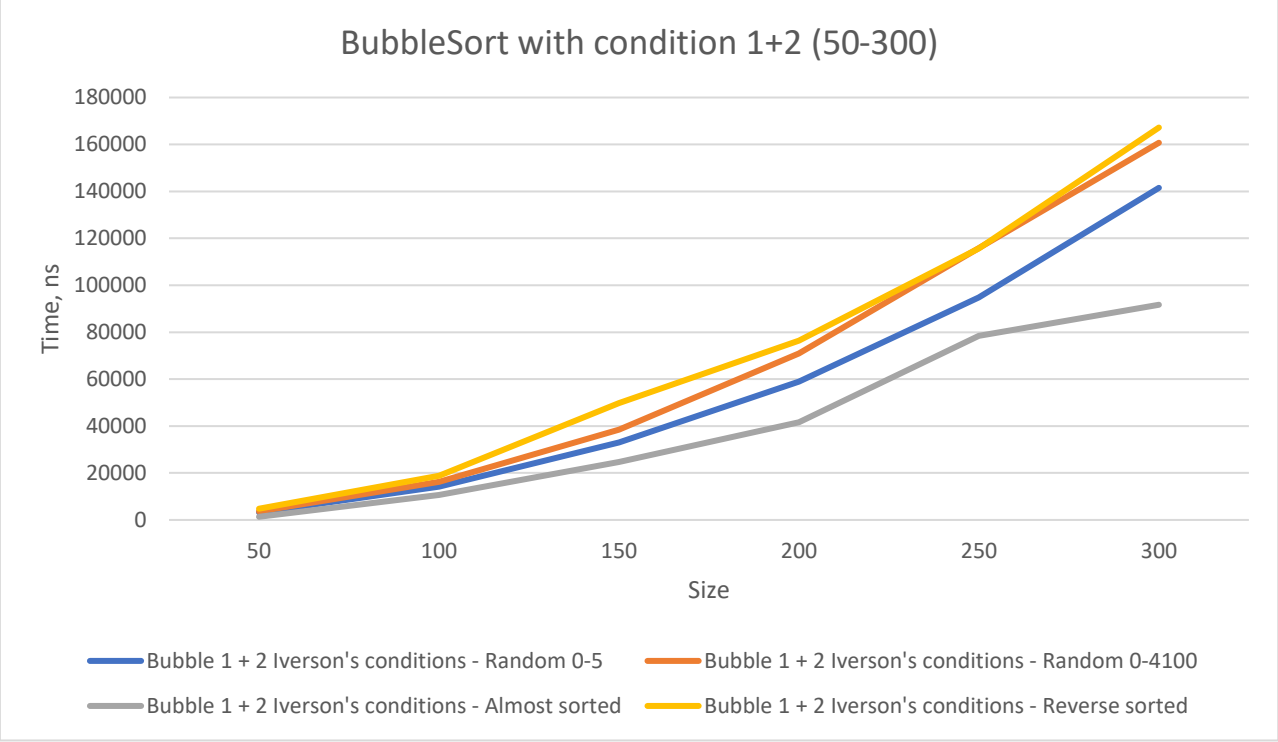


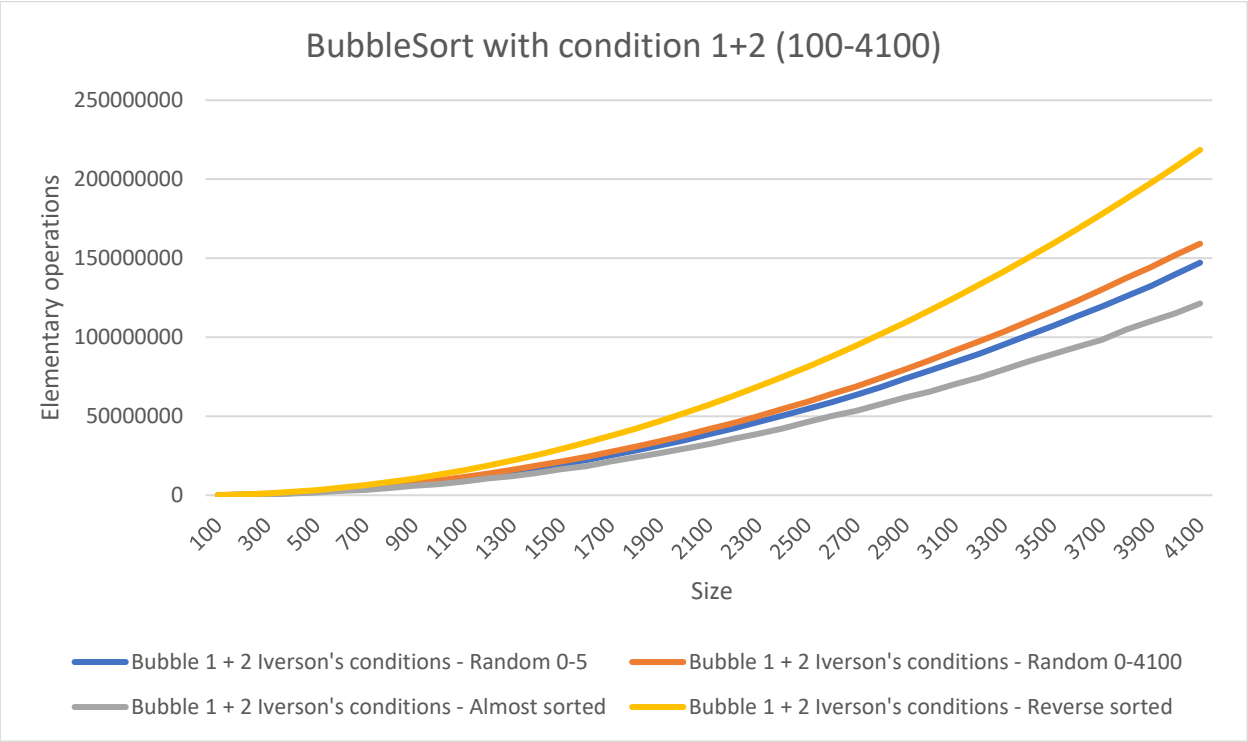
Аналогично обычной сортировке пузырьком.

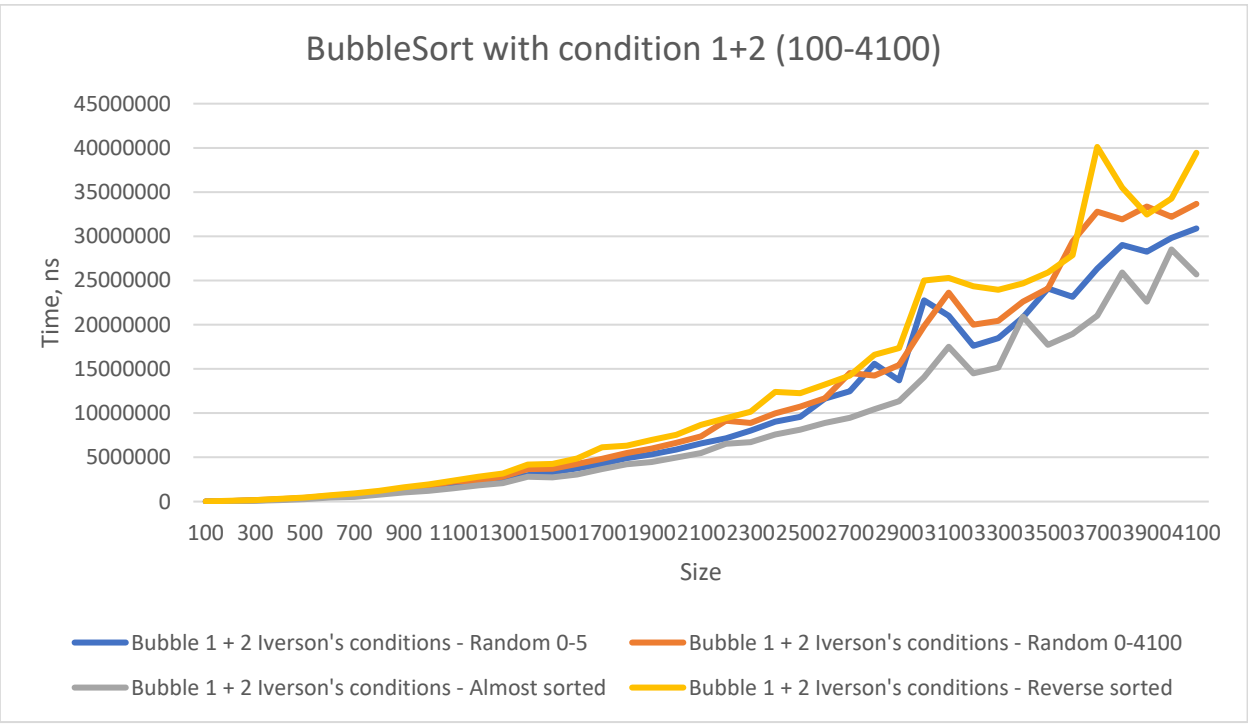
В худшем случае работает медленнее алгоритма без оптимизаций, т.к. постоянно проверяется ещё одно условие.

12. Сортировка пузырьком (1-е и 2-е условие Айверсона).





