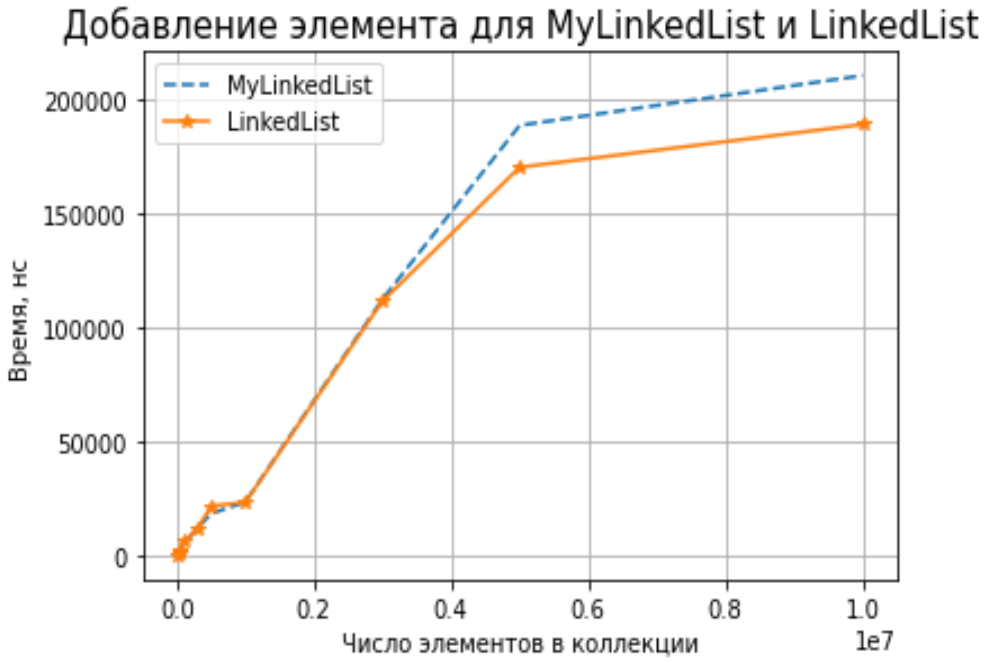
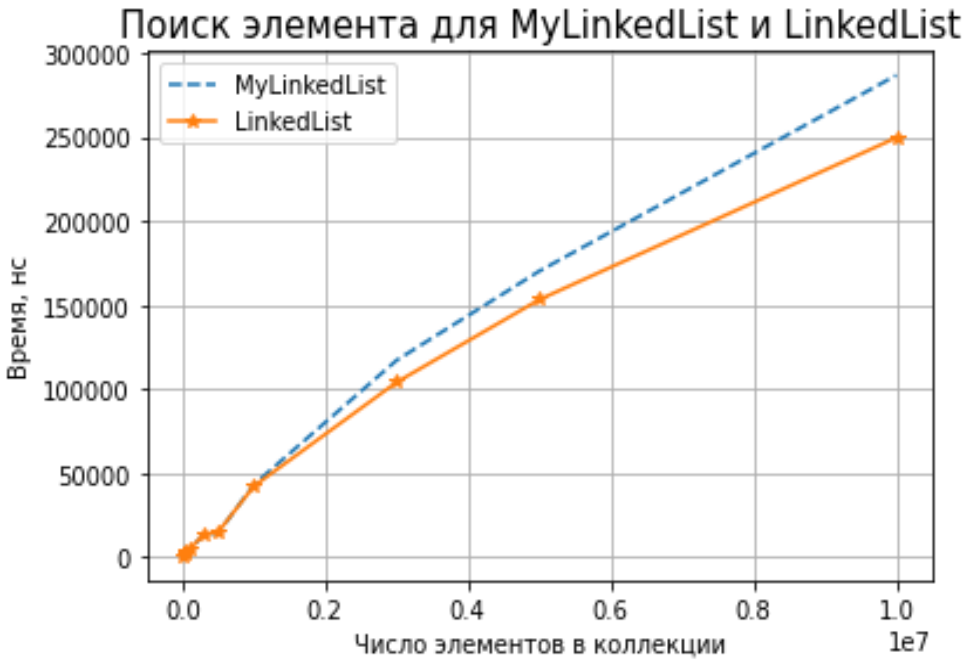
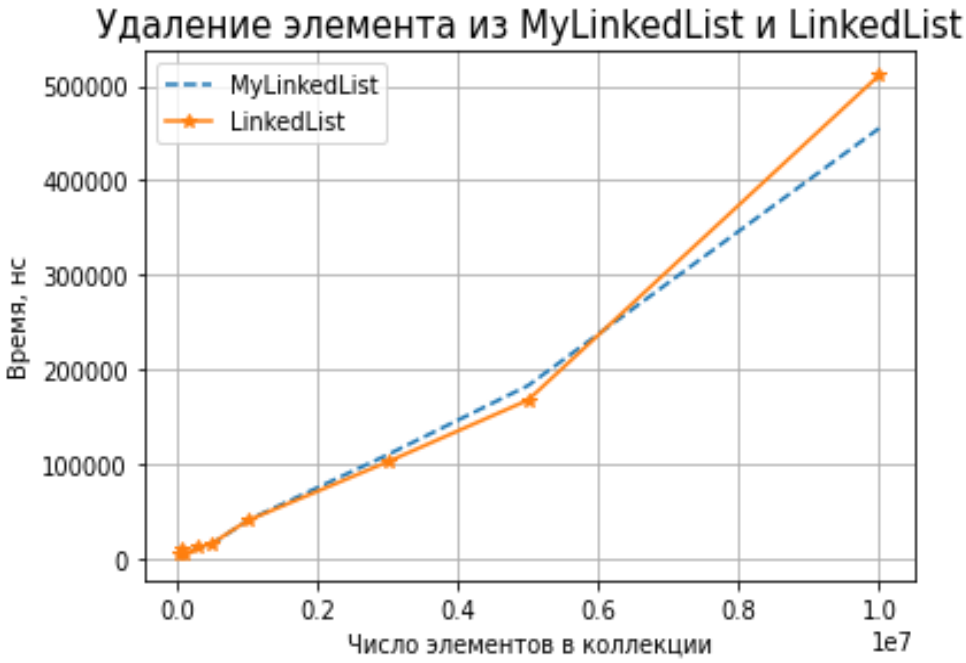
Сравнение производительности MyLinkedList со стандартным LinkedList.

