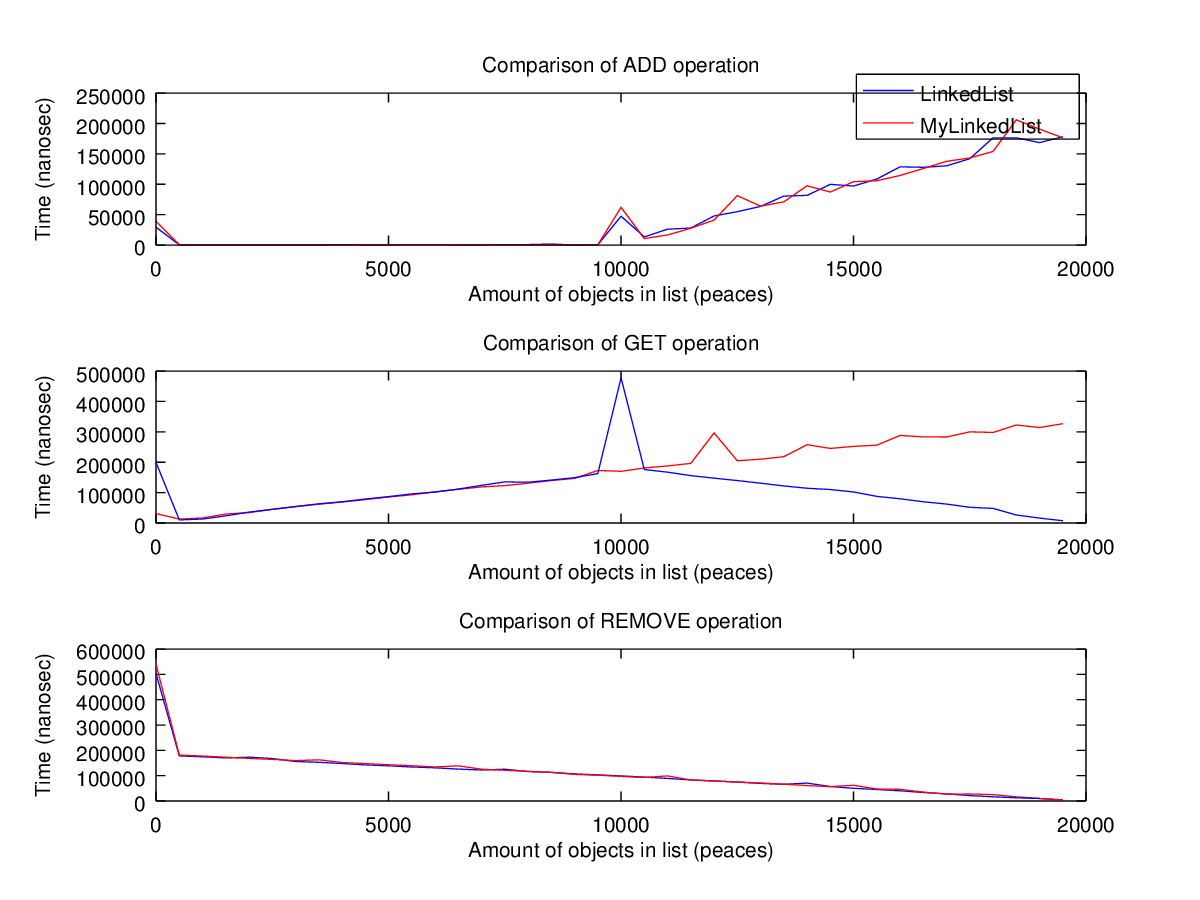
Отчет по домашней работе на тему «Java collections»

**Задание 1.** Сравнение производительности, написанной мной коллекции с java реализацией.

Сравнение делаю по трем основным операциям: добавление, поиск, удаление элемента. Количество элементов в коллекциях 20000. Время выполнения каждой из операций считаю для каждого элемента массива, т.о. получаю 20000 моментов времени для операции ADD, 20000 моментов времени для операции GET, 20000 моментов времени для операции REMOVE. Построил графики зависимости времени выполнения от количества элементов в коллекции для каждой из операций.