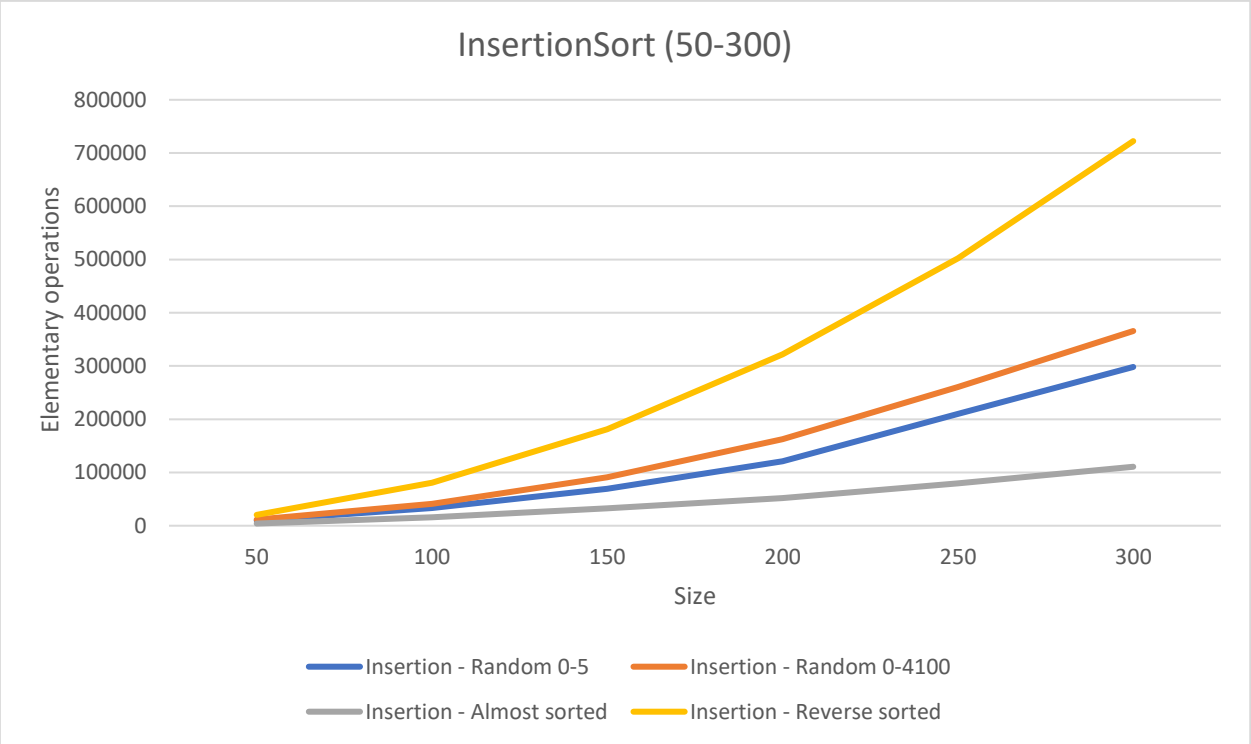


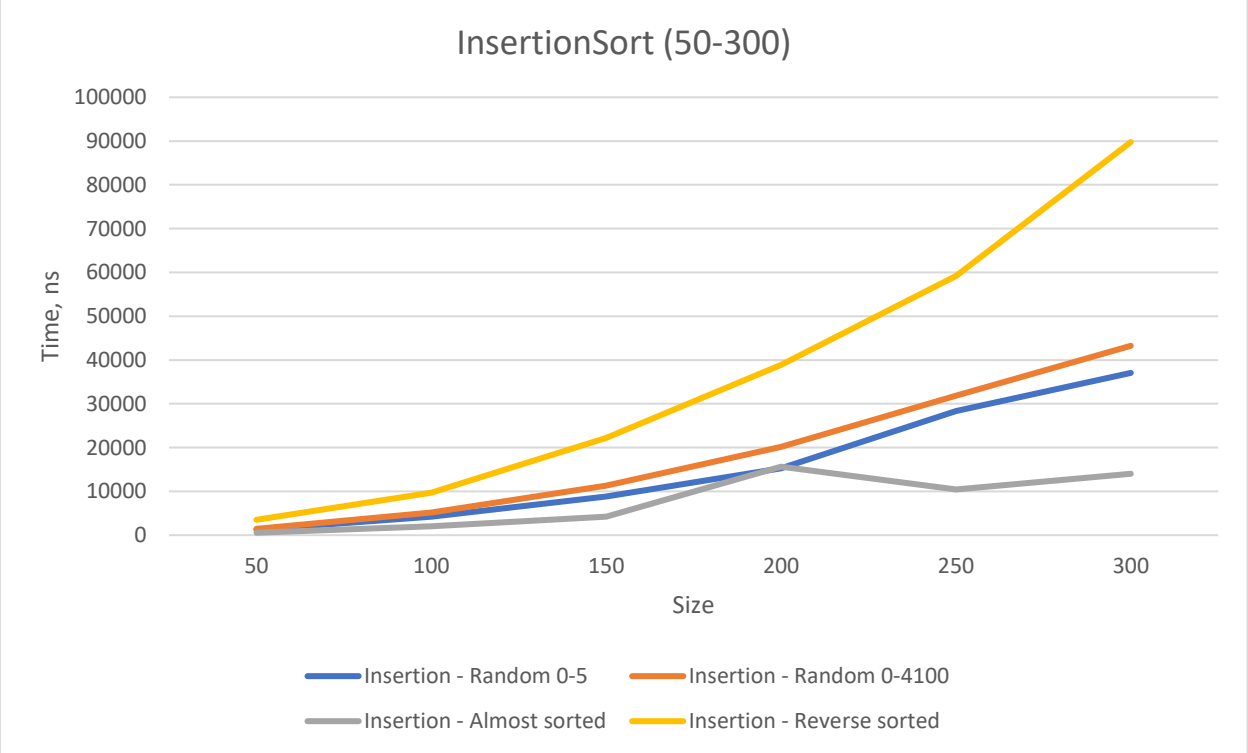


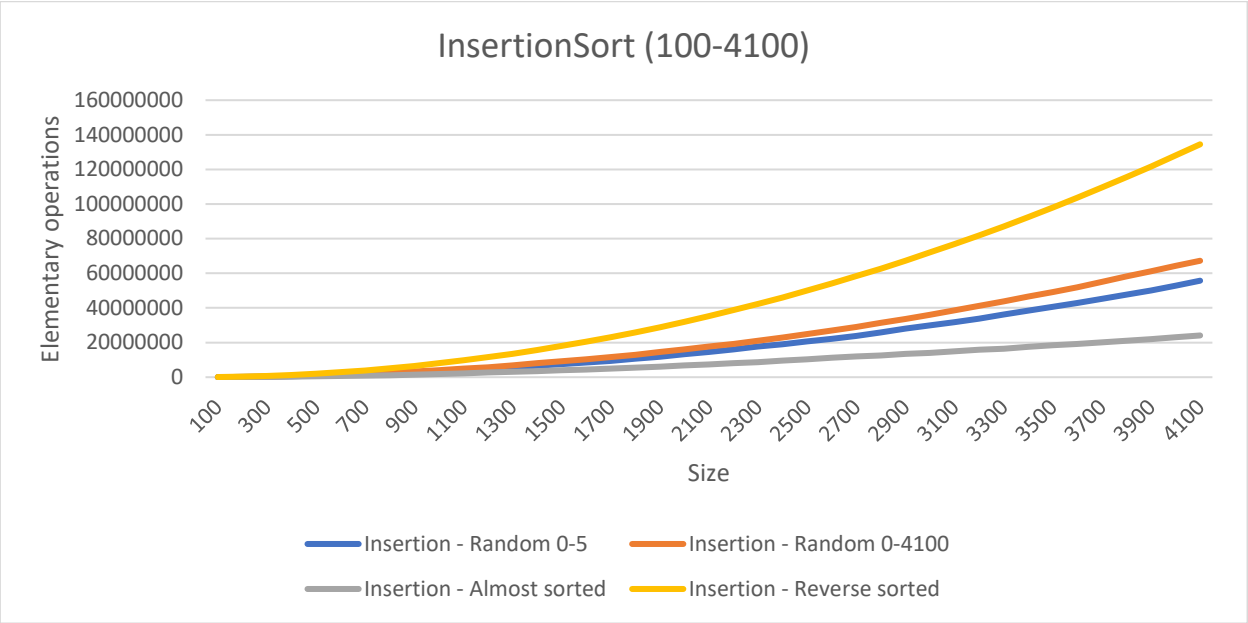
Аналогично обычной сортировке пузырьком.

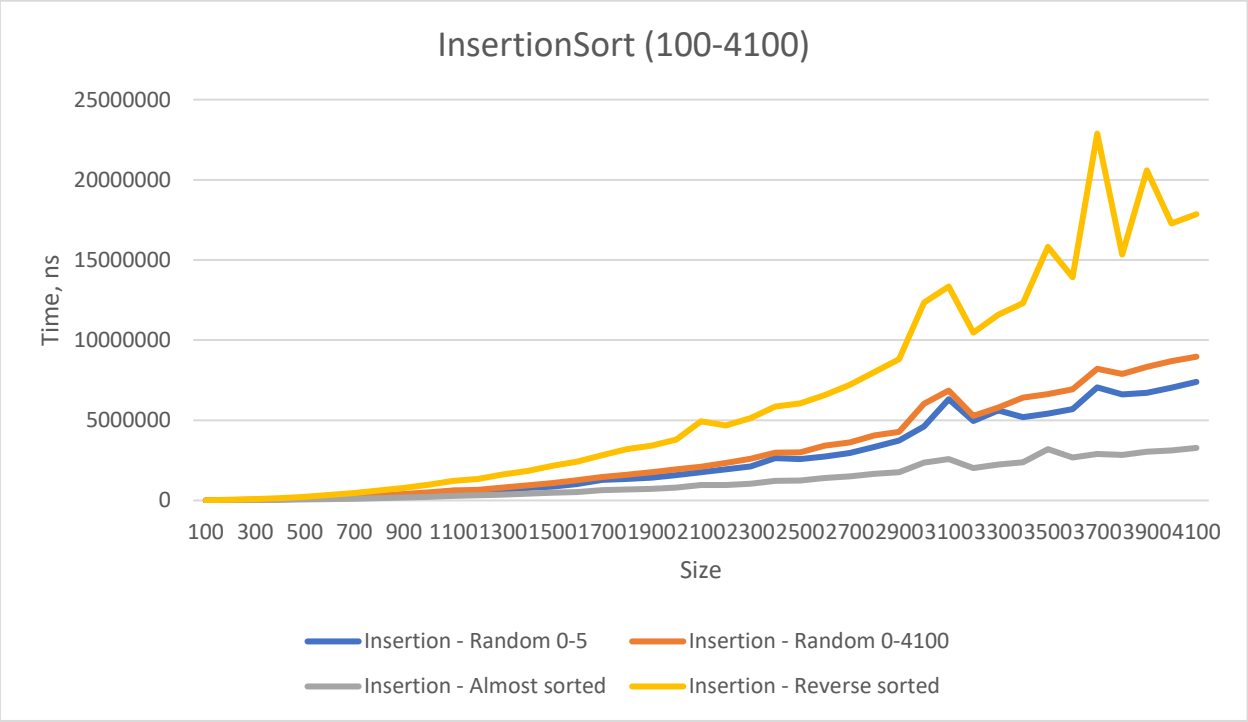
В худшем случае работает медленнее алгоритма без оптимизаций, т.к. выполняются лишние действия, а число проходов цикла не сокращается.

13. Сортировка обычными вставками.