Для сравнения коллекций был взят массив целочисленных значений для создания коллекций с различным числом элементов 10000, 30000, 50000, 100000, 300000, 500000, 1000000, 3000000, 5000000, 10000000, а также было применено усреднение для каждого из значений в массиве с целью повторяемости результатов.



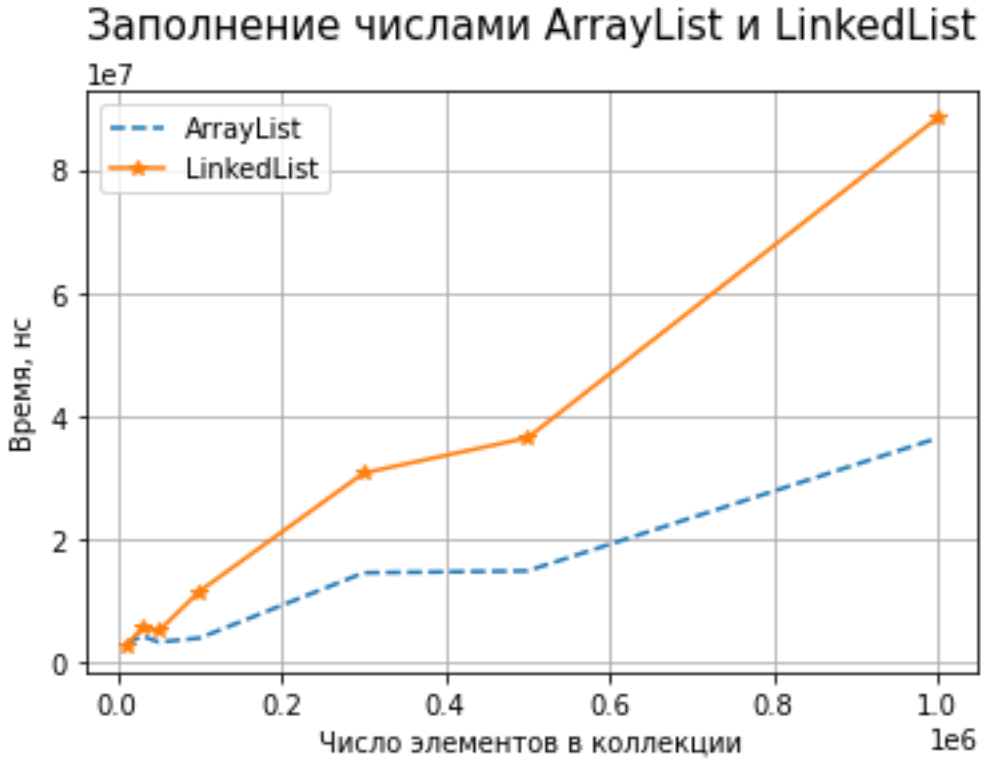




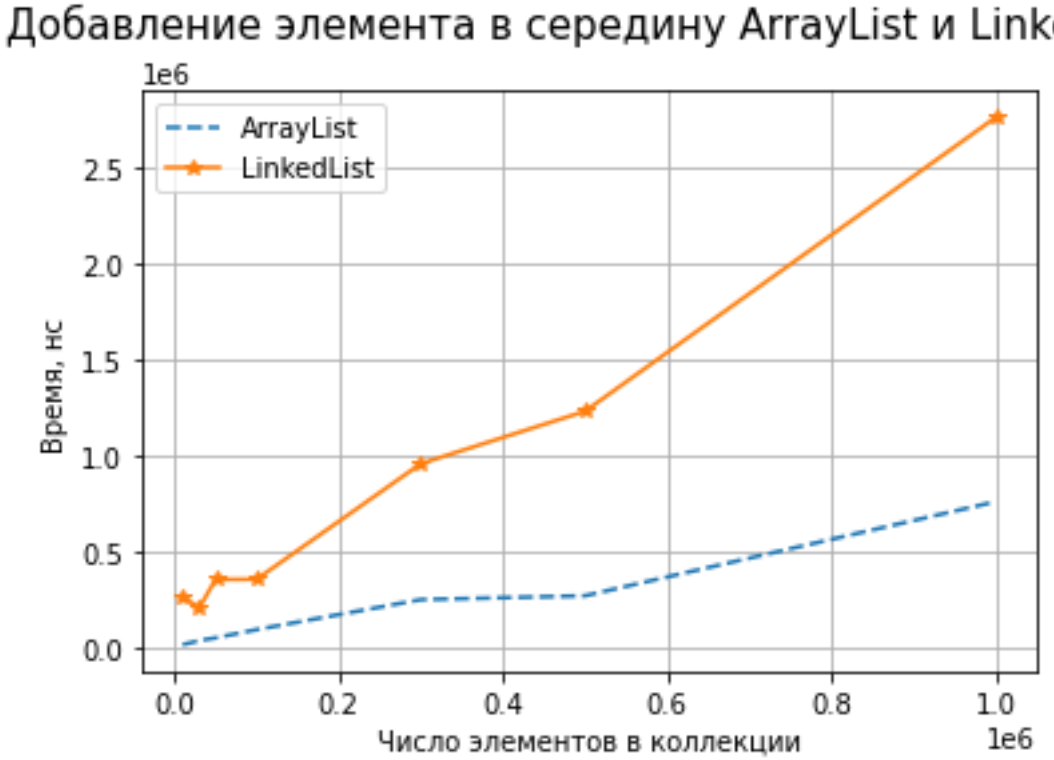
По графикам может показаться, что для больших коллекций MyLinkedList эффективнее, чем встроенная коллекция LinkedList. На самом деле это не так, потому что при повторении вычислений времени с помощью функции *nanoTime*, результаты могут повториться, а