**Выводы по 2 заданию 4-ого дз по Java**

1. **Сравнение ArrayList и Linked list: (100000 элементов)**

filling ArrayList with 100000 elements [AL] --> 73346906

filling LinkedList with 100000 elements [LL] --> 33177892

adding elements in the middle of ArrayList [AL] --> 19681131

adding elements in the middle of LinkedList--> [LL] 754390483

contains element near the end [AL] --> 11617036

contains element near the end [LL] --> 7543780

indexOf element near the end [AL] --> 3057507

indexOf element near the end [LL] --> 1473550

get in the middle of ArrayList [AL] --> 22071218

get in the middle of LinkedList [LL] --> 40202208522

removing elements one by one in ArrayList [AL] --> 10202266

removing elements one by one in LinkedList [LL] --> 47423869

LinkedList гораздо выгоднее для добавления элементов в конец списка (работает в 2 раза быстрее), в среднем в 1,5-2 раза лучше отрабатывает метод contains и indexOf.

ArrayList быстрее на порядок отрабатывает добавление элементов в середину списка, в 4,5 раза быстрее работает с удалением элементов из списка и примерно в 2000 раз быстрее получает конкретный элемент по индексу.

Вывод: для простого накапливания элементов, проверки наличия определенных элементов без частого доступа к ним лучше использовать LinkedList. Для остальных задач целесообразнее ArrayList (особенно выигрывает при адресации по индексу и если нужно вставлять элементы в середину списка и удалять некоторые элементы периодически).

1. **Сравнение** **HashSet, LinkedHashSet и TreeSet: (10000 элементов)**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

HashSet [add-filling] --> 7768328

LinkedHashSet [add-filling] --> 7413469

TreeSet [add-filling] --> 23885010

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

HashSet [add-insert ] --> 15796657

LinkedHashSet [add-insert ] --> 2158698

TreeSet [add-insert ] --> 1966807

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

HashSet [contains ] --> 3016454

LinkedHashSet [contains ] --> 2370805

TreeSet [contains ] --> 6576860

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

HashSet [remove ] --> 4967089

LinkedHashSet [remove ] --> 2500184

TreeSet [remove ] --> 2971981

TreeSet характеризуется замедленным заполнением множества с нуля и медленным поиском конкретного элемента, быстрой вставкой в уже заполненное множество, и быстрым удалением

HashSet – быстрое заполнение с нуля, медленная вставка в уже имеющееся множество, отличная скорость поиска элемента, медленное удаление

LinkedHashSet – самое быстрое заполнение с нуля, быстрая вставка, быстрое удаление, быстрый поиск – все результаты лучшие (быстрая вставка чуть-чуть отстает)

**с. Сравнение** **HashMap, LinkedHashMap и TreeMap: (10000 элементов)**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

HashMap [add-filling] --> 5132545

LinkedHashMap [add-filling] --> 6530209

TreeMap [add-filling] --> 12998528

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

HashMap [add-insert ] --> 2375781

LinkedHashMap [add-insert ] --> 17805449

TreeMap [add-insert ] --> 10920381

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Value found? -> true

HashMap [contains(v)] --> 54004150

Value found? -> true

LinkedHashMap [contains(v)] --> 1632474

Value found? -> true

TreeMap [contains(v)] --> 2741836

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Key found? -> true

HashMap [contains(k)] --> 527468

Key found? -> true

LinkedHashMap [contains(k)] --> 707852

Key found? -> true

TreeMap [contains(k)] --> 423591

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

HashMap [replace ] --> 3004948

LinkedHashMap [replace ] --> 3881053

TreeMap [replace ] --> 36520593

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

HashMap display time --> 12116202

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

LinkedHashMap display time --> 5799966

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

TreeMap display time --> 10665044

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

HashMap [remove ] --> 4578642

LinkedHashMap [remove ] --> 3879187

TreeMap [remove ] --> 20474820

HashMap хорош для заполнения, вставки нового элемента по ключу, поиску по ключу, неплохо отрабатывает удаление элементов из списка по ключу. Но что касается value – тут беда. Не рекомендуется для поиска по значению и для доставания значений из map-а.

LinkedHashMap хорош для начального заполнения, поиску по value, неплох для замены значений по ключу. Отлично отрабатывает при переборе всех значений из map (вывод на экран). Лучший в очистке всех элементов map. Когда важен порядок ввода – он твой бро.

TreeMap хорош для поиска по value и по key – отлично подойдет как база знаний, чтобы консультироваться по этим значениям. Ужасен в обновлении – вставки и удаления не ок. Пользоваться, но не изменять – его конёк.