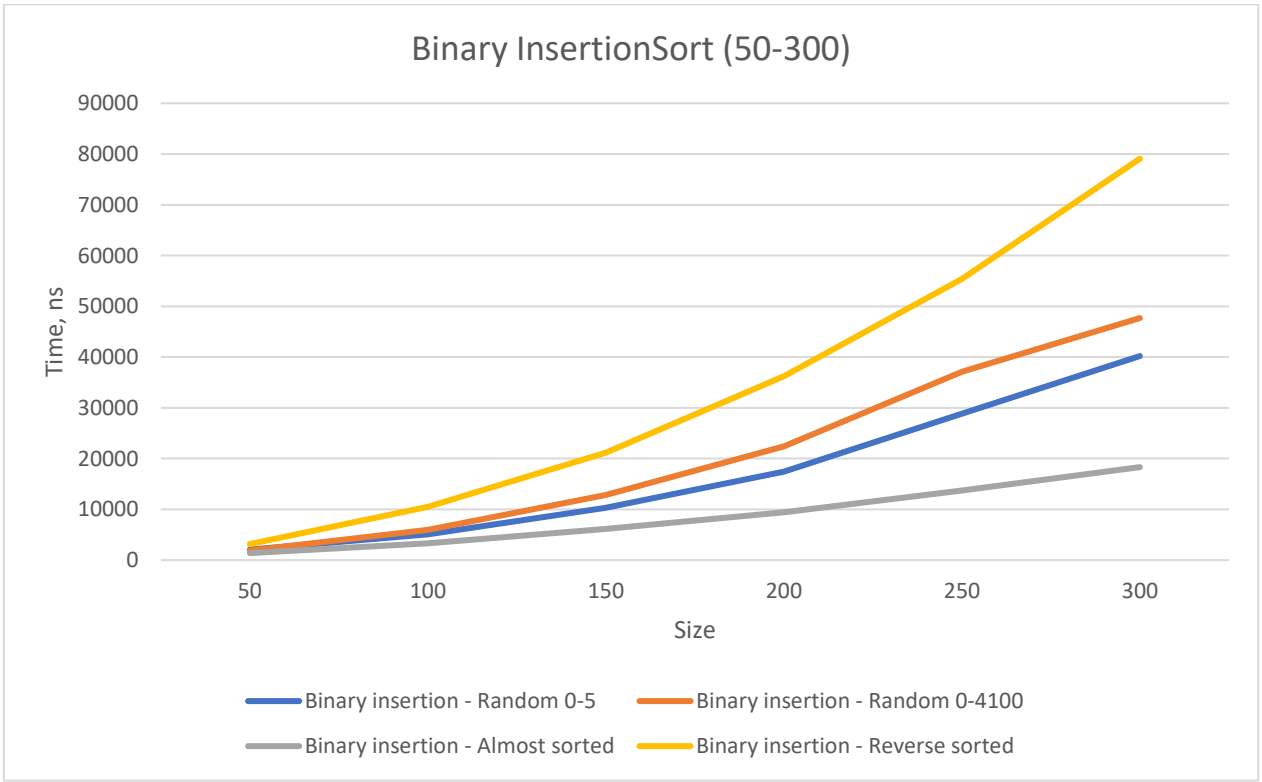


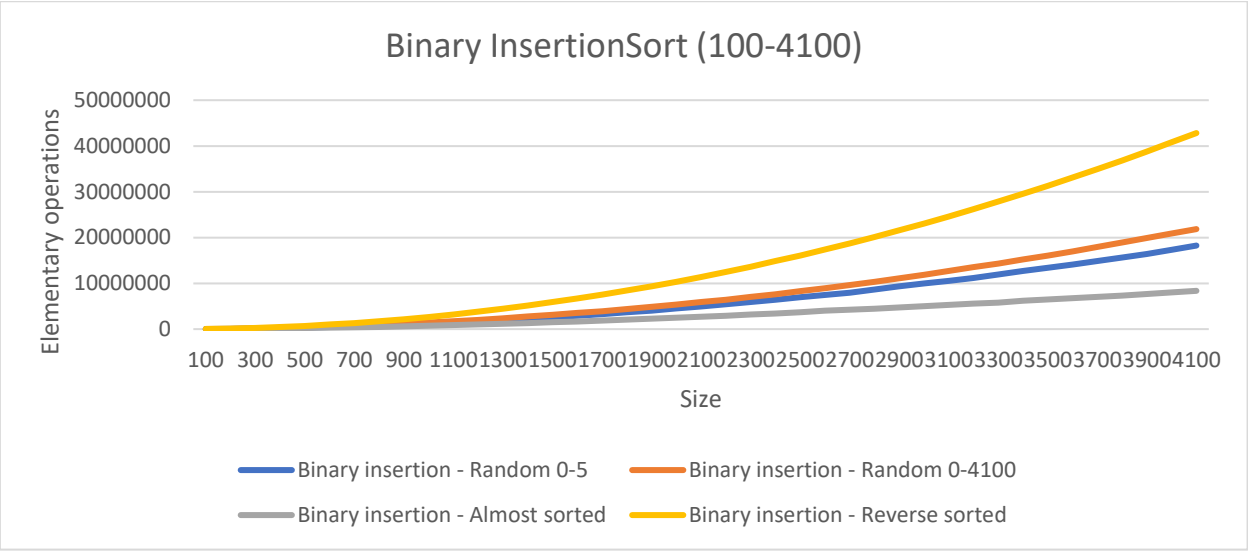
В худшем и среднем случае работает за $O(n^2)$.

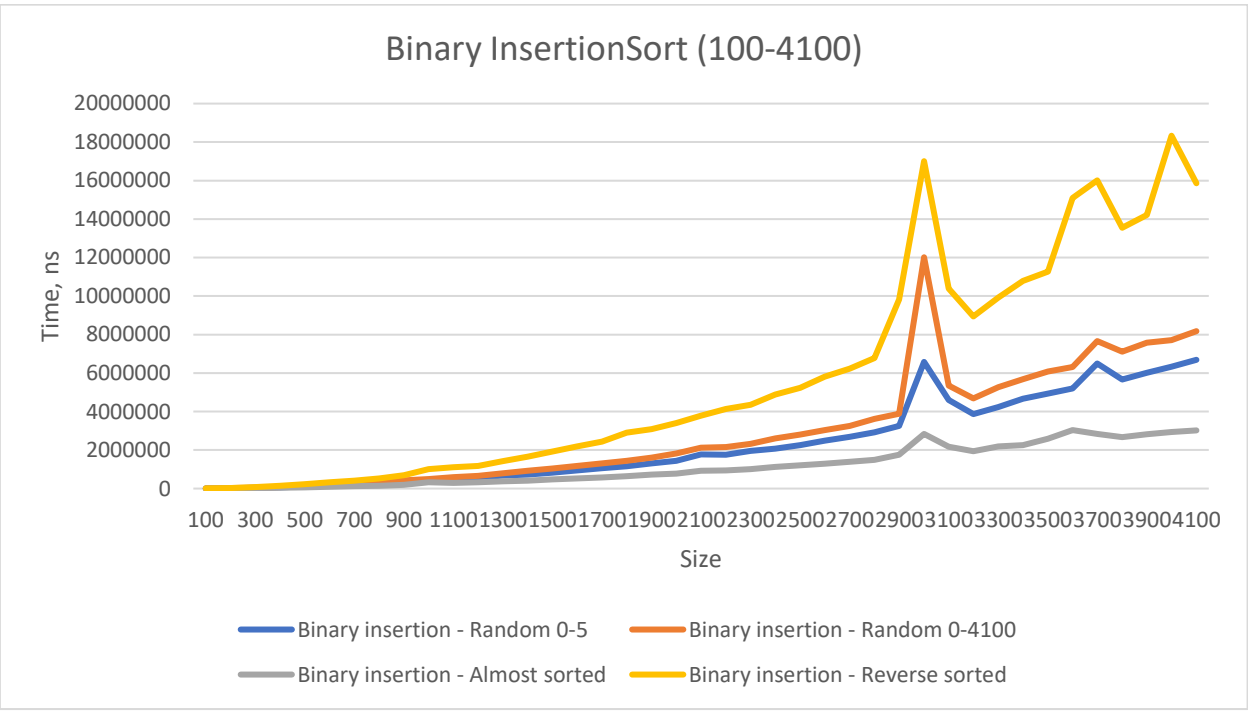
Худшим случаем является обратно отсортированный массив, т.к. для каждого последующего элемента приходится сдвигать отсортированную часть.

Лучшим случаем является отсортированный массив. В данном случае алгоритм отрабатывает за O(n), т.к. просто один раз проходится по всему массиву. Поэтому близкий к лучшему случаю – почти отсортированный массив показывает лучший результат.

14. Сортировка бинарными вставками.



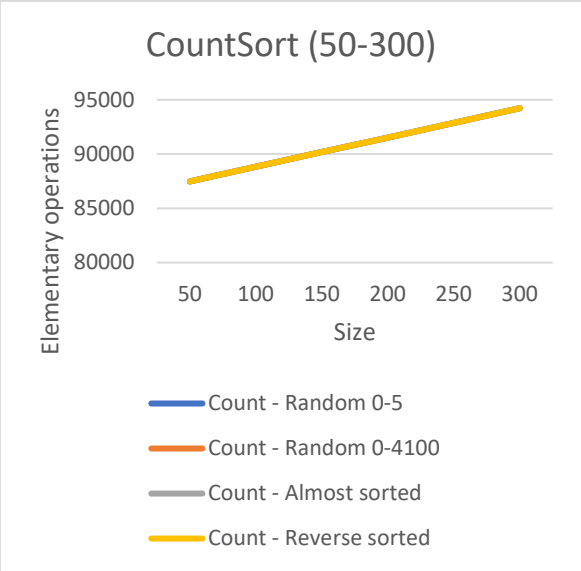


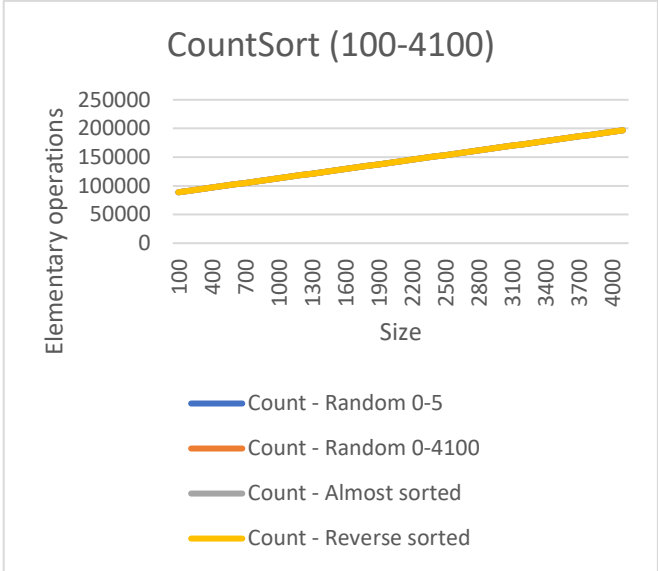


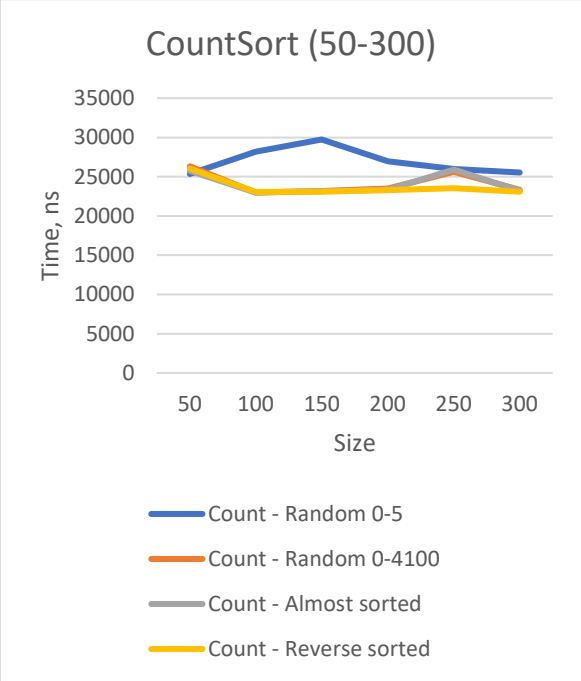
Имеет такую же ассимптотическую сложность, как и сортировка обычными вставками, но работает быстрее, т.к. поиск места для вставки осуществляется за O(log n).

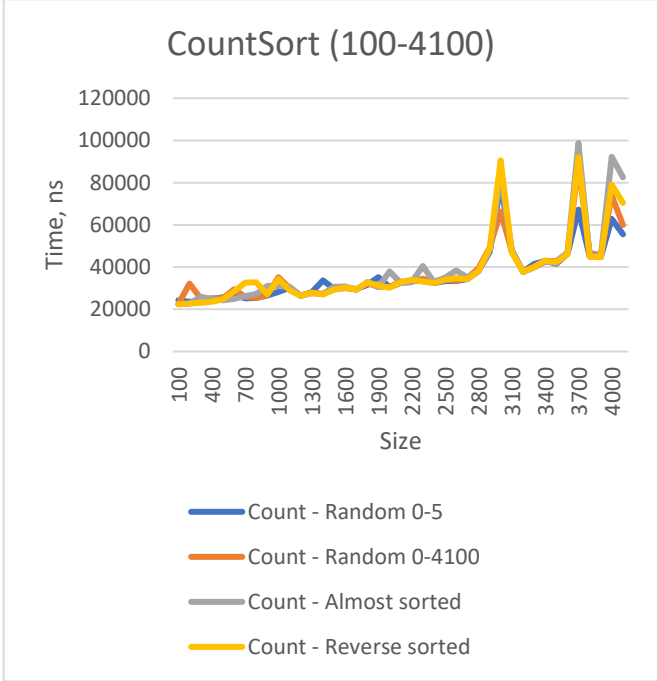
Лучшие и худшие случаи определяются аналогично сортировке обычными вставками.

15. Сортировка подсчётом.