могут получиться и противоположными. При этом усреднение не решает проблемы повторяемости результатов. По всей видимости такой разброс связан с оптимизацией кода в Java и jit-компиляцией, а также работой операционной системы в целом. Исходя из этих соображений, можно сделать вывод, что принципиальной разницы между коллекциями нет.

Сравнение производительности основных коллекций в Java.

1. Сравнение ArrayList и LinkedList.

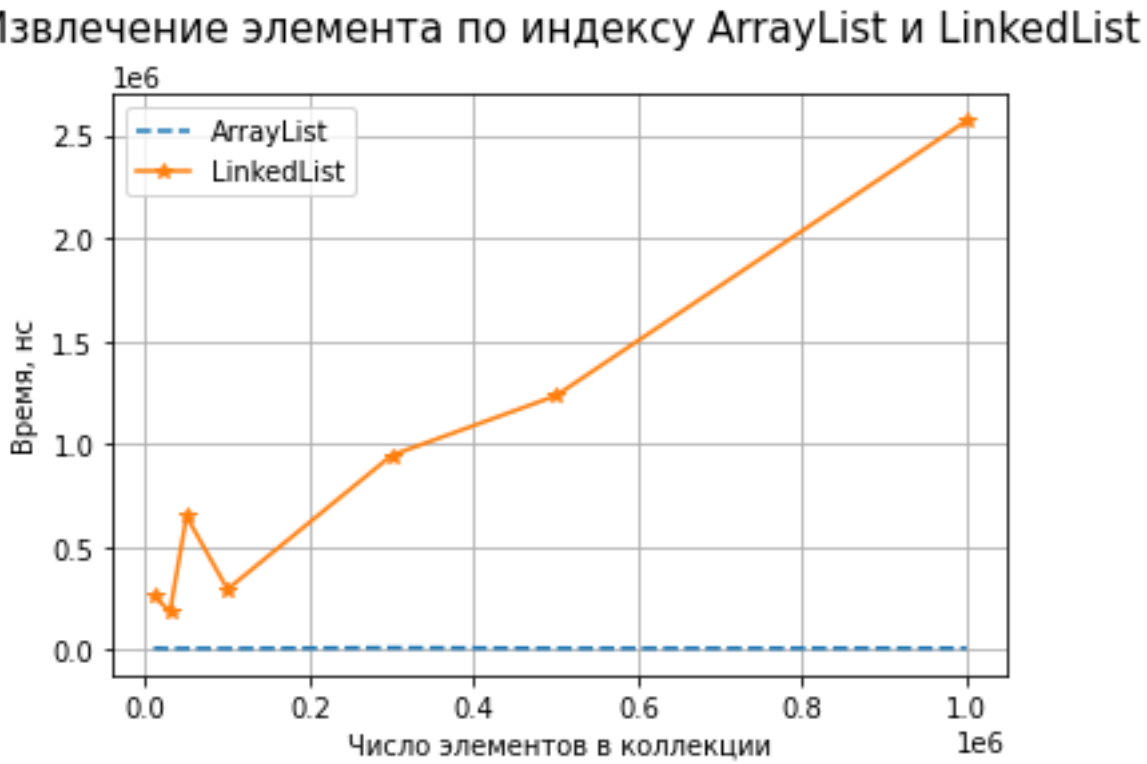


Заполнение числами происходит быстрее для ArrayList, чем для LinkedList, так как отсутствует необходимость создать для каждого элемента ссылки на предыдущий и последующий.