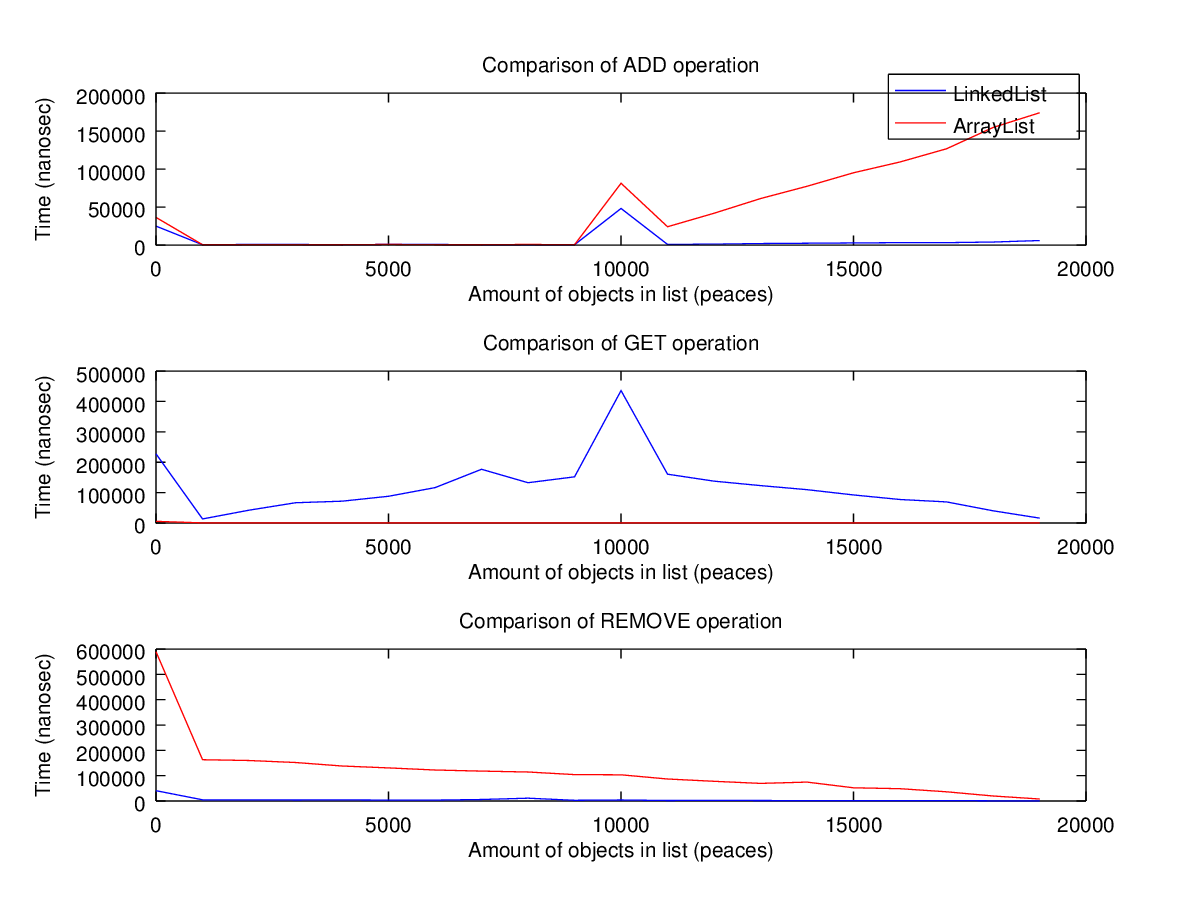
Красным цветом – моя реализация, синим – java реализация.

Как таковых сильных отличий в опрециях add и remove не нашел. Что касается операции get тут есть заметная разница. Затрудняюсь ответить, почему java реализация с числом элементов после половины начинает убывать. В java doc`e (в LinkedList) нашел метод поиска ноды. И она ищется с двух сторон(то есть, если запрашивается элемент, индекс которого больше половины размера коллекции, то нода перебирается с конца). В моем MyLinkedList, такого метода нет. Соответственно, вполне логично, что элемент с большим индексом требует большего времени(график как раз растет).

**Задание 2.**

1. Сравнение производительности ArrayList и LinkedList.





Отличие второго графика от первого в том, что я попытался его проинтерполировать.