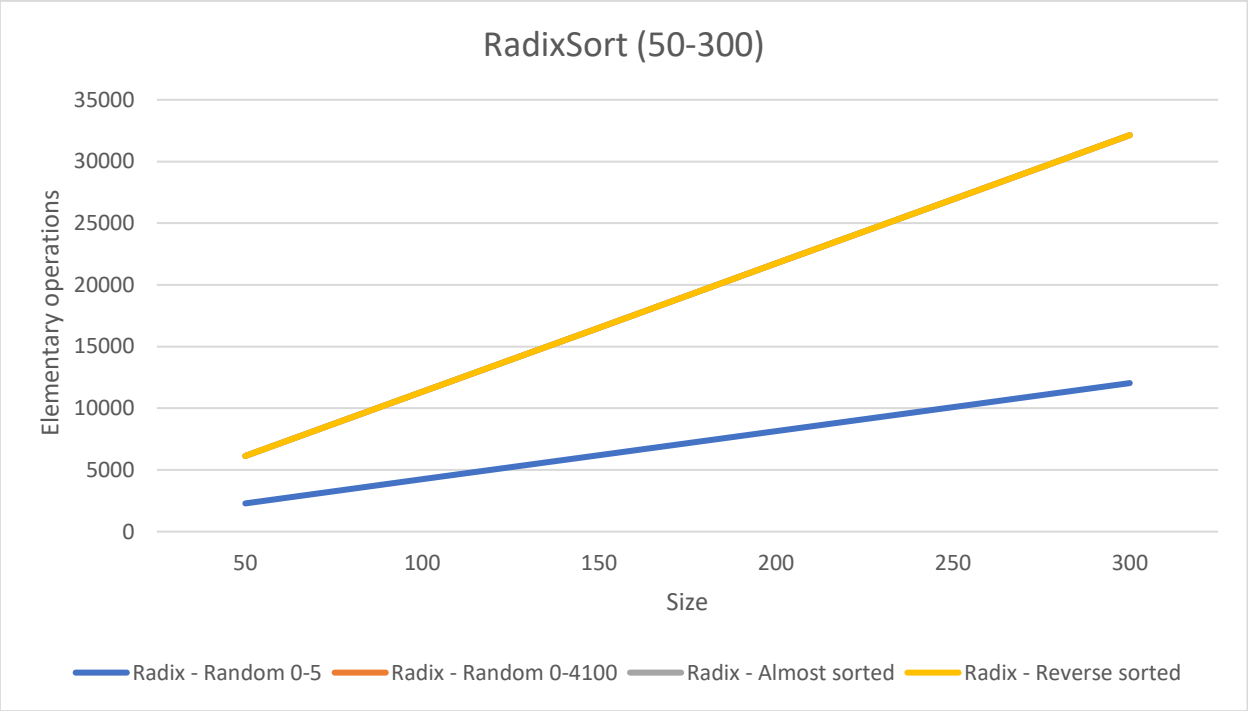


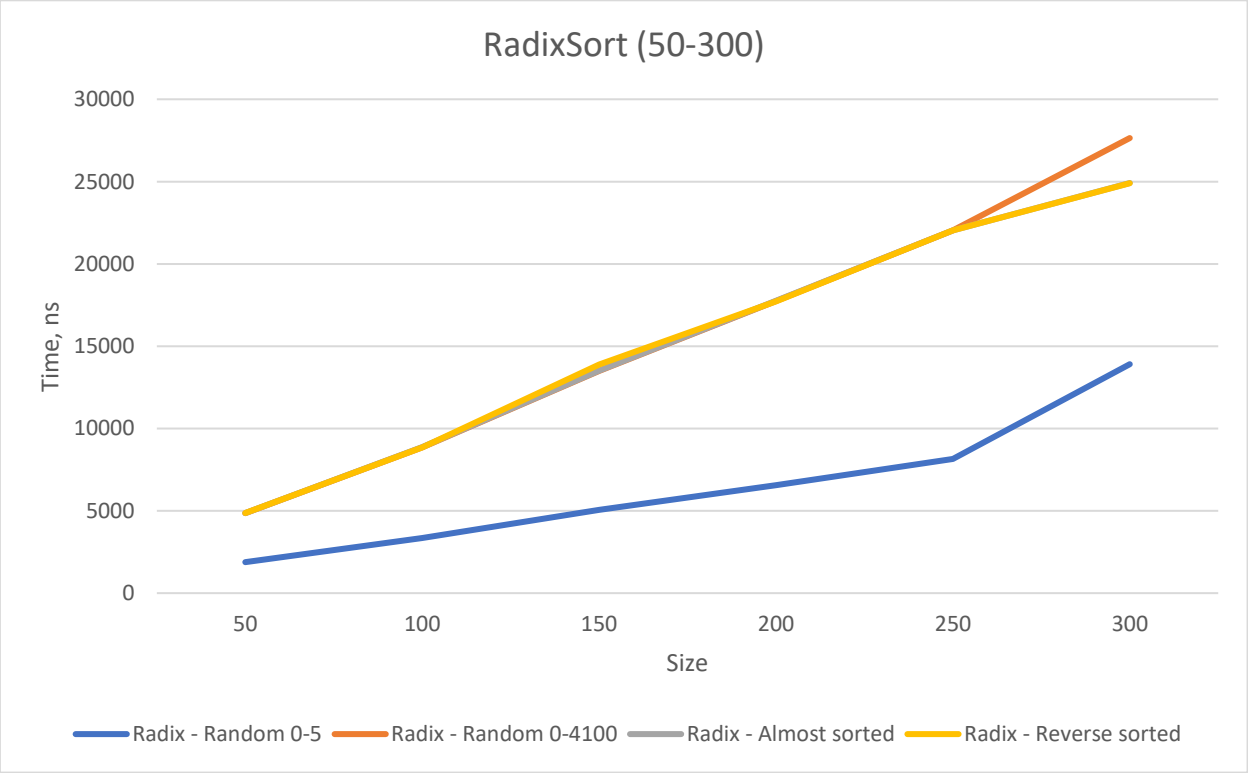


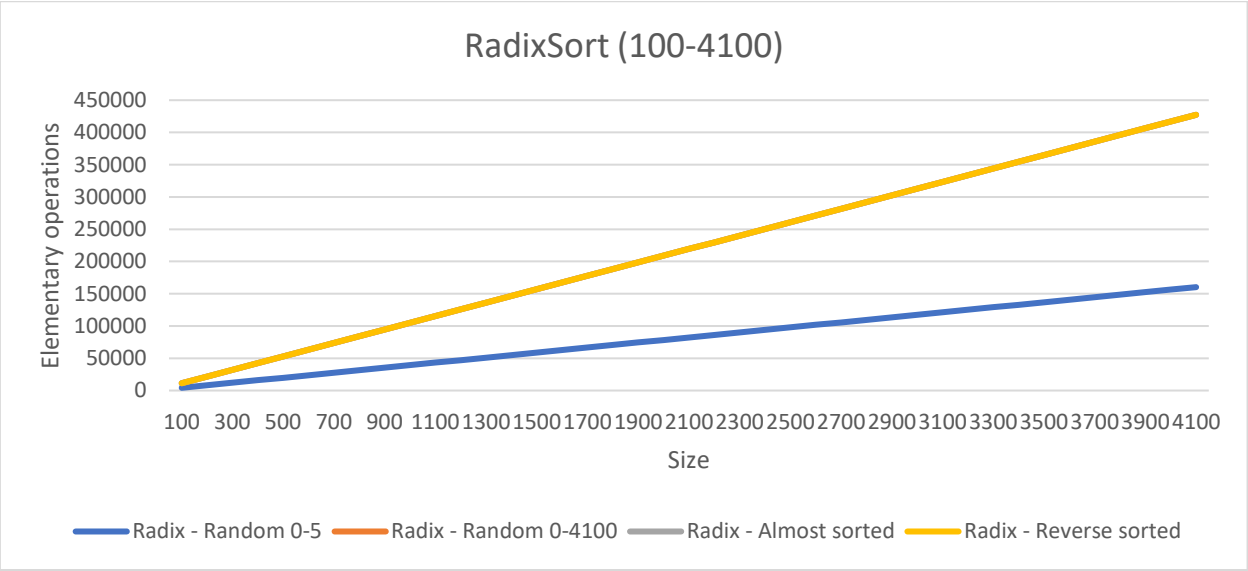


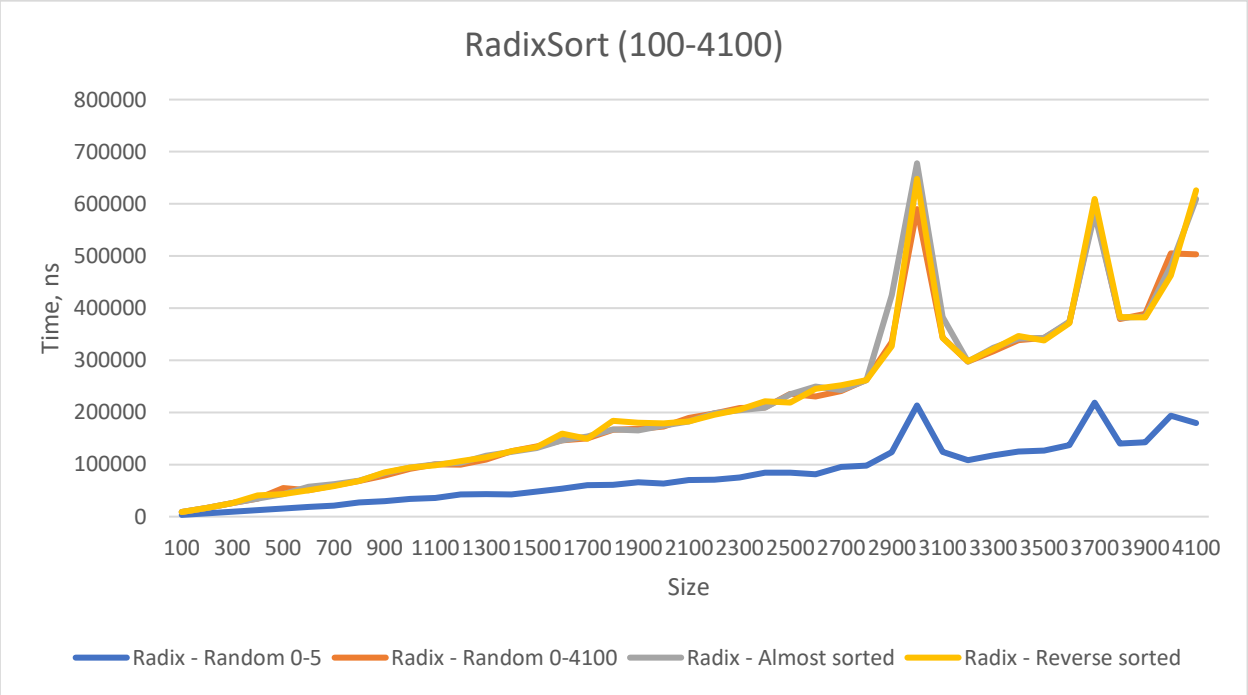
Сортировка является линейной (Θ(n)). Выполняется за линию внезависимости от входных данных (при условии, что значения находятся в едином диапазоне). На графике есть случайные выбросы, но при достаточно большом наборе он выравнивается в линию.

16. Цифровая сортировка (LSD, по основанию 10).