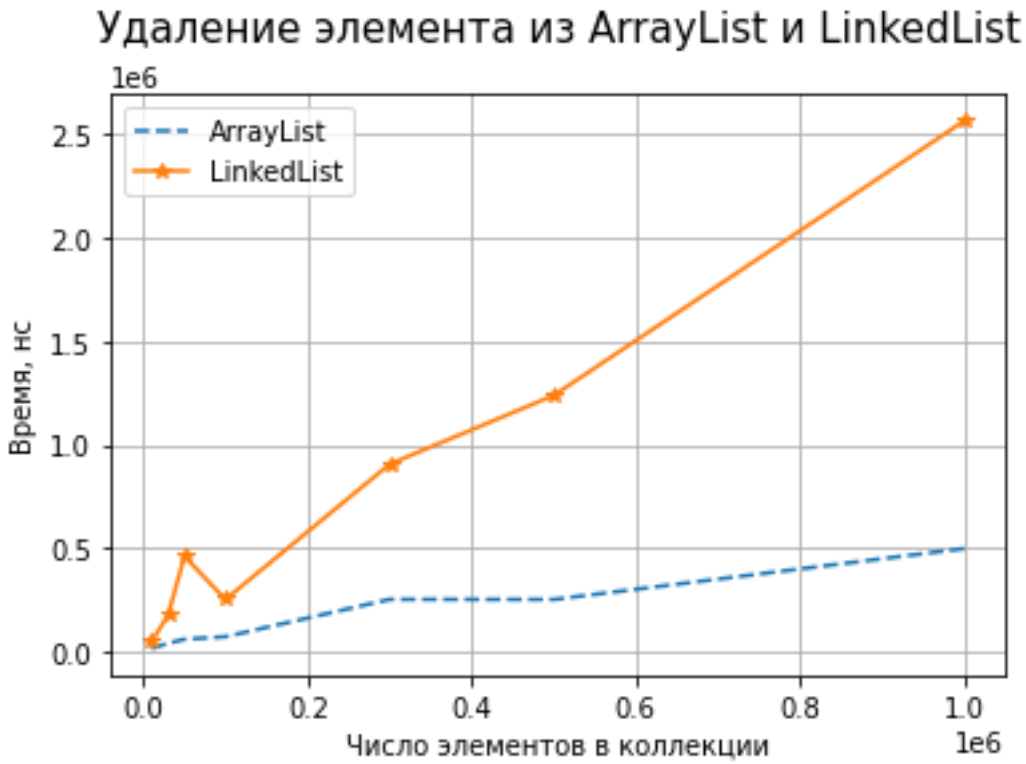


Сложность добавления элемента в середину в LinkedList увеличивается быстрее, чем сложность добавления элементов в середину ArryList, из чего можно сделать вывод, что проще сдвигать элементы, чем менять ссылки.

Добавление элемента в начало и в конец для LinkedList, очевидно, занимает постоянное время, как и добавление элемента в конец ArrayList. Добавление элемента в начало ArrayList занимает большое время, так как нужно сдвигать все элементы.



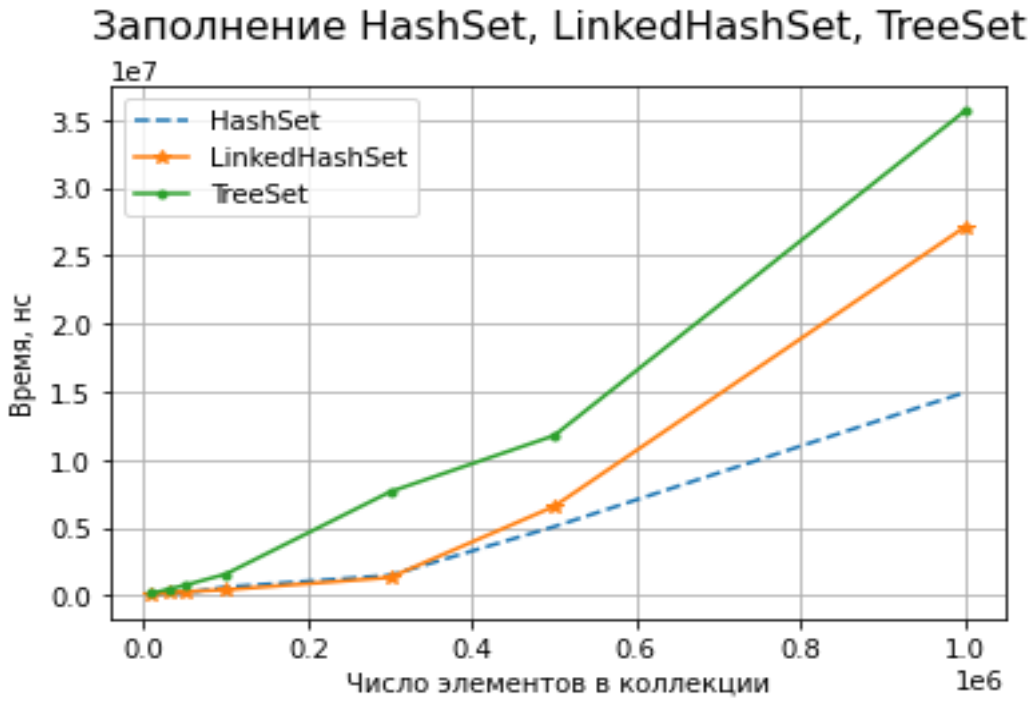
Извлечение элемента по индексу из ArrayList занимает постоянное время, а для LinkedList требует перебора всех элементов, предшествующих необходимому.



