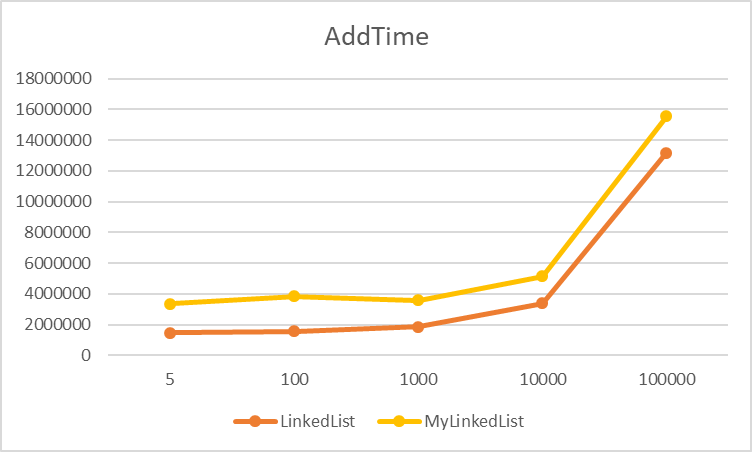
**Сравнительный анализ производительности основных Java коллекций**

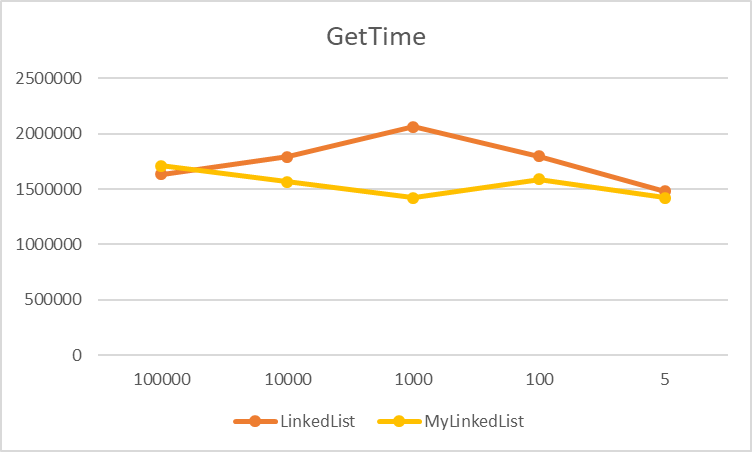
***Сделала Бугаева Алиса***

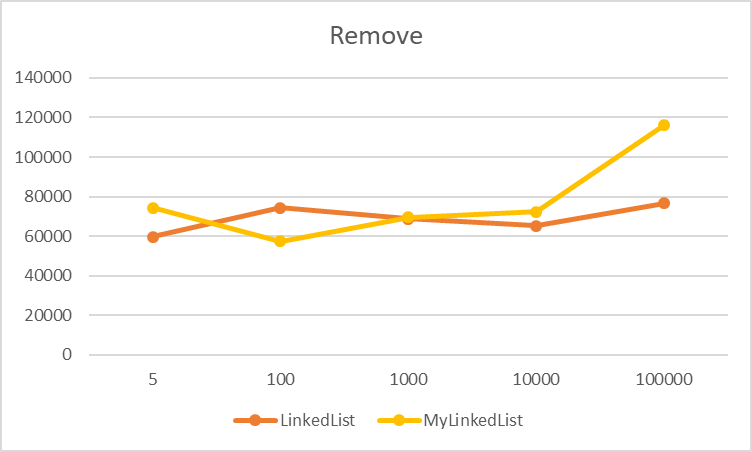
C помощью System.nanoTime() я провела оценку времени выполнения основных операций в Java коллекциях, а так же сравнила свой LinkedList с джавовым. На всех графиках по горизонтали будет отмечено количество элементов, а по вертикали время, измеренное с помощью System.nanoTime().