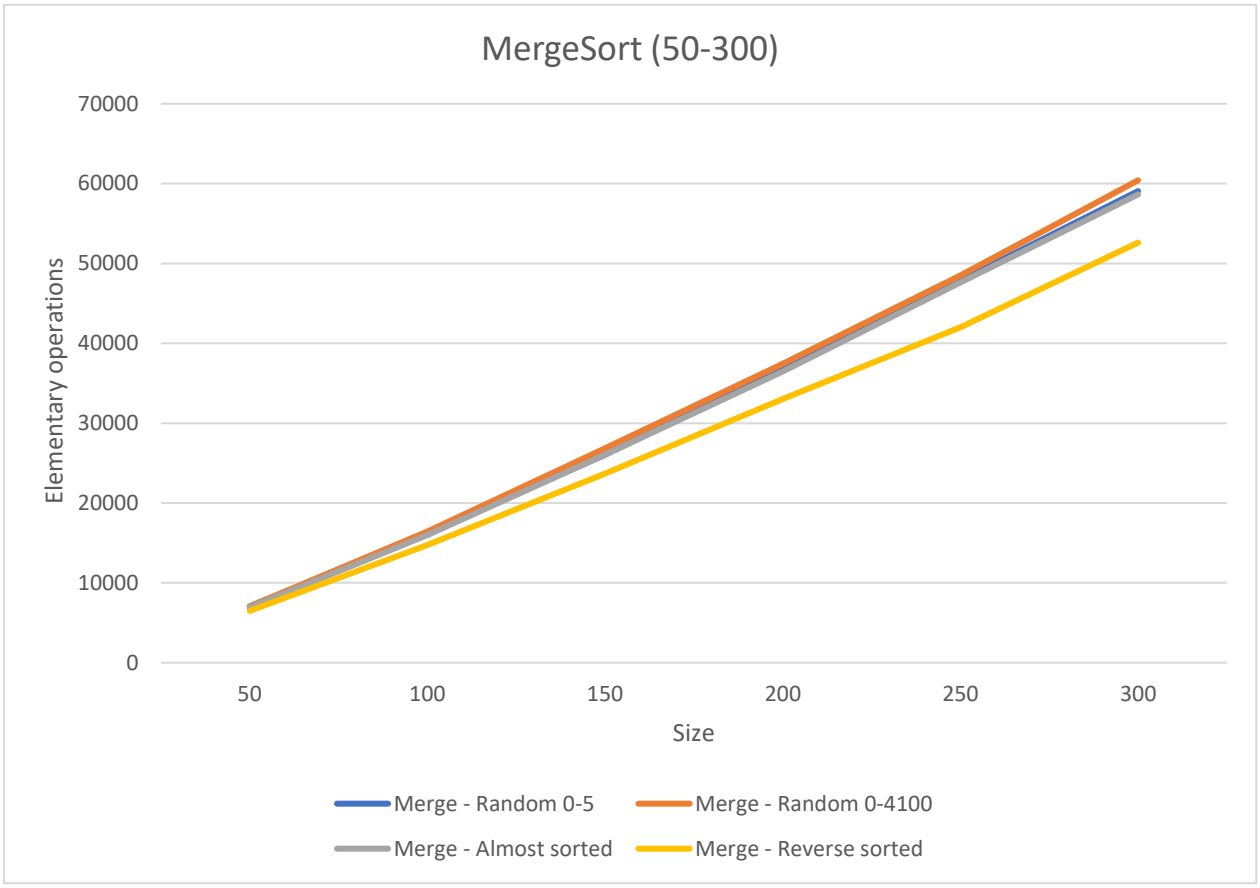


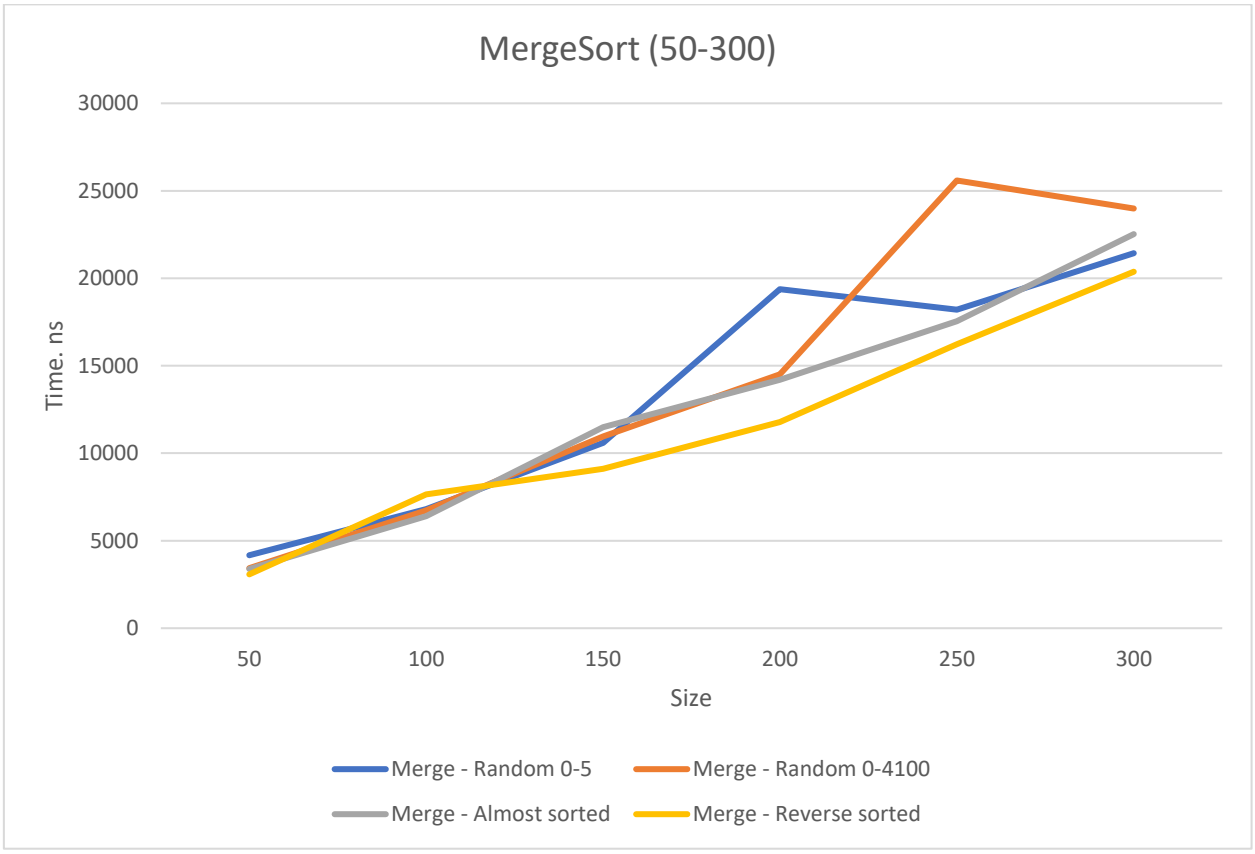


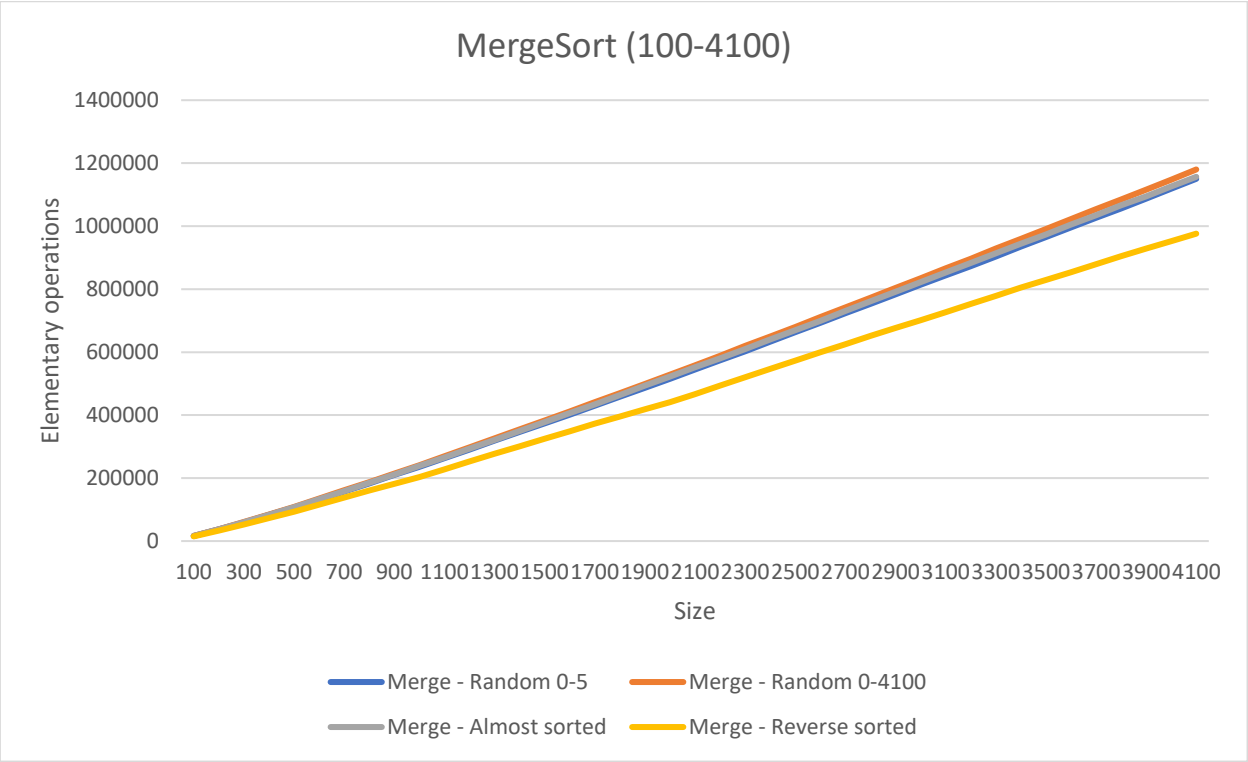
Работает за Θ(m * (n + k)), где m – число разрядов, n – размер массива, k – основание.

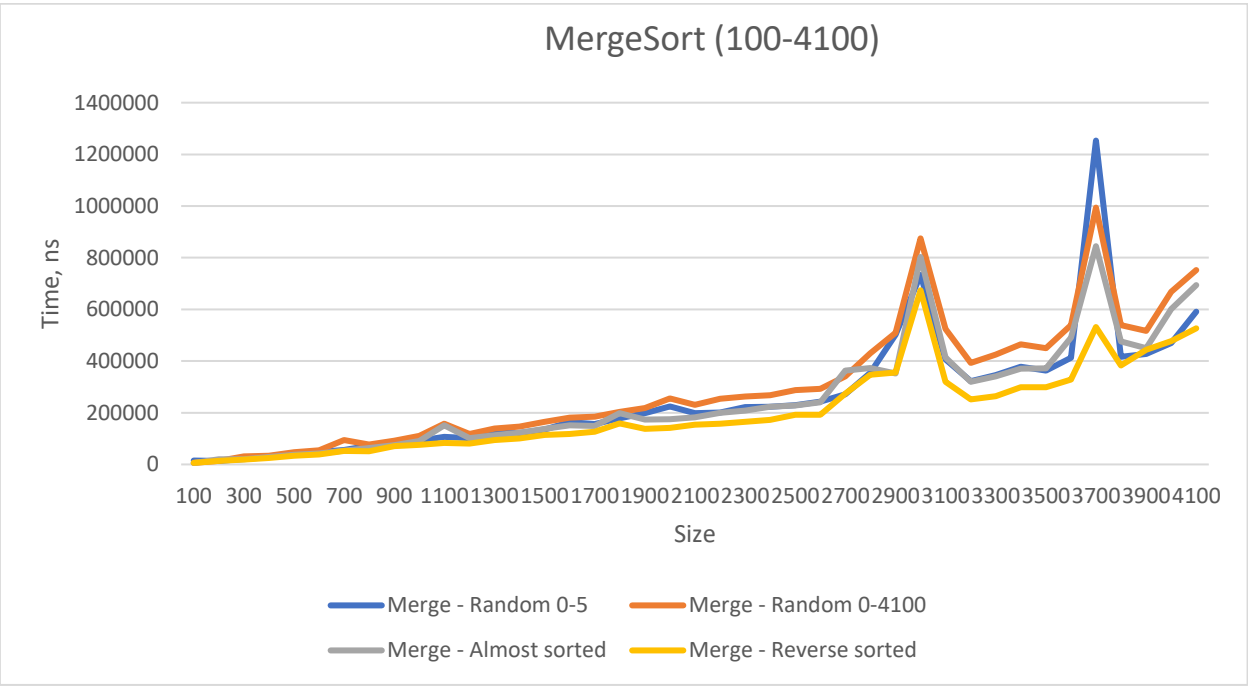
По данным графикам лучше всего цифровая сортировка отрабатывает на массивах из случайных чисел в диапазоне от 0 до 5, т.к. в них всего по одному разряду.