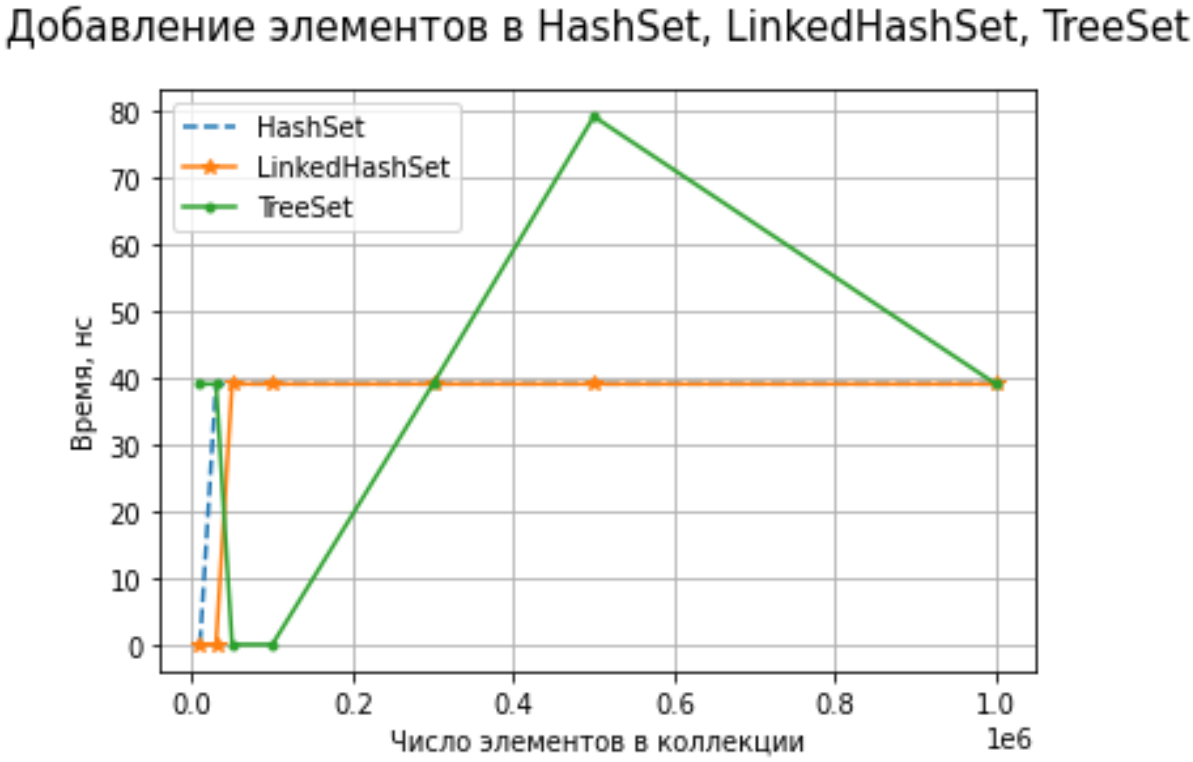
Удалять, соответственно, тоже проще из ArrayList, так как в нем нет необходимости работать со ссылками.

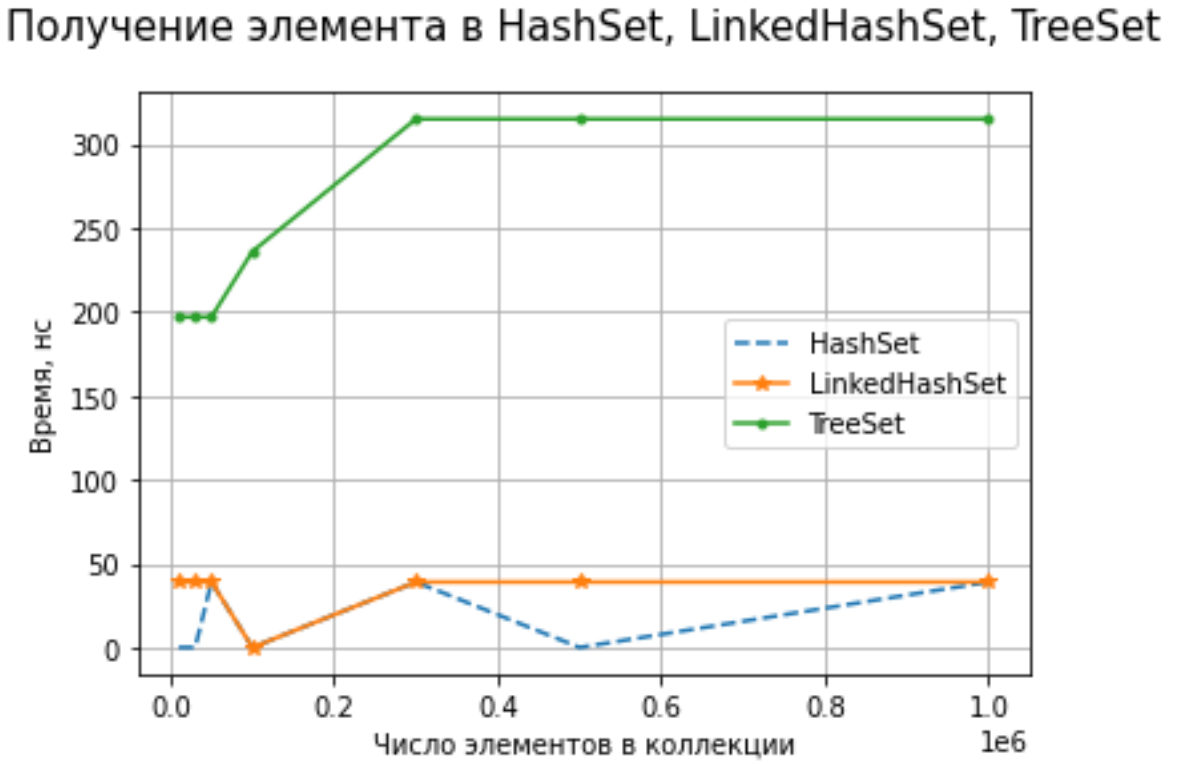
Из указанных выше графиков можно сделать вывод, что в подавляющем большинстве случаев удобнее работать с ArrayList. LinkedList удобен лишь при добавлении элементов в начало коллекции.