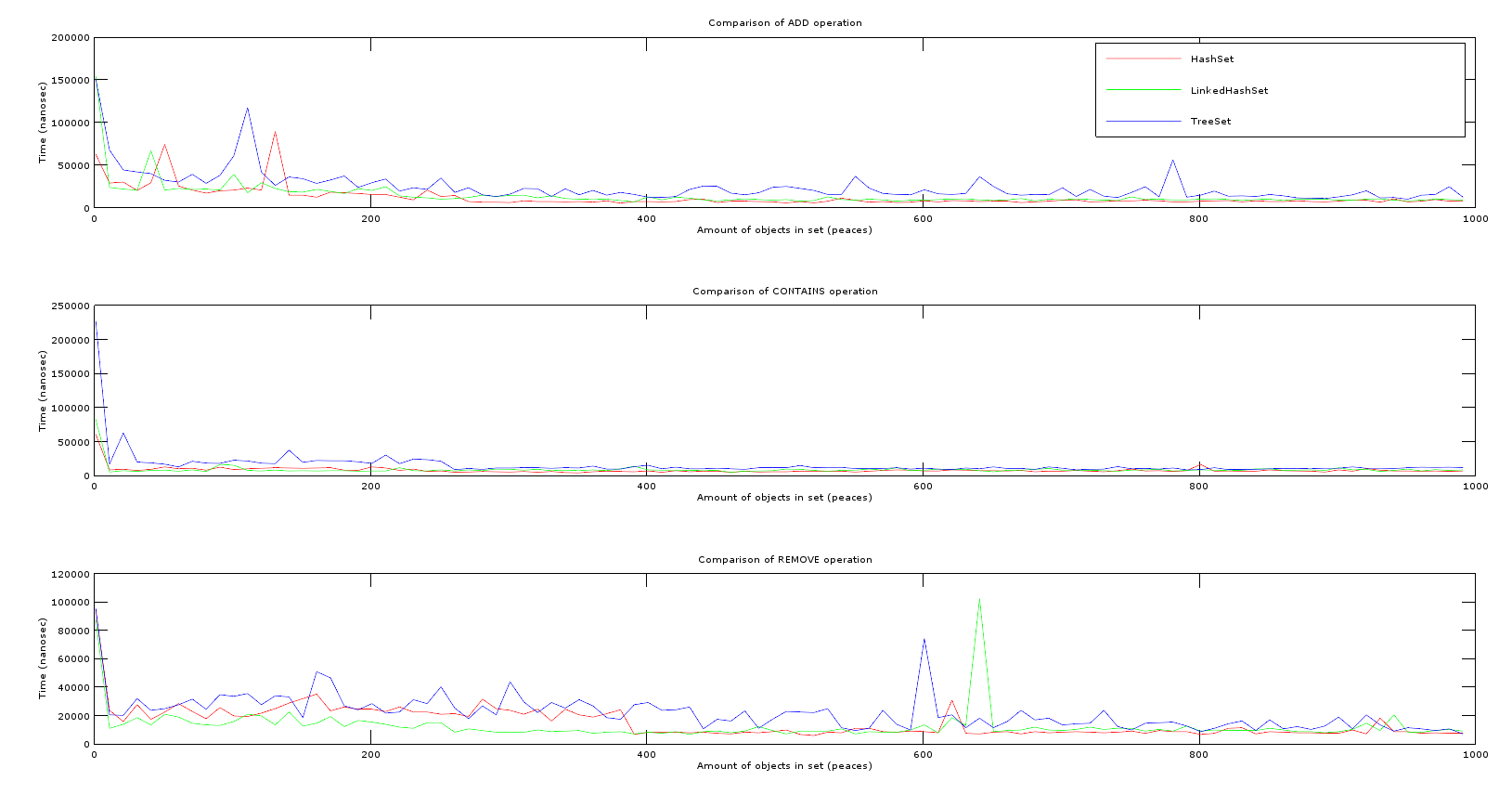
Операция ADD. До 10000 элементов вставка производилась в конец коллекции, соответственно, существенной разницы в скорости между двумя реализациями не наблюдается. Поле 10000 элемента, вставка производилась точно в половинный индекс, то есть на каждой итерации добавлял элемент в позицию, равную [10000]. Отлично видно, что после 10000 элемента, реализация ArrayList`a начинает затрачивать больше времени на выполнение операции. На «последних» элементах, вставка происходила в почти середину коллекции, что приводило к наихудшему времени выполнения для ArrayList`a.