17. Сортировка слиянием.





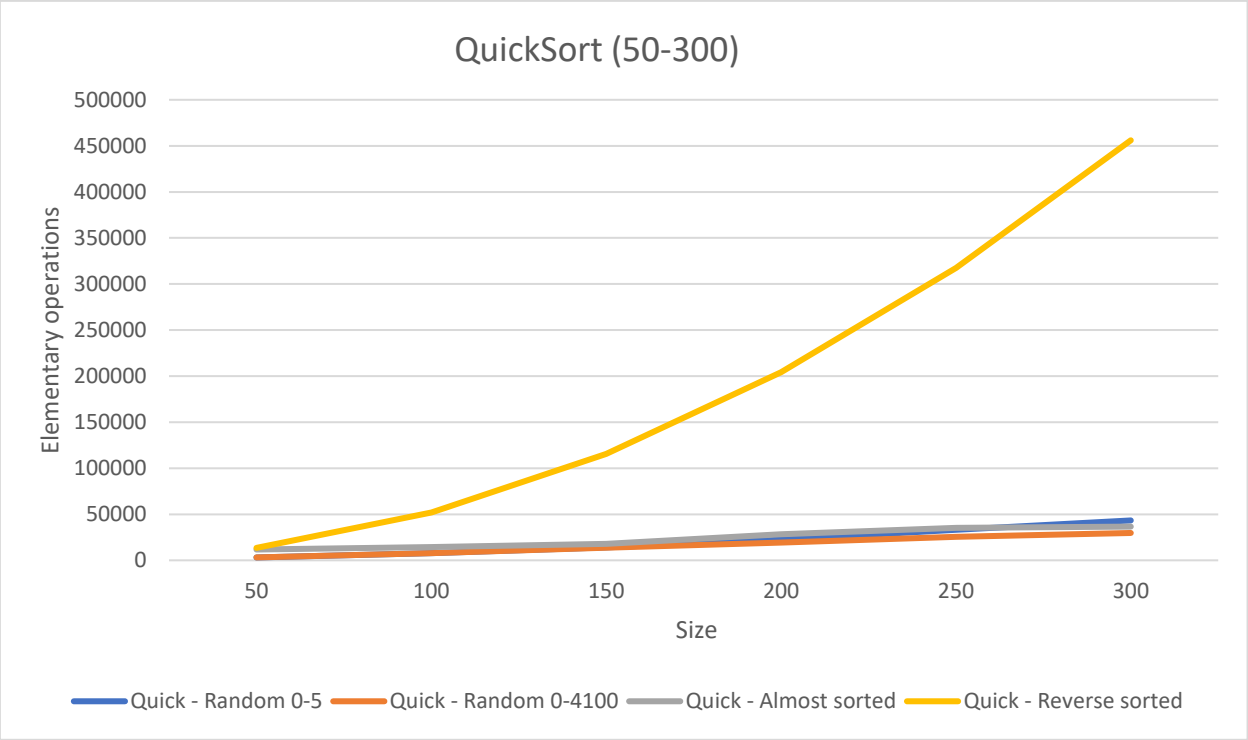


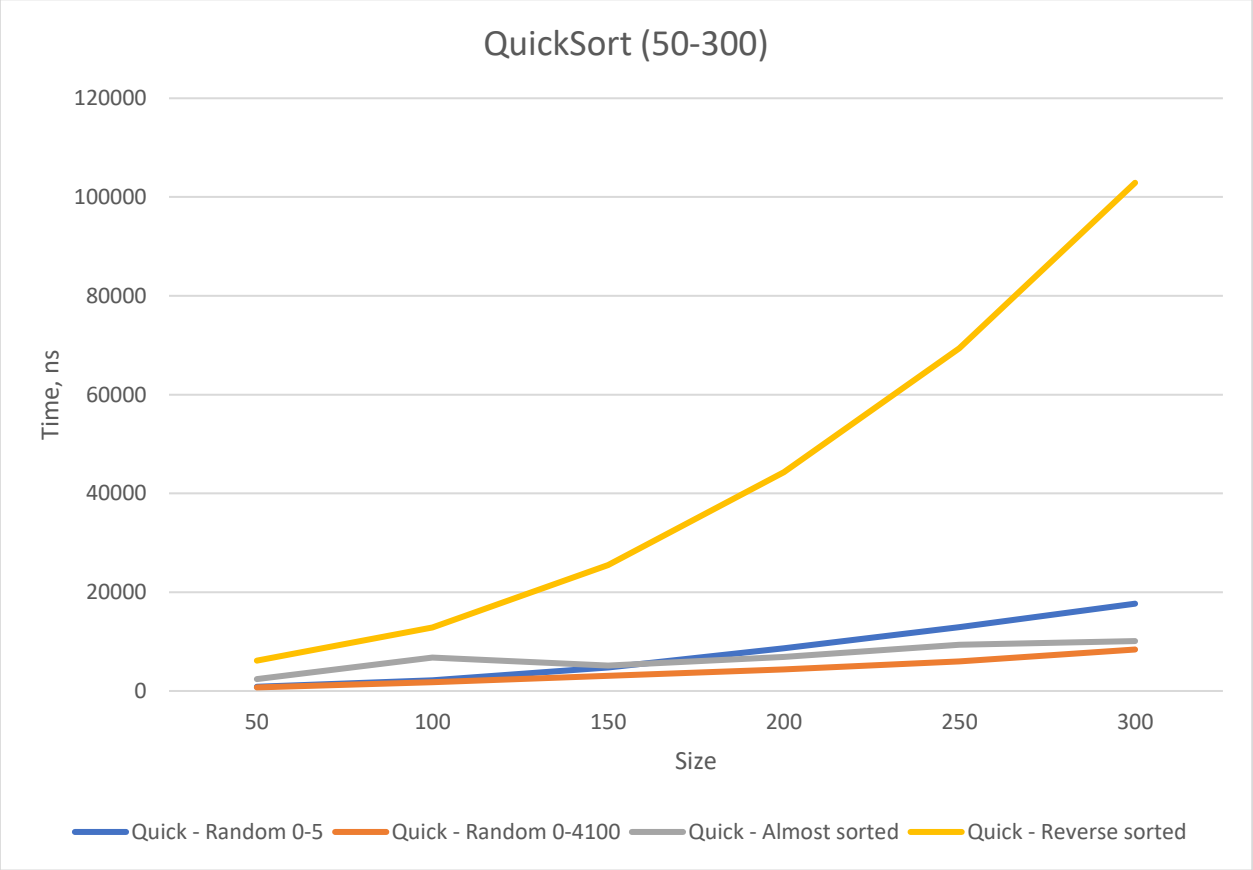


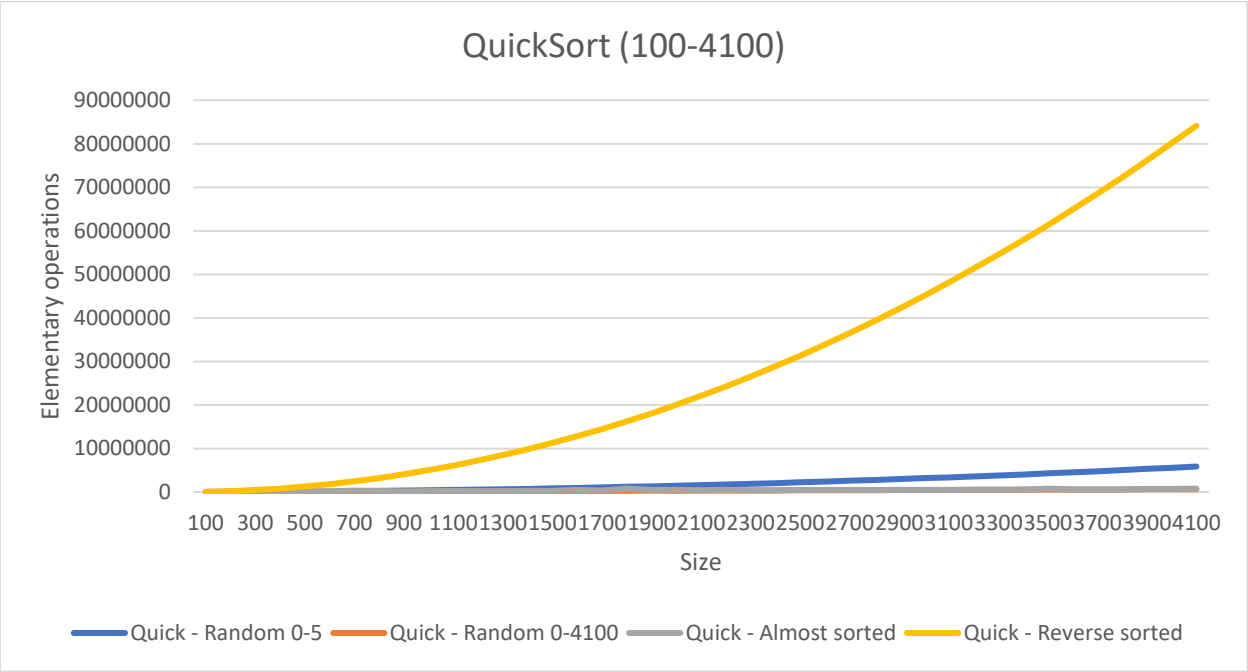
Работает практически одинаково на всех входных данных ($\Theta(n \log n)$).

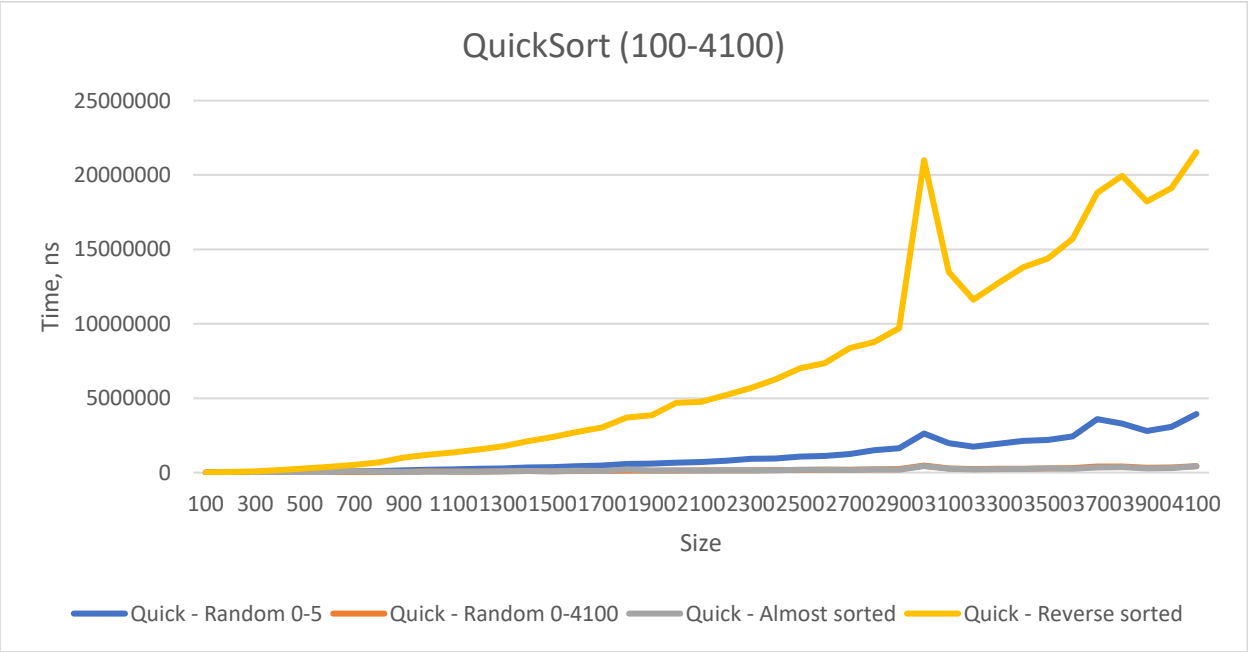
Обратно отсотрированный массив отрабатывает чуть быстрее в данной реализации, т.к. при слиянии пары подмассивов все элементы одного подмассива >= всех элементов другого подмассива, поэтому элементы сравниваются, пока не закончится подмассив с меньшими элементами, а остальные дописываются без сравнения.