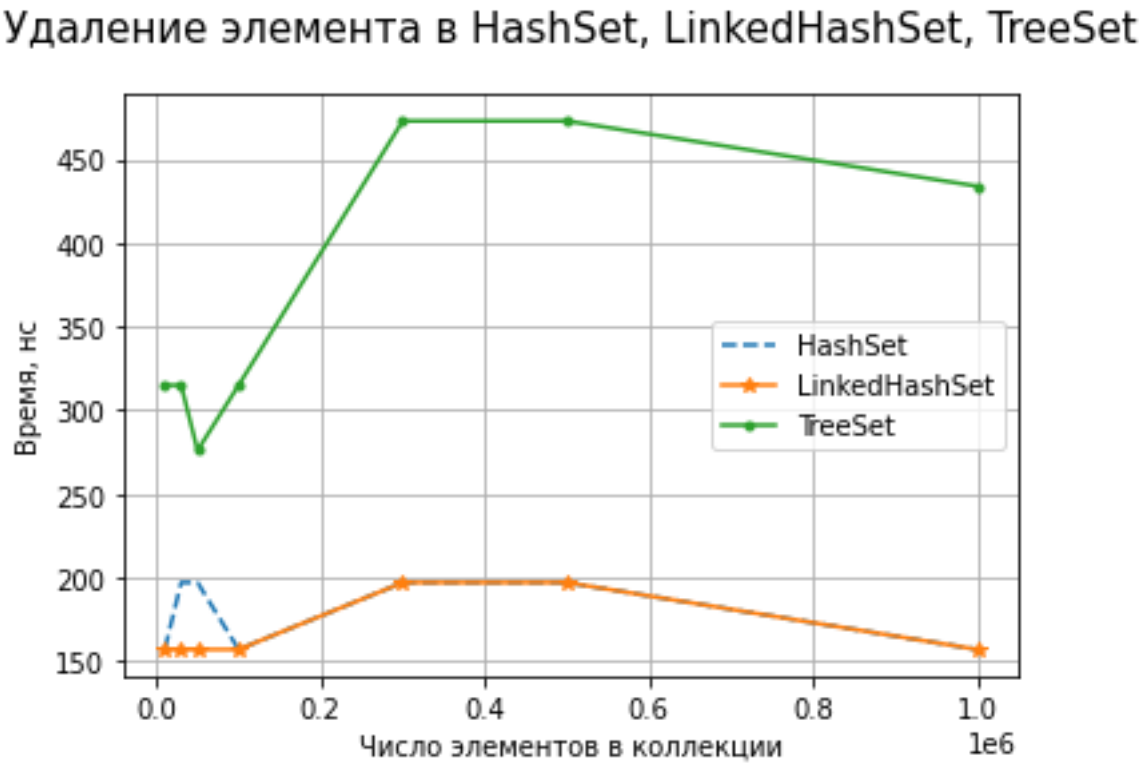
2. Сравнение производительности HashSet, LinkedHashSet, TreeSet.



Добавление элементов в HashSet происходит быстрее, чем в LinkedHashSet и TreeSet, так как нет необходимости создавать ссылки на другие элементы.