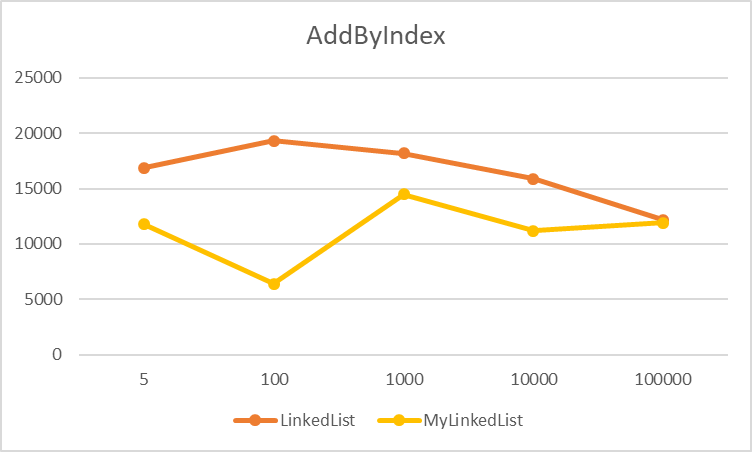
1) My*LinkedList vs LinkedList*

По графикам мы можем видеть, что добавление элементов быстрее сделано в джавовой версии LinkedList. Удаление элементов занимает примерно одинаковое в совокупности время, возможно, в джавовой реализации немного лучше. А что касается получения и добавления элементов по индексу, то быстрее эти операции выполняет MyLinkedList.





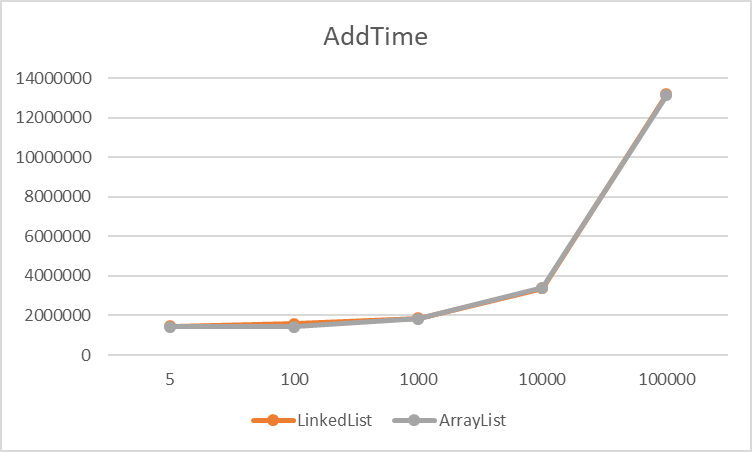




***Вывод:*** Поскольку некоторые операции в моей пользовательской реализации получилось сделать быстрее, можно сделать вывод, что создание собсвенной версии иногда может помочь сделать реализацию какой-то коллекции быстрее для некоторых операций, а значит это неплохая практика.

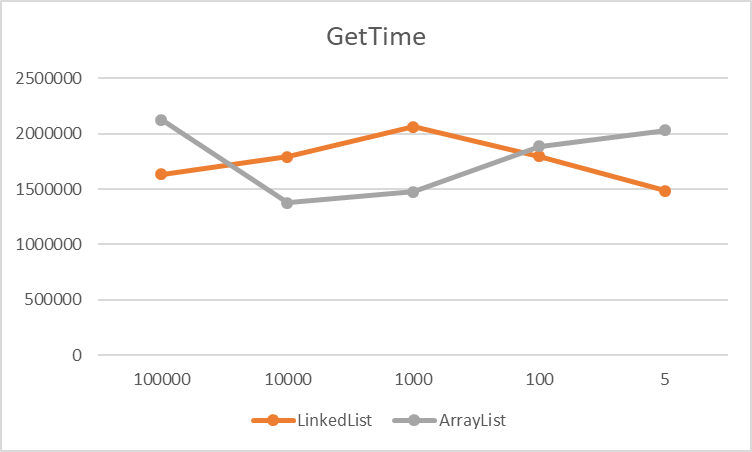
2) *ArrayList vs LinkedList*

Первая операция, которую мы рассмотрим – это добавление элементов в конец.



