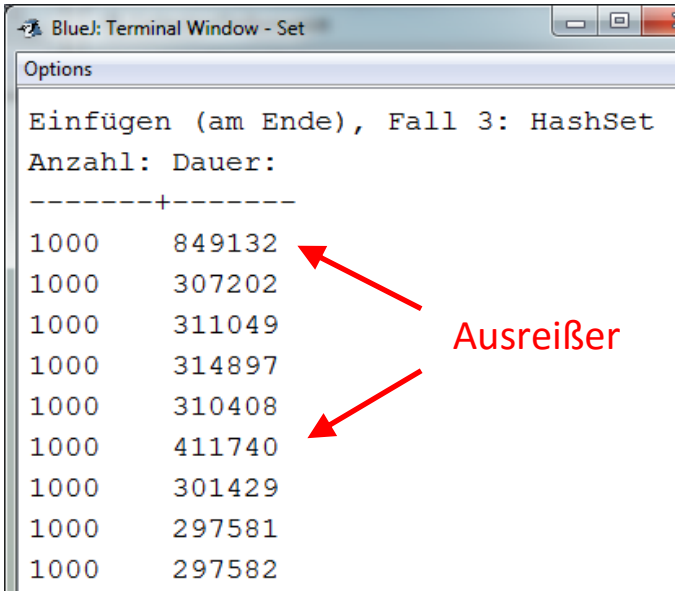
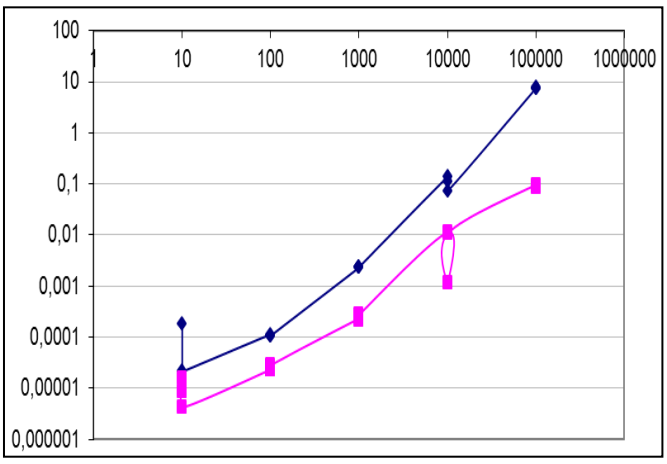
AU06 – Performancevergleiche Collections

Die folgenden Messungen sollen mit Hilfe der Methode `system.nanoTime()` durchgeführt werden. Diese Methode liefert den aktuellen Zeitstempel in der höchsten verfügbaren Genauigkeit als `long`-Wert zurück. Durch mehrfachen Aufruf bzw. Subtraktion der Rückgabewerte kann die Laufzeit eines Programmteils bestimmt werden.

Vergleiche die Laufzeit für das Einfügen von Elementen des Typs `Integer` jeweils bei `ArrayList`, `LinkedList`, `HashSet`, `TreeSet`, `HashMap` und `TreeMap`. Führe die Messung für 10, 100, 1.000, 10.000, 100.000 und 1,000.000 Elemente durch. Wiederhole jede Messung mehrere Male, um Messergebnisse ausschließen zu können, die durch externe Einflüsse (Speicherverwaltung, Betriebssystem) verfälscht wurden. Dokumentiere den Verlauf der Laufzeiten als Tabelle und als Grafik (z.B. mittels Excel).






```
public class CollectionPerformance {
    public static void messung(int anzahl) {
        HashSet<Integer> hs = new HashSet<Integer>();
        long startZeit = System.nanoTime();
        for (int i = 0; i < anzahl; i++) hs.add(new Integer(i));
        long dauer = System.nanoTime() - startZeit;
        System.out.println("" + anzahl + "\t" + dauer);
    }
    public static void main(String[] args) {
        System.out.println("Einfügen (am Ende), Fall 3: HashSet");
        System.out.println("Anzahl:  Dauer:");
        System.out.println("-------+-------");
        messung(1000);
        messung(1000);
        messung(1000);
        . . .
```

Führe die Messungen für folgende Szenarien durch:

- **Einfügen** in die Collection:
    - **AU06a** ArrayList / LinkedList: Einfügen unterschiedlicher Objekte vom Typ Integer jeweils mit einem **sequenziell aufsteigenden Wert** von 0…anzahl **am Ende** der Liste.
    - **AU06b** ArrayList / LinkedList: wie AU06a, aber **am Beginn** der Liste.
    - **AU06c** ArrayList / LinkedList: wie AU06b, aber jeweils mit einem **zufälligen Wert** im Intervall 0…anzahl **an zufälliger Stelle** (z.B. index = Wert!) der Liste.
    - **AU06d** HashSet / TreeSet: Einfügen unterschiedlicher Objekte vom Typ Integer jeweils mit einem **sequenziell aufsteigenden Wert** von 0…anzahl.
    - **AU06e** HashSet / TreeSet: wie AU06d, aber mit **einem zufälligen Wert** im Intervall 0…anzahl.
    - **AU06f** HashMap / TreeMap: Einfügen unterschiedlicher Objekte vom Typ Integer jeweils mit einem **sequenziell aufsteigenden Key** von 0…anzahl. Der **Value** soll immer ein **leerer String** sein.
    - **AU06g** HashMap / TreeMap: wie AU06f, aber der **Value** soll als **String aus dem Key** gebildet werden.
    - **AU06h** HashMap / TreeMap: wie AU06g, aber der **Key** soll ein **zufälliger Wert** im Intervall 0…anzahl sein.
- **Lesen** in der Collection:
    - **AU06i Sequenzielles Auslesen aller** Objekte, bei ArrayList mittels for-Schleife, bei LinkedList mittels while-Schleife und next(), bei HashMap und TreeMap mittels Iterator und keySet(), bei HashSet und TreeSet mittels Iterator.
    - **AU06j Auslesen einiger** Objekte über einen **zufälligen Index / Key**.
- **Löschen** in der Collection:
    - **AU06k Sequenzielles Löschen aller** Objekte; siehe Hinweise bei AU06g.
    - **AU06l Löschen einiger** Objekte über einen **zufälligen Index / Key**; siehe Hinweise bei AU06h.