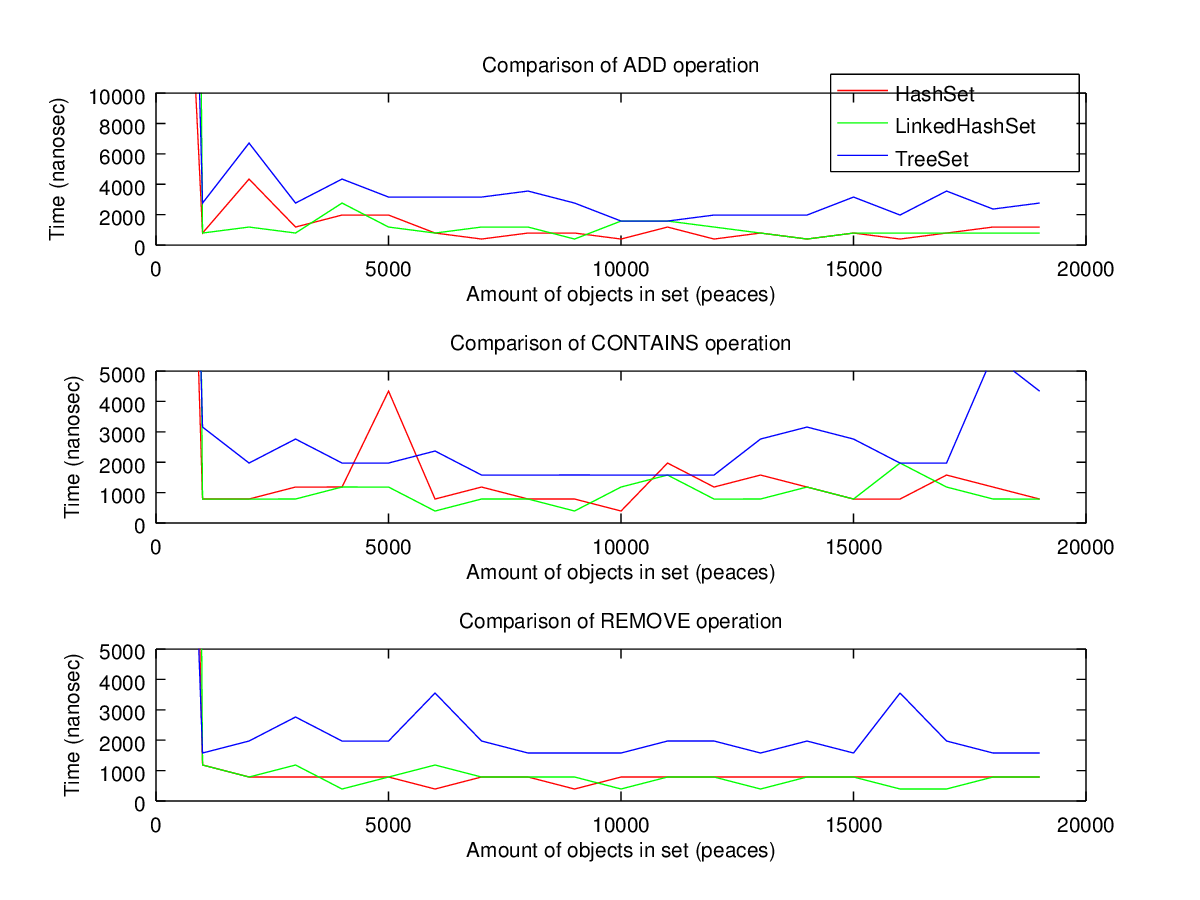
Что касается операции REMOVE, то каждый раз я удалял из коллекции элемент, который лежал в середине коллекции. Известно, что LinkedList имеет сложность O(1) на удаление (как и на вставку). График это подтверждает (примерное равное время). В свою очередь, ArrayList требует больше времени на операцию remove, что также видно на графике.

Операция GET. Известно, что ArrayList имеет сложность O(1) на поиск, а LinkedList O(n).

График подтвержает, что ArrayList имеет преимущество при поиске элемента.

1. Сравнение HashSet, LinkedHashSet и TreeSet.