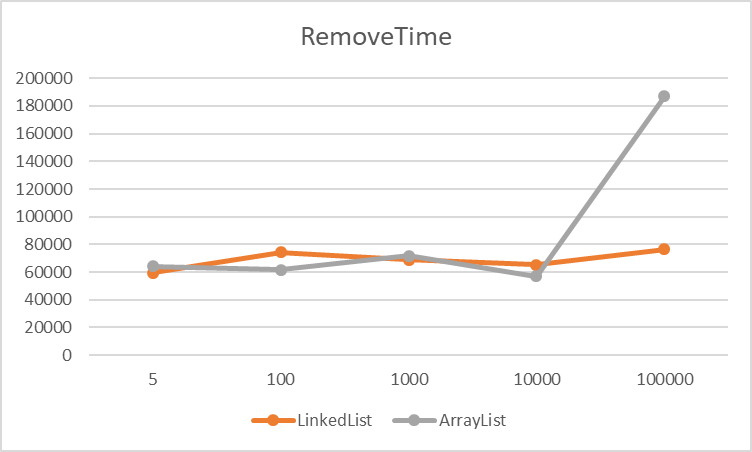
Как можно видеть из графика, производительность этих операций примерно одинакова. Так быть и должно, так как временная сложность вставки в конец в этих видах коллекций одинакова.

Следующей давайте рассмотрим операцию взятия элемента по индексу.



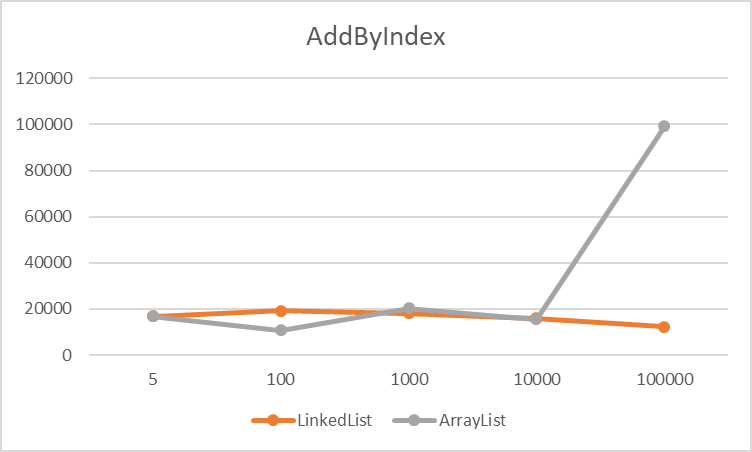
Из графика мы видим, что в зависимости от количества элементов то одна, то другая операция вырывается вперёд. Однако для средних значений количества элементов лучше справляется всё-таки ArrayList. По сути, так быть и должно, ведь временная сложность ArrayList для этой операции это О(1) , а для LinkedList это О(n/4).

Теперь рассмотрим операцию удаления элемента по индексу.



Из графика мы видим, что для среднего количества элементов коллекции работают примерно одинаково. Однако для количества элементов больше 10000 ArrayList почему-то резко начитает показывать намного более плохие результаты. По сути, ArrayList всегда должен быть хуже, ведь временная сложность ArrayList для этой операции это О(n/2), а для LinkedList это О(n/4).

Рассмотрим операцию вставки элемента по индексу.



Из графика мы видим, что в зависимости от количества элементов то одна, то другая операция работает быстрее. Для 100 элементов ArrayList даже справляется с задачей быстрее. Однако для количества элементов больше 10000 ArrayList резко начитает показывать намного более плохие результаты. В этом есть логика, да и вообще, ArrayList всегда должен быть хуже, ведь временная сложность ArrayList для этой операции это О(n/2), а для LinkedList это О(n/4).

***Вывод:*** ArrayList реализован внутри в виде обычного массива, а LinkedList реализован в виде связного списка: набора отдельных элементов, каждый из которых хранит ссылку на следующий и предыдущий элементы. Поэтому при вставке элемента в середину ArrayList, приходится сначала сдвигать на один все элементы после него, а уже затем в освободившееся место вставлять новый элемент. А чтобы вставить элемент в середину LinkedList, достаточно поменять ссылки его будущих соседей. Такую же тенденцию можно наблюдать и с операцией удаления.

