Итак, мы создали свою коллекцию MyList

Сравним ее производительность с производительностью ArrayList и LinkedList.

попробуем для начала **метод add(),** сделаем вставку в начало 4 элементов типа Integer. Время будет в наносекундах.

Время:

* MyList: 1331300
* ArrayList: 38400
* LinkedList: 90800

Теперь будем следовать заданию. Вставка 1000 элементов типа Integer.

Время:

* MyList: 1078400
* ArrayList: 592800
* LinkedList: 434900

Довольно интересно. MyList сработал даже быстрее, нежели на 4 элементах. Я думаю, можно списать на то, что время в наносекундах

Upd. Разброс чисел какой-то неадекватный. Попробую сделать для 100к элементов. Будем брать 10 результатов и выводить среднее. Если заглянуть в приложение, то можно увидеть, что в среднем наша коллекция отрабатывает быстрее, нежели LinkedList, но не так шустро, как это делает ArrayList. Кстати, зеленым я выделил все, что связано с Integer.

Что-то мы задержались на вставке. **Перейдем к вставке в конец**.

Итак, LinkedList со вставкой в конец справился идеально, относительно других сравниваемых коллекций. Тут наша коллекция явный аутсайдер. Да и разброс у нее по времени выполнения больше, если смотреть на другие коллекции.

**Поиск по индексу** тоже показал, что наша коллекция далеко не лидер, но зато лишний раз на практике убедились, что get() работает на порядок быстрее в ArrayList, нежели в LinkedList. Тут, я думаю, не нужно объяснять, почему это так 😊.

Приступим к сравнению **метода remove().**

В нашем случае результаты получились не такими жуткими и удручающими, как это было раньше. Нам удалось даже обогнать по времени LinkedList. И график получился ровнее. Относительно всего остального. Скорее всего, если еще раз перезапустить скрипт, то нас встретят не такие радужные цифры. Поэтому результат не стоит считать объективным, как и везде.

Теперь заменим Integer на Circle. Ожидаются результаты, которые будут аналогичны тем, что было с числами, только элементов возьмем поменьше, ибо вывода ждать не очень весело, а результат сходу увидеть хочется. Тут я все сравнения буду делать в классе MainTest, тут уже с рефакторингом. Main оставил специально, там Integer сравнение.

Тут почти полное дублирование предыдущих пунктов, только с другим типом данных. Результаты тоже будут в приложении, выделил голубым.

**Добавление в начало.**

Тут все аналогично той картине, что мы видели с целочисленными типами, даже никаких выводов дополнительных сказать не могу

**Добавление в конец.**

Тут все несколько интереснее. Наша коллекция, которая была явным андердогом в данном виде соревнований, заняла 2 строчку в средней скорости работы в конкретной номинации. А LinkedList, выступавший с позиции явного фаворита, отработал дольше всех остальных сравниваемых коллекций.

**Поиск по индексу.**

Тут все аналогично результатам для Integer. ArrayList на несколько порядков быстрее работает относительно других коллекций.

**Удаление элемента.**

Результат очень схож с тем, что мы видели в методе get(), однако тут разница выражена не так явно, да и вообще скорость выполнения операций заметно ниже.

Кстати, я попытался один раз использовать remove(), чтобы циклом очистить коллекцию. Работает неприлично долго. В Main под цифрой 6 это показано, так что get ready!

Пункт первый совпадает с пунктом со вторым для подпункта а, поэтому только вывод.

2.а. Лучше всего себя показал ArrayList в случае, когда вставка идет в начало коллекции.

Также ArrayList просто на порядок быстрее работает, если речь заходит об удалении объекта или же поиска по индексу.

2.б. Результаты в табличке оранжевым выделены. Вставка в начало для HashSet, LinkedHashSet, работают с одинаковой скоростью, а TreeSet значительно уступает им в производительности. Если вставка идет в конец, то тут картина повторяется, однако разница в производительности меньше. Еще я попробовал метод size(). Разница на уровне погрешности, поэтому без разницы.

2.в. Здесь я буду сравнивать методы put(), и size(). Выделил розовым в приложении. Size() , как оказалось, абсолютно нет смысла сравнивать, ибо там все плюс-минус одинаково. Put() самый долгий у LinkedHashMap. Между TreeMap и HashMap большой разницы не вижу.

P.S. как я понял, в нанотайм методе смысла было мало, ибо разброс большой. К тому же там, я думаю, зависит результат еще и от gc, ибо на графике можно прекрасно увидеть, что там подъемы и спады цикличны. Будто он до момента определенного работает, потом gc вычищает все, после чего скорость работы увеличивается. Если я не прав, то напиши это, пожалуйста, в фидбэке.