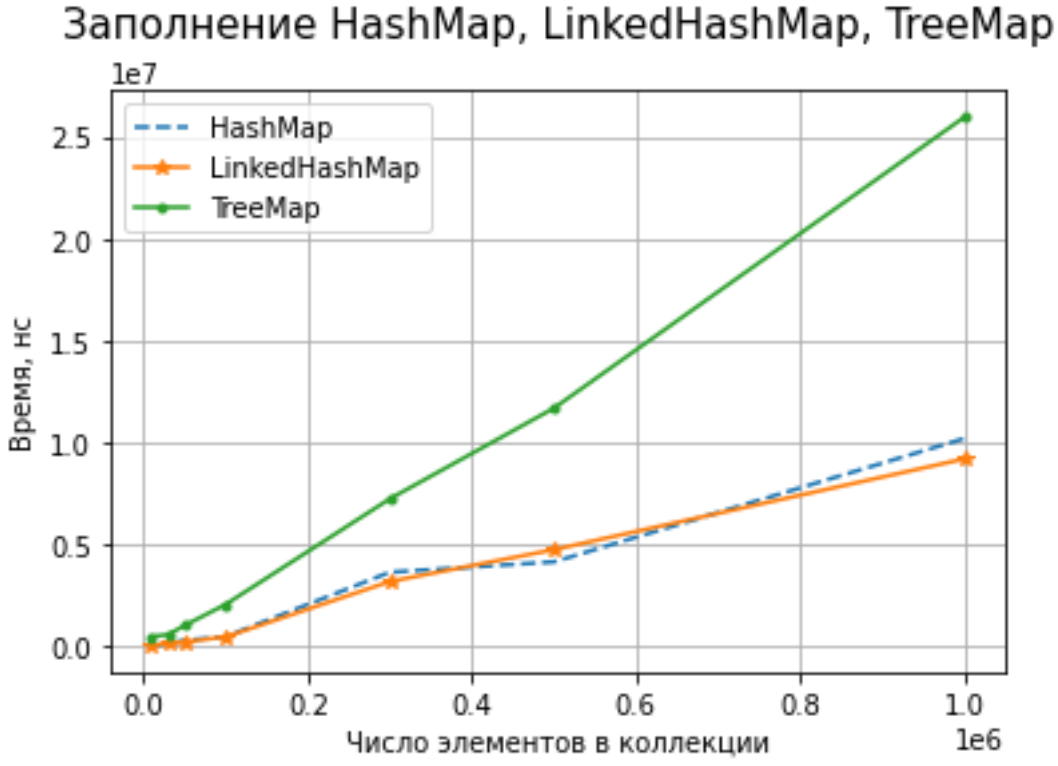




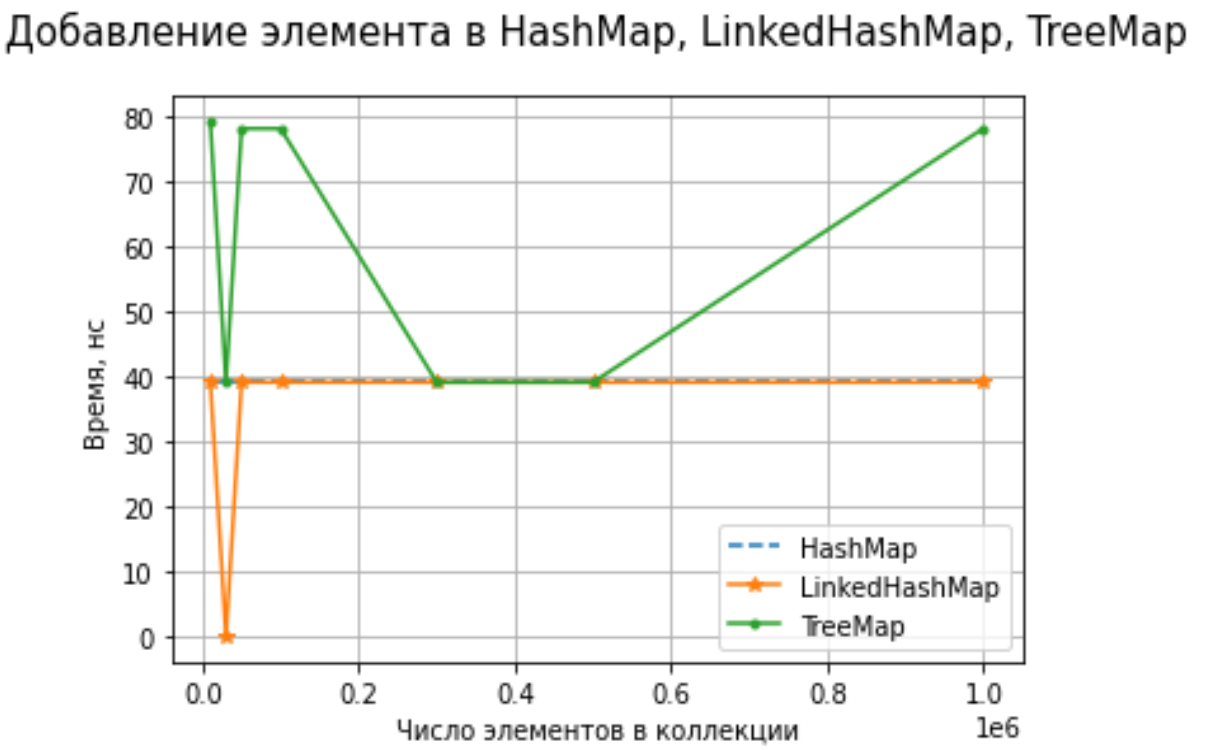


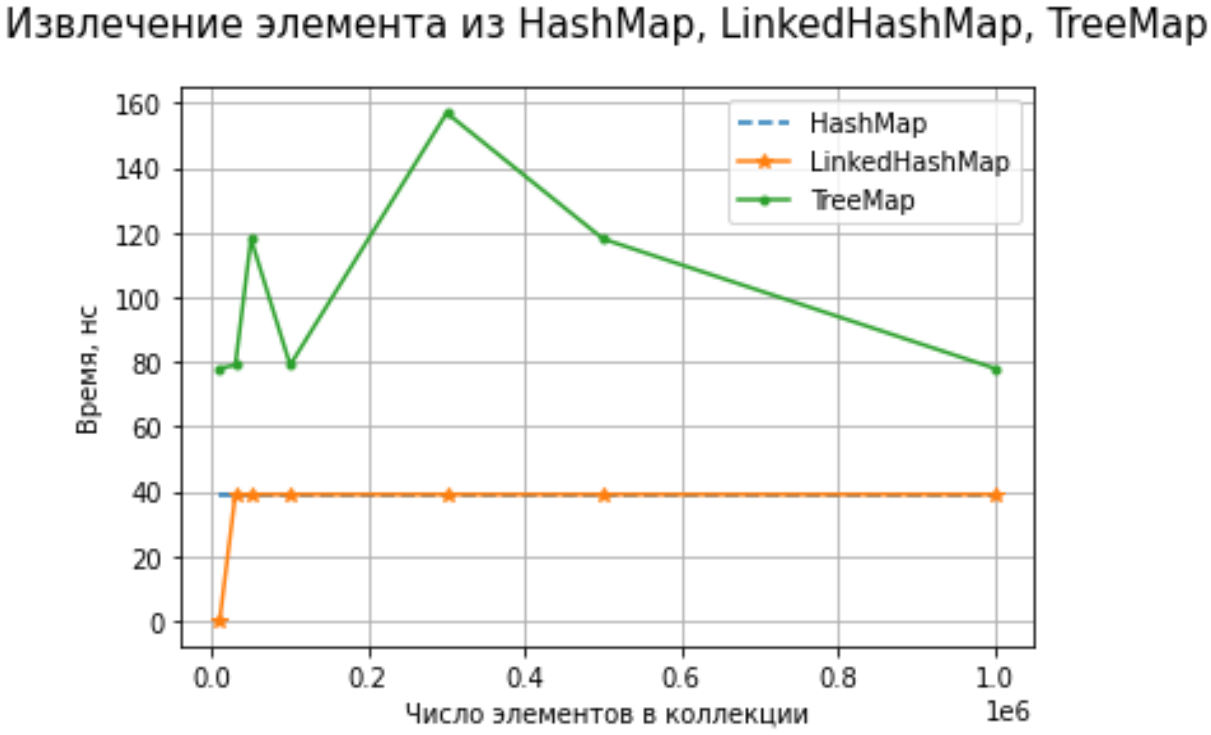
Операции добавления, получения и удаления для HashMap и LinkedHashMap происходят быстрее, чем для TreeMap. Это связано с затратами на формирование связей между узлами дерева и балансировкой.