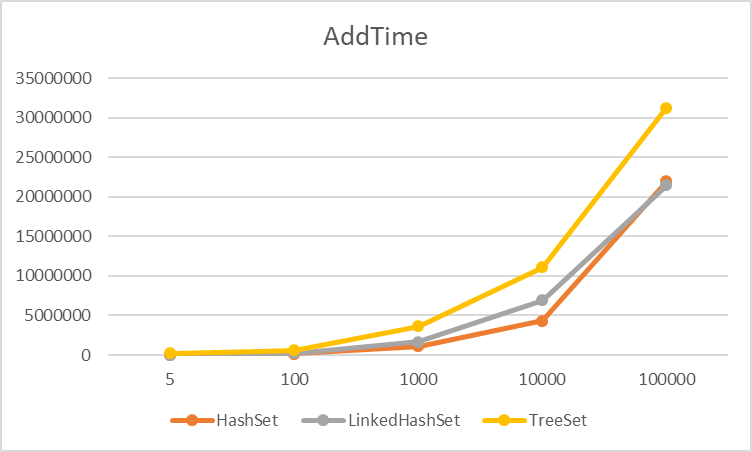
Зато в ArrayList быстро реализованы получение элемента, так как в ней мы просто обращаемся к соответствующему элементу массива.

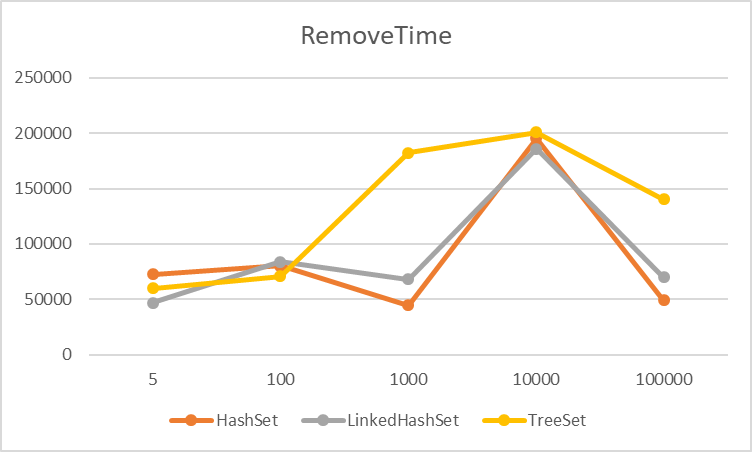
А вот операция добавления элемента работает примерно одинаковое количество времени для обеих операций.

А значит, в зависимости от того, какие операции будут выполняться чаще и следует выбирать один из этих двух видов коллекций.

3) **HashSet vs LinkedHashSet vs TreeSet**

Как можно видеть из графиков для добавления и удаления элементов, HashSet и LinkedHashSet тратят практически одинаковое время на эти операции, в то время как TreeSet затрачивает чаще больше времени, однако это различие тоже не супер велико.