18. Быстрая сортировка.









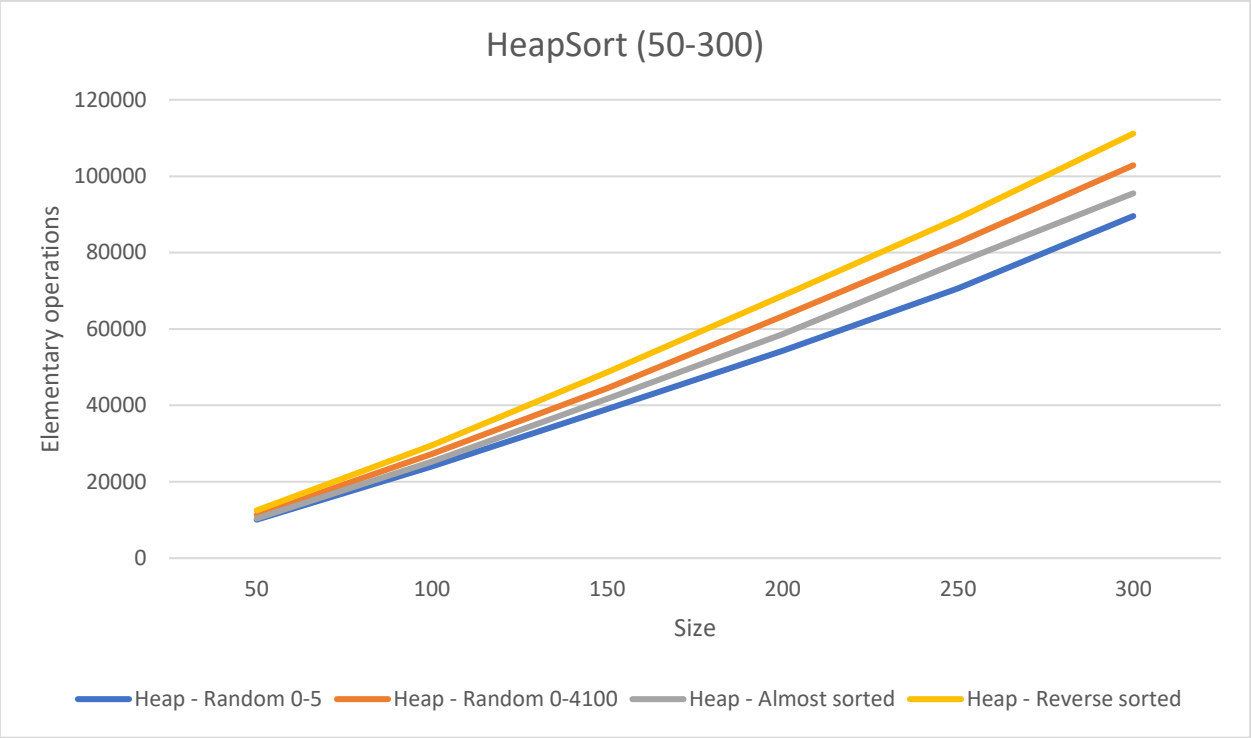
В среднем работает за O(n log n).

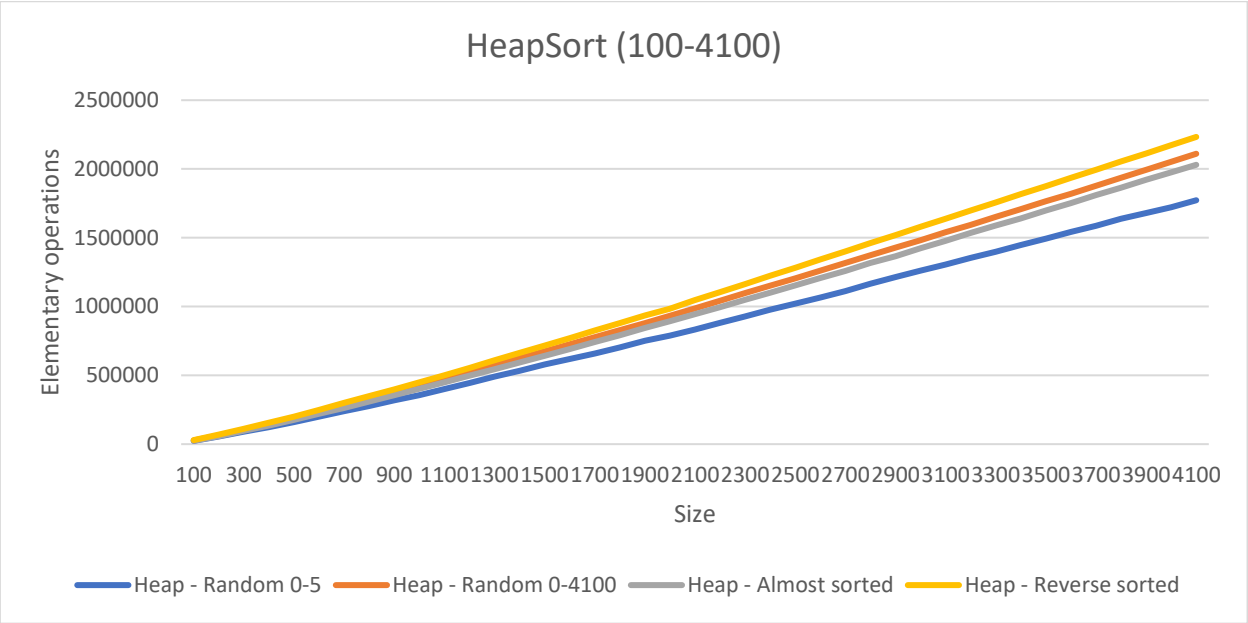
В худшем случае за $O(n^2)$. Худшим случаем является обратно отсортированный массив, т.к. опорный элемент – первый в массиве, следовательно, на каждой итерации происходит n обменов, а размер уменьшается на 1 – сложность вырождается в квадрат.

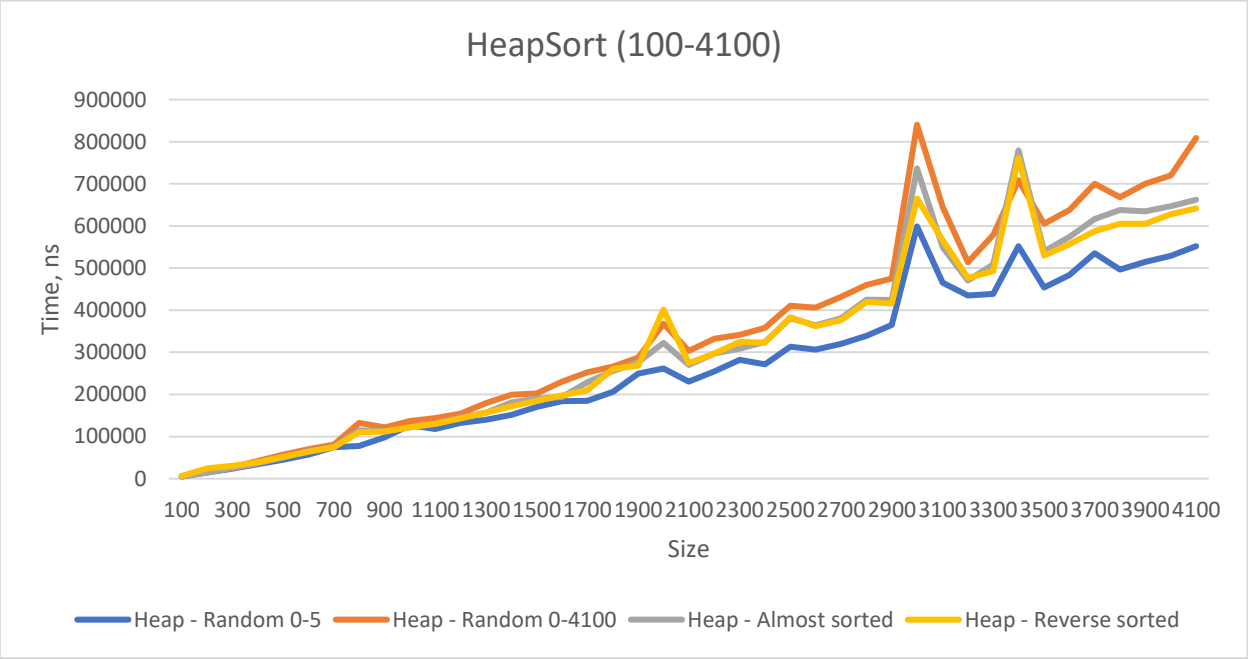
В массиве из случайных элементов в диапазоне от 0 до 5 высокая вероятность попасть в максимум. Таким образом приближаемся к худшему случаю.

Почти отсортированный массив в связи с реализацией не является худшим случае, хотя мы почти на каждой итерации попадаем в минимум, т.к. в таком случае почти не происходит обменов.

19. Сортировка кучей.





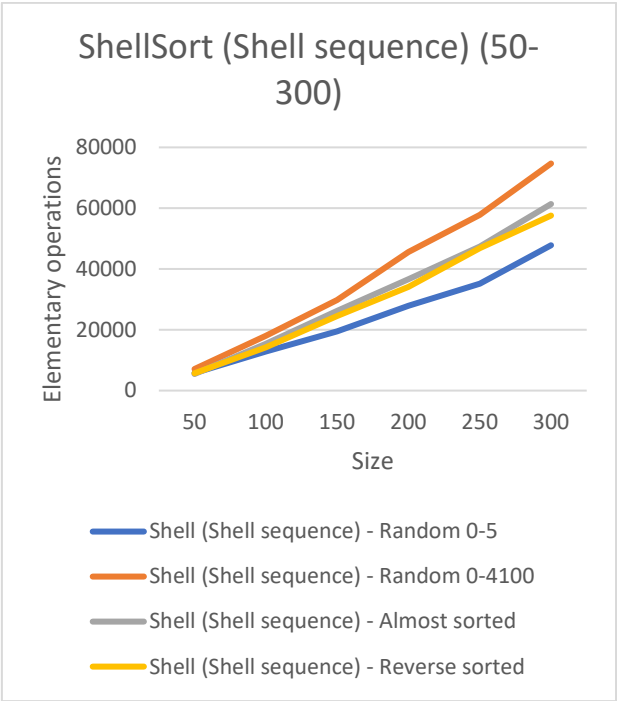


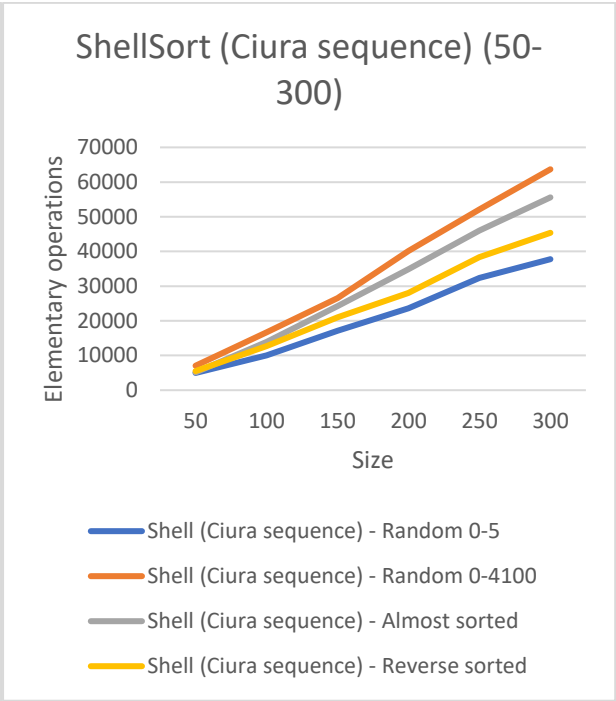
Работает практически одинаково на любых входных данных (O(n log n)).