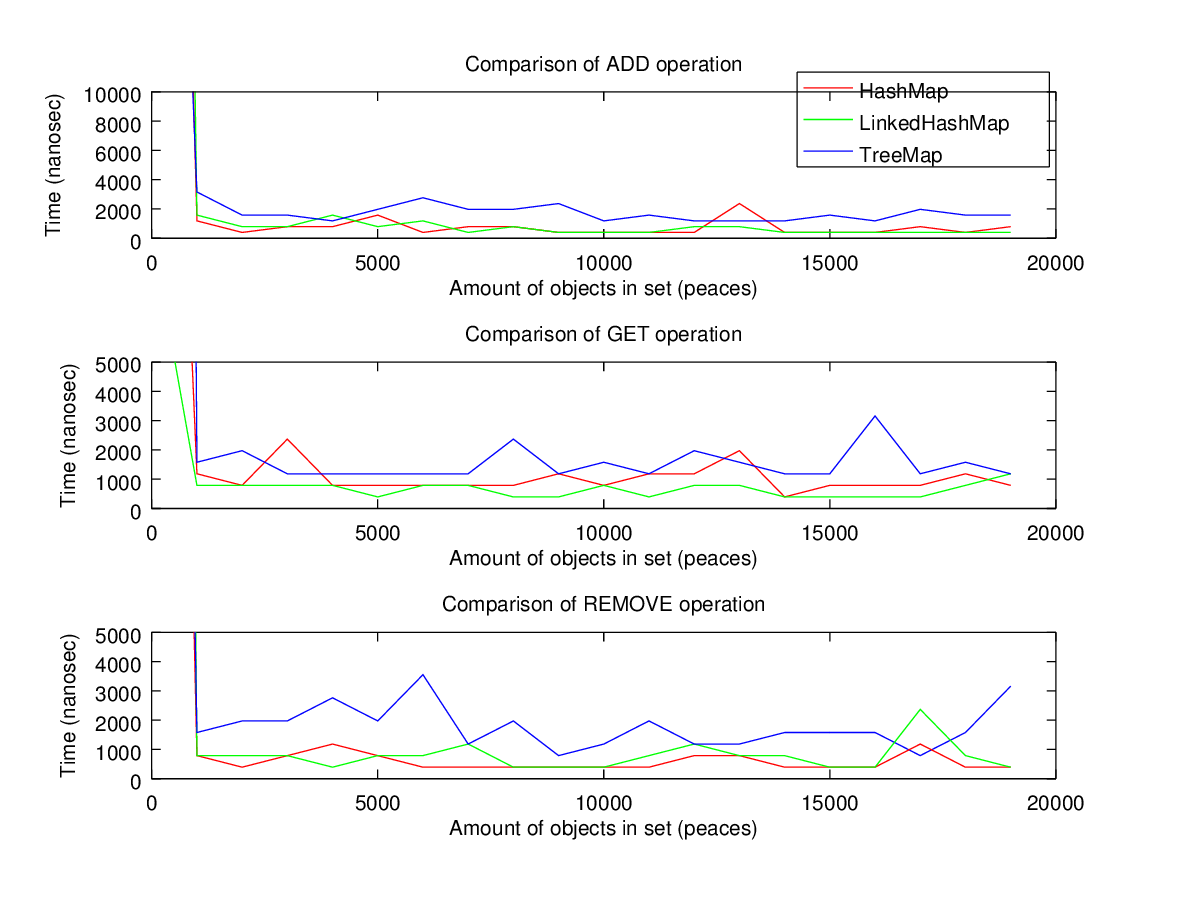




На первой картинке представлены результаты для 1000 элементов. На второй – 20000.

Будем обсуждать последнюю: в первую очередь по графикам видно, что независимо от операции, реализация TreeSet работает медленнее других двух реализаций. Это связано с тем, что она сортирует элементы коллекции и выстраивает красно-черное дерево. Сложность всех операция O(log n), что достаточно быстро, но медленне, чем O(1), которой обладают реализации HashSet и LinkedHasSet. Учитывая тот факт, что измерение ведется в наносекундах, эти реализации, согласно графику, имеют схожее время выполнения для каждой из операций.

1. Сравнение HashMap, LinkedHashMap и TreeMap.



Из графиков видно, что, опять же таки, TreeMap работает медленне, т.к. сортирует элементы коллекции и основана на красно-черном дереве. LinkedHashMap и HashMap затрачивают меньшее время для операций add, get, remove, чем TreeMap.