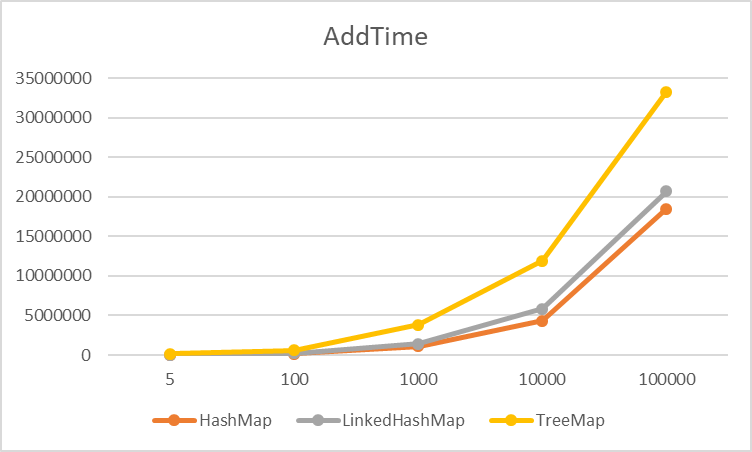


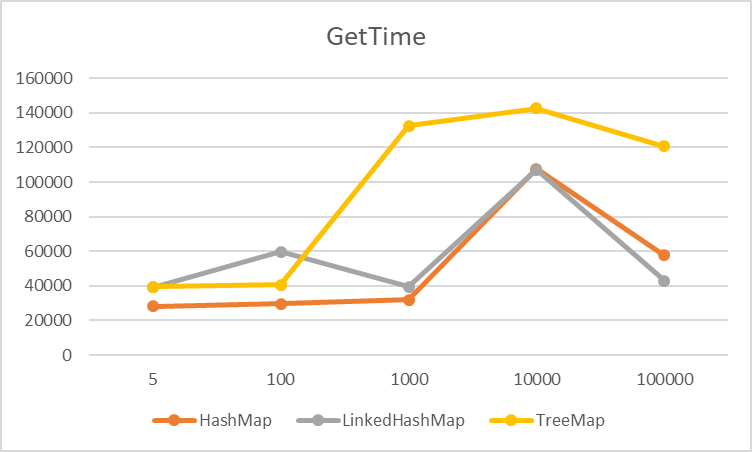


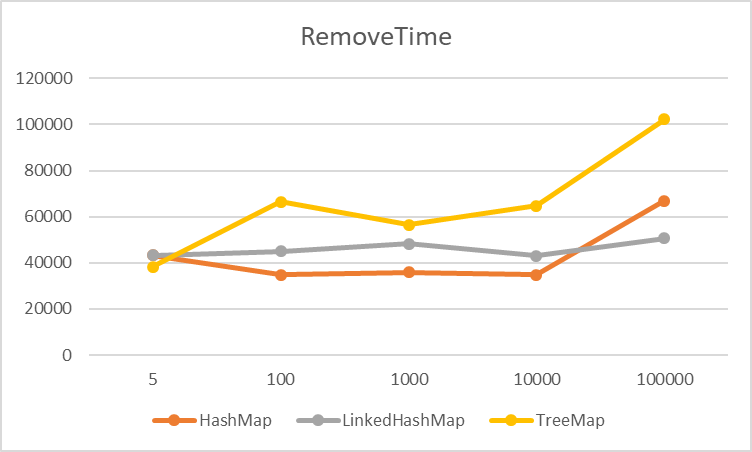
***Вывод:*** Из тестов стало понятно, что по временной сложности эти операции очень похожи, немного отстаёт только TreeSet. Следовательно, при выборе одной из этих трёх коллекций мы должны больше обращать внимание на другие их особенности. Так, LinkedHashSet, в отличие от HashSet поддерживает связный список элементов набора в том порядке, в котором они вставлялись, что позволяет организовать упорядоченную итерацию вставки в набор. TreeSet сохраняет объекты в отсортированном порядке по возрастанию. А значит, если для вас важно, чтобы элементы были отсортированы, то нужно использовать именно его.

4) **HashMap vs LinkedHashMap vs TreeMap**

Несмотря на то, что структура данных коллекций сильно отличается от всех предыдущих, то есть здесь появляется уже пара значений (ключ и значение), можно заметить схожесть с HashSet, LinkedHashSet и TreeSet. То есть HashMap и LinkedHashMap показывают примерно одинаковые результаты времени, а TreeMap в основном несколько отстаёт от них. Это можно поддвердить и оценкой вычислительной сложности операций добавления, удаления и получения элеметов по ключу. Ведь для HashMap и LinkedHashMap она равна О(1), а для TreeMap это O(log(n)).







***Вывод:*** Вывод здесь также схож с 3 пунктом. Из тестов стало понятно, что по временной сложности эти операции очень похожи, немного отстаёт только TreeSet. А следовательно, при выборе одной из этих трёх коллекций мы так же должны больше обращать внимание на другие их особенности. HashMap не дает абсолютно никаких гарантий относительно порядка итераций. Он может (и будет) даже полностью изменяться при добавлении новых элементов. TreeMap будет повторяться в соответствии с "natural ordering" ключей в соответствии с их методом compareTo() (или внешне поставляемым Comparator ). Дополнительно, он реализует интерфейс SortedMap, который содержит методы, зависящие от этого порядка сортировки. LinkedHashMap будет повторяться в том порядке, в котором записи были помещены на карту.