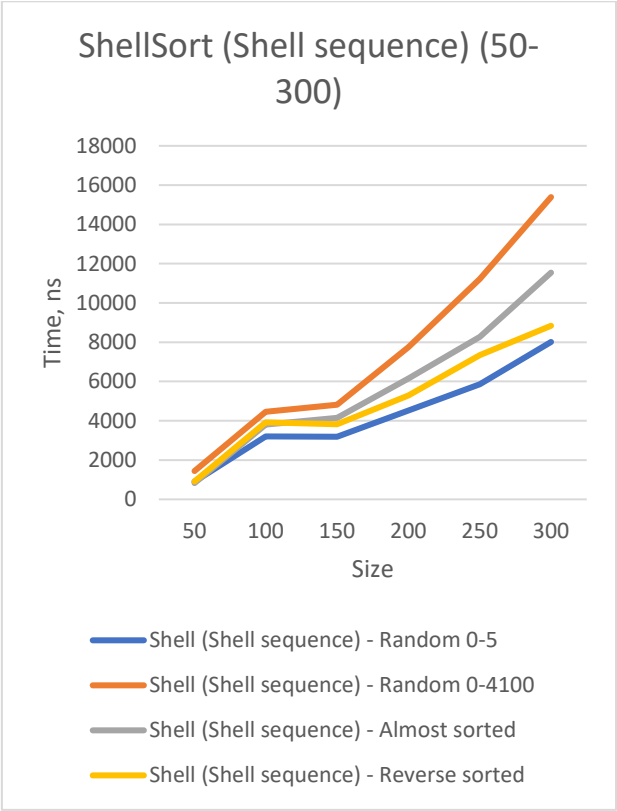
В лучшем случае все элементы равны, тогда heapify отрабатывает за O(1), а значит сортировка отрабатывает за O(n). В массиве из случайных чисел в диапазоне от 0 до 5 многократно повторяются одинаковые элементы, поэтому такой набор данных немного ближе остальных к лучшему случаю, из-за чего и показывает лучший результат.

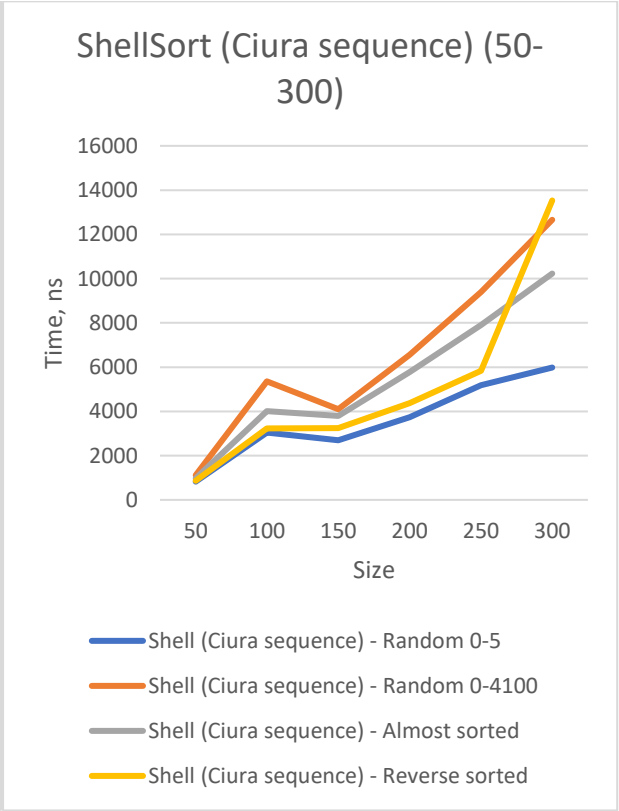
Обратно отсортированный массив отрабатывает чуть хуже остальных, т.к. повышается сложность построения кучи: приходится делать относительно много обменов.

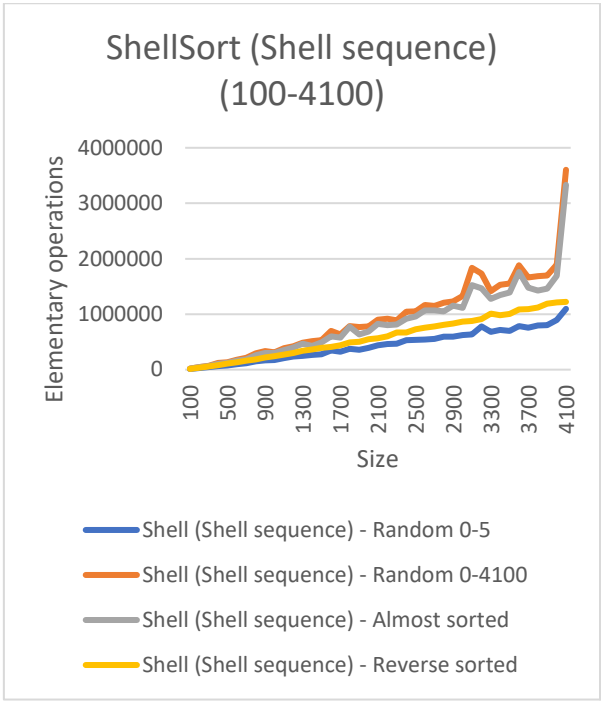
20. Сортировка Шелла

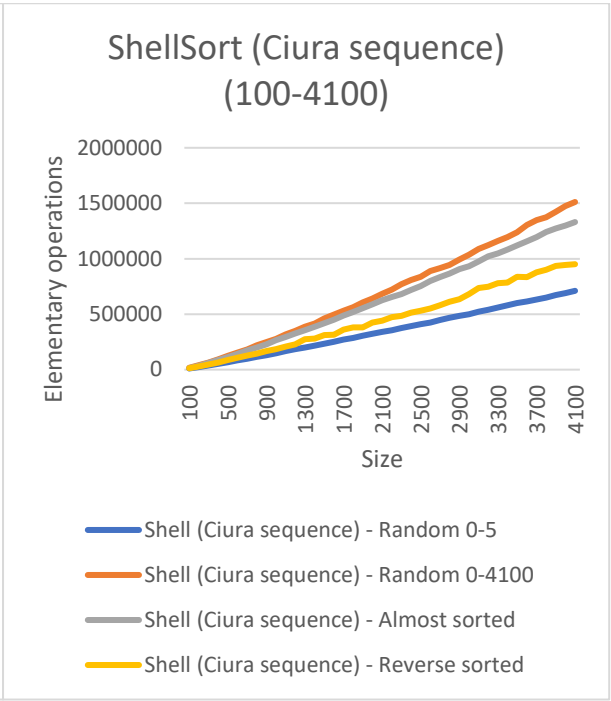


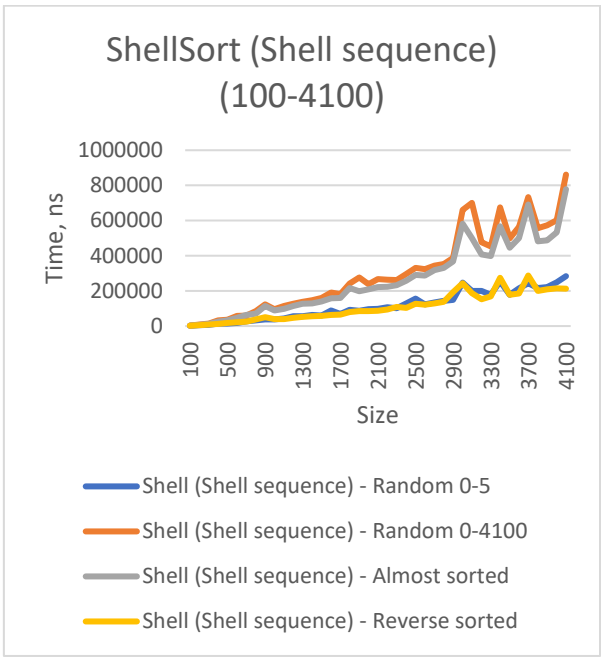


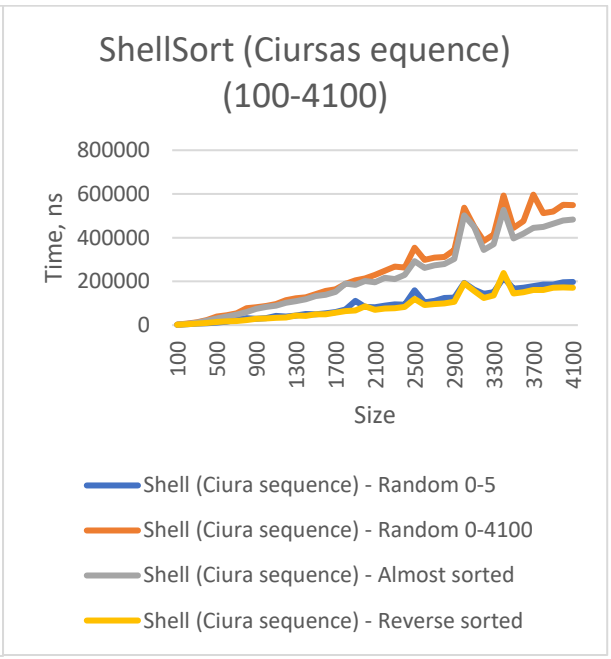





ShellSort (Shell sequence)
(100-4100)
Elementary operations
4000000
3000000
2000000
1000000
0
100
500
900
1300
1700
2100
2500
2900
3300
3700
4100
Size
Shell (Shell sequence) - Random 0-5
Shell (Shell sequence) - Random 0-4100
Shell (Shell sequence) - Almost sorted
Shell (Shell sequence) - Reverse sorted


ShellSort (Ciura sequence)
(100-4100)
Elementary operations
2000000
1500000
1000000
500000
0
100
500
900
1300
1700
2100
2500
2900
3300
3700
4100
Size
Shell (Ciura sequence) - Random 0-5
Shell (Ciura sequence) - Random 0-4100
Shell (Ciura sequence) - Almost sorted
Shell (Ciura sequence) - Reverse sorted


ShellSort (Shell sequence)
(100-4100)
Time, ns
1000000
800000
600000
400000
200000
0
100
500
900
1300
1700
2100
2500
2900
3300
3700
4100
Size
Shell (Shell sequence) - Random 0-5
Shell (Shell sequence) - Random 0-4100
Shell (Shell sequence) - Almost sorted
Shell (Shell sequence) - Reverse sorted


ShellSort (Ciursas equence)
(100-4100)
Time, ns
800000
600000
400000
200000
0
100
500
900
1300
1700
2100
2500
2900
3300
3700
4100
Size
Shell (Ciura sequence) - Random 0-5
Shell (Ciura sequence) - Random 0-4100
Shell (Ciura sequence) - Almost sorted
Shell (Ciura sequence) - Reverse sorted