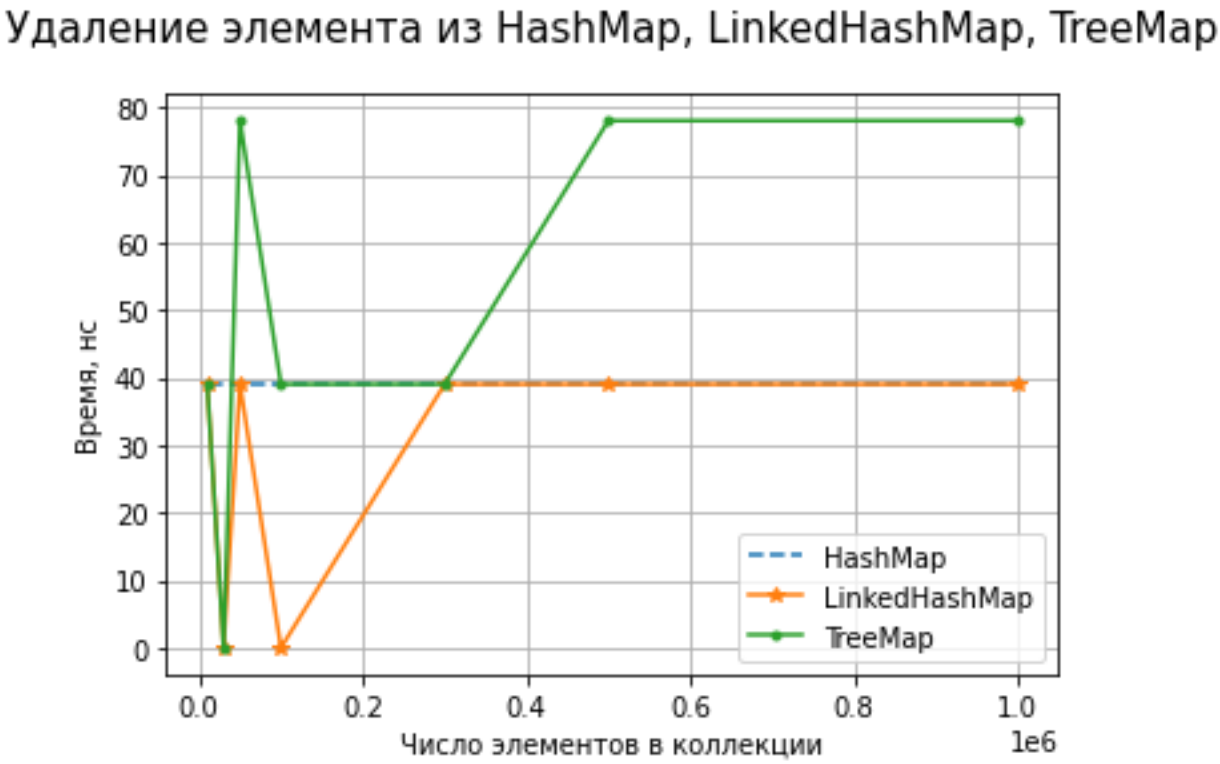
3. Сравнение производительности HashMap, LinkedHashMap, TreeMap



Результат аналогичен коллекциям Set. Сортировка дерева наиболее затратный по времени процесс.







По графикам для добавления, извлечения и удаления элемента из HashMap, LinkedHashMap, TreeMap видно, что время выполнения зачастую для каждой из коллекций близко к постоянному. Время выполнения операций для TreeMap больше, чем для HashMap и LinkedHashMap в результате сортировки и балансировки дерева. Коллекций HashMap и LinkedHashMap примерно равноценный по времени выполнения.

Все графики представляют собой ломаные линии, причем усреднение по количеству повторений однотипных операций не приводит к схождению результата к какому-то определенному значению. Из этого можно сделать вывод, что оценка времени выполнения операций – грубый способ оценки, который позволяет лишь определить существенную разницу между